Using Gesture Detection as a User Interface for Customized Earphones

Fabian Kortekaas

September 10, 2021

TU Delft Supervisors:

Ger de Graaf & Paddy French

Supervisors at Dopple:

Jaap Haartsen & Frank Hooijschuur

In partial fulfilment of the requirements for the degree of

Master of Science

Biomedical Engineering

at the Delft University of Technology

Preface

Host Organization

The project is offered to me by the TU Delft in collaboration with Dopple. Dopple was founded in 2017 and is located in Assen, the Netherlands. The company focusses on the production of wearable and wireless electronics, like Bluetooth headsets and earphones.

Student Motivation

I am a student in biomedical engineering that chose the track of medical devices and bioelectronics. My interest lies in combining my experience in software engineering with the knowledge of Human movement sciences and electrical devices to create new products. Dopple provided an opportunity to do a graduation project in this area. The project is made up of 32 ECTS.

Goal of the Project

The goal of this project is to come up with and test a new idea for a user interface (UI) applied in wireless earphones/headsets. This user interface will make it possible to control regular audio player actions, by using hand gestures. Software will be written for a microcontroller to extract data from sensors that have the potential/ability to recognize gestures.

Value of the Project

Remote sensing is becoming more popular and is applicable in many fields. Removal of controller equipment and replacement with the human body itself as the controller is a practical and also very time efficient procedure. A good example is the Kinect application used by the Xbox manufacturers [1], where infra-red (IR) cameras capture the movements of players to convert them into in-game actions. The idea here is to implement an IR sensor in order to recognize a select amount of hand gestures. These gestures can be used to perform different actions on a device related to audio players. There are sensors for moving gestures, like for example a swipe forward, that indicates that the user wants to go to the next song. However, after conducting a search for non-moving gesture sensing there is no application/sensor found. If this UI would be applied for head-and earphones this action has to be transferred to a computer or mobile phone (the media player), this means that there has to be a microcontroller in between the sensor and the device, for receiving, processing and transferring the data. Because the headphones are rather small, the sensor and the board that it is placed on should be small and also flexible. Next to that, small devices also have a limited power supply. There is thus a need for sensors and controllers with a low power consumption. When successful, this functionality can be applied to more than just headphones. It can be used on any device that has user-actions that have to be easily accessible.

Acknowledgements

I would like to thank my supervisors Ger de Graaf and Paddy French for guiding me through the thesis. I also want to thank Jaap Haartsen and Frank Hooijschuur, for giving me the opportunity to work on this project. It was a pleasure to attend our weekly brainstorm sessions, listen to your feedback and show you my progress throughout the process. Thank you all for your time and expertise.

Abstract

Nowadays, Dopple's wireless earphones have touch buttons as a user interface, however these take in quite some physical space. Since the earphones are becoming smaller over the years, there is a need for a new type of touchless user interface, that is smaller. In order to create that new user interface, research is done on the topic of remote gesture sensing with sensors that can fit onto wireless earphones. When gestures are recognized by the system, a corresponding action can be taken, like for example pausing the music. Small infra-red imaging sensors are chosen as a solution to the problem. Its images are analysed by a trained image recognition neural network created with Python and Keras. This network takes an image as input and outputs a gesture. Each gesture is supposed be linked to an action in the new user interface. This report focusses on the retrieval of low resolution infra-red images and neural network training/machine learning. The APDS-9500, an already existing moving gesture sensor, is used as a comparison with regard to the new neural network technique. It achieves an accuracy of 92.3% with 5 different gestures. The AMG Grid Eye is an 8 by 8 pixel infra-red camera for which 5 gestures are trained. For raw images, 5 gestures are recognized with an accuracy of 79.2%. With the help of pre-processing in the form of contrast increasing and linear extrapolation the accuracy is increased up to 92.4%. The FLUKE 279 FC is a high resolution camera mounted on a multimeter of which the images are downscaled to 30 by 30 pixels. It is found out that while the size of the model increases, accuracy also increases up to 97.2% for 5 gestures. When the FLUKE is tested with 9 different gestures, while also optimizing for size, an accuracy is achieved of 98.2% with a model of 203 kB. The study proves that 30 by 30 infra-red images contain enough information to use gesture recognition with a small neural network.

Contents

List of Figures and Tables

Chapter 1

Introduction

1.1 Problem Definition

Innovation in the music industry for portable audio devices, like earphones and headphones, is required to keep up with the growth in customer demand. Think of the relatively recent development of wireless earphones, a trend that has gained significant popularity in the past 5 years. The market value of the headphones and earphones industry has grown in those same years and predictions are that this growth will keep going for the years to come [14]. Companies have to make an effort to distinguish themselves from the increasing amount of competition that exists in this industry.

Dopple is such a company, focussing on innovation and trying to find new techniques to implement in electrical wearable devices. One of the things Dopple is interested in, is finding a new way for users to interact with wireless earphones. Modern over-the-ear headphones have more than enough space on the device for a multitude of buttons and/or sliding switches. Earphones however, are very compact and cannot rely on such a user interface due to a lack of physical space on the device. If a user wanted to stop or forward music on their device, he/she is required to distinguish and press very small buttons on the earphone or open the device media player. A possible solution to this is to mount a small sensor onto the earphone capable of sensing gestures. This way, a gesture made by a user can be detected and then sent to the audio playing device as an action that has to be performed.

Gestures are an important part of human communication. A gesture is defined as a movement or posture of the human body that expresses an idea or meaning. These movements are mostly made with the hand or the head, like waving or making a facial expression. In the case of this application the focus lies on hand gestures. During this project it has to be decided which hand gestures are used and best suited for this UI. The gestures will have no prior meaning other than to the UI, but an attempt can be made to make the implemented gestures as intuitive as possible. These gestures are also dependent on the chosen method, will there be moving or non-moving gesture sensing. Moving gesture sensing implies the measurement of moving hands (e.g. left to right), while non-moving gesture sensing measures the shape of a non-moving hand. A consideration should be made for using moving, non-moving or both gesture sensing methods, while also keeping into account the limitations of the small sensor and earphone. Wireless earphones have a limited power supply and also a limited amount of computational power. Extensive algorithms should be avoided as well as power hungry sensors.

1.2 Knowledge Gap

The first gap to fix is the knowledge of microcontrollers and gesture sensors, since the use of these devices is at the core of this project. Based on the choices made for sensors and microcontrollers, a technique has to be thought of and created as a possible solution to the problem that is portrayed in section 1.1. The gap also includes the knowledge that has to be gained on the sensor circuits, power consumption and connections required to make the devices functional. The next gap to bridge is the knowledge of available methods for gesture recognition. The state of the art and the possibilities that are not explored yet have to be studied. Based on the closing of the two before mentioned gaps, a choice can be made for method and hardware. A specific programming language to write the software is not mandated by Dopple, though the company mostly uses C++ and microcontrollers also rely on this language. Nevertheless, the language can be one that I feel comfortable with, which means that it will either be Matlab, C++ or Python. Whichever is chosen, a recap of the language and its syntax is necessary. This would be more costly for C++ and less for Python and Matlab of which I would describe my experience as intermediate to good. Depending on the choices made for hardware and method, more challenges will become visible, like the methods and libraries required to reach the goal of this thesis.

1.3 Goal of the Thesis

The thesis is meant as a research project, the outcome will be a conclusion and analysis of the method and hardware chosen in this study. Where the method is a UI, capable of taking multiple instructions, like for example pausing audio files and lowering volume. The hardware is a sensor capable of recognizing the gestures for the UI and a microcontroller for processing and forwarding the commands that are read from the sensor. The minimum number of detectable hand gestures will be 5, based on the most used controls in audio playing devices.

- Pause / Play
- Volume Up
- Volume Down
- Play Next
- Play Previous

An analysis of the accuracy of the gesture recognition will be presented together with a recommendation for future development/research.

1.4 Research Questions

The research questions are derived from the problem definition and the preferences from Dopple:

- What are the current techniques for gesture recognition and how accurate are these techniques?
- What sensors are available for recognition and can be implemented while holding for the criteria for power consumption, processing power, size and price?

- What method can be chosen to meet those criteria and how will it be implemented?
- How accurate is the chosen method and how does it compare to current techniques?

1.5 Outline of this Report

The report starts with an analysis of different methods for detecting gestures followed by a short overview of literature on the state of the art gesture recognition sensors in chapter 2. Next, a selection of different sensors is listed in chapter 3. In chapter 4, a method will be proposed, based on machine learning and image recognition. In the chapter 5, it is explained how the hardware is set up. This chapter will also explain the software used to make the sensor functional. In chapter 6, it is explained what the theory behind machine learning and image recognition is. In chapter 7, the Keras API and the structure of the machine learning algorithm will be explained. In chapter 8, it will be explained how the data is gathered. In chapter 9 the results will be shown. And in chapter 10 and chapter 11 a discussion and a conclusion will be given, together with a suggestion for future research.

Chapter 2

Introduction in Gestures and Gesture Sensors

Gesture detection is still in its early developing stages, there are companies focussing on the subject and some, like Broadcom [15], have been successful. However, it is not yet applied in every day technology, current UI's mainly use touch screens, mouse or keyboard. In some cases this is the easiest possible way, in others, like wireless earphones, gesture detection is a potential alternative. There are different sensors that can detect gestures. There is ultrasonic, electromagnetic an IR sensing. Ultrasonic waves can cause dizziness and other negative side effects. And electromagnetic sensors can heat up tissue in close proximity. This is why we focus on IR sensors as suggested in a thesis by Søderholm (2021) [16]. In this chapter we can look at what methods/sensors already exist on today's market, capable of detecting gestures.

2.1 Infra-red Sensors for Gesture Detection

An IR sensor consists of photodiodes that can generate different forms of data from IR light. From that data a gesture can be recognized with help of the right algorithm. However, in order to implement the gesture recognition into earphones, the sensing element has to be relatively small. This leaves a small amount of photodiodes to feed the algorithm. This is a trade-off problem between the info that the sensor can pass to the algorithm (which is related to number of different gestures it can measure) and the size of the sensor. The photodiodes are usually sensitive to a specific IR wavelength, this does not mean that ambient light (red, green and blue) can not be implemented. But because ambient light sensing (ALS) has a clear disadvantage in non/low light environments, it is not practical.

IR sensors can have their own light emitting diode(s) (LED) on the device, emitting the wavelength of light that the photodiode/sensing element is sensitive to. This way, when an object enters the sensing area, light will be reflected from that object back into the sensor, recognizing that there is an object nearby. This form of sensor is called an active IR sensor. IR light has a very large spectrum ranging from 700 nm to 1 mm. The spectrum of IR light is divided into three ranges of wavelengths:

- Long-Wave IR (7.5-14µm)
- Medium-Wave IR (3-5µm)
- Short-Wave IR (1-3µm)

Active IR sensors are sensitive to the shorter wavelengths of the IR spectrum. Longer wavelengths are used for relatively expensive (for example military) equipment, because longer wavelengths can travel further and can also move through obstacles more easily.

There are studies that use passive IR sensing (Wojtczuk et al. 2013 [17]), this requires a middle to far IR light sensor. Heat radiating objects are a primary source of far IR light. For this application, and in this market, the near-IR is (currently) the cheaper and more practical alternative. The short wavelength LED's are cheap and sufficient for short distance measurements. It is also less noisy because the light is not emitted/reflected from humans in large enough quantities to affect the cameras that measure it. This is also the more studied form of hand gesture detection, since most studies on gesture detection use a wavelength of around 900 nm. An advantage to passive IR sensors however, is its low power consumption due to the absence of LED's.

There are already sensors on the market that use gesture recognition, these are active gesture sensors capable of sensing movements over the sensing element (left, upwards etc.). The simplest form of a gesture sensor

Active IR Sensor Advantages	**Passive IR Sensor Advantages**
No noise from heat sources	Low Power Consumption
Cheaper	No LED Timing/Programming
Smaller	Longer Distance Measurements

Table 2.1: Advantages of active and passive IR sensors.

contains one sensing element and one LED. As suggested earlier with the trade-off effect, this sensor would be small, but is only able to distinguish an object/gesture approaching the sensor from one moving away from the sensor. More gestures can be distinguished with more sensing elements and/or more LED's. Because the LED is the most energy consuming part of these devices, the first option is preferred. In some examples/studies from the following sections examples can be given with multiple LED's, this can be favourable when looking at the costs. However, in all cases an extra LED can be replaced by an extra photodiodes/sensor, as long as it is clear to the software which LED lights up at what time.

2.1.1 Distance and Position-Based Sensors

There are multiple methods for detecting gestures with IR sensors, this method is based on calculating the distance from one or more sensors to the object. There are two ways of doing this, first is measuring the amount of light reflected from the object into the sensor. A certain relation is defined between light intensity and distance from the sensor. However, the light reflected back from objects is also based on the reflection coefficient of that object and its size. Luckily, there is not that much variation in the reflection coefficient in human skin. As for the other disadvantage there is a problem: when a large hand is used at the same distance compared to a smaller hand, more light will be reflected back to the sensor regardless of distance. This is where the second technique comes in: time-of-flight sensing. Time-of-flight does not rely on a relationship between intensity and distance, but on time and distance. The sensor measures the time it takes for the light to travel from the LED to the hand an then back to the sensor. Making it independent of reflection properties of the human skin and object size. Distance as described here is one scalar value. When using multiple LED's or sensors, more distances can be measured from different points on the sensor. When knowing the difference in location between these sensing elements, the

2D or 3D distance of the object can be measured as seen in figure 2.1. Lastly, an algorithm has to be applied to find a pattern/timing in the distance data to see if any gesture is made. A pattern can be, for example, a left entry of the hand followed by a right exit, which would indicate a swipe from left to right. It could also be a static gesture, for example keeping the hand at a given distance for 2 seconds.

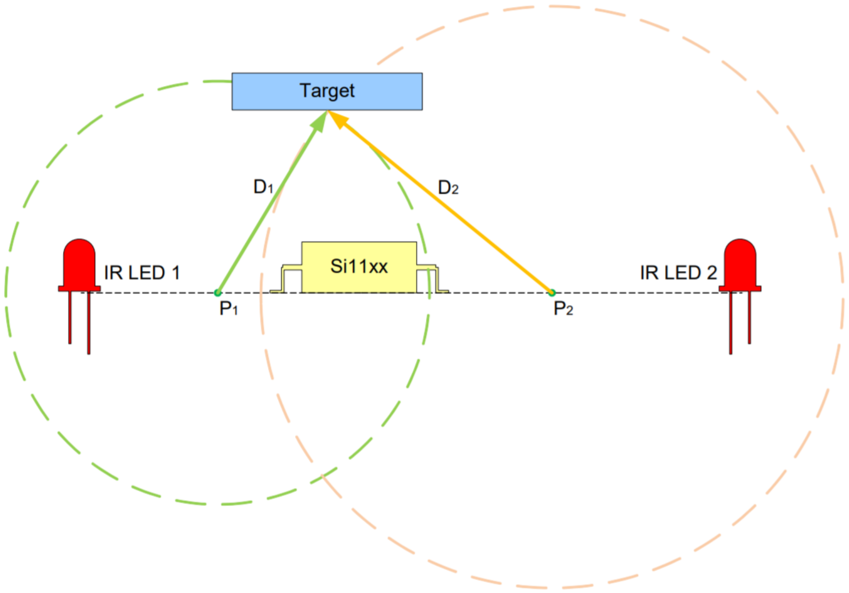


Figure 2.1: Estimation of distance from the sensor based on two LED's and one sensing element.

There are a few things that have to be noted here. First is that the LED's should emit alternately and quickly. If emitting is done simultaneously or it is not clear which LED lights up at what time, it would make the measuring 2 distances, as shown in figure 2.1, impossible. This could also be solved by using a layout with 2 sensing elements and one LED. Secondly, time-of-flight sensors are usually equiped with Vertical Cavity Surface Emitting Laser (VCSEL), a more efficient and less power hungry light source compared to the LED. The third thing to note is that this method is visualized and makes sense when using a very small object. An open hand however, is a wider object and would take effect on the approach, since the hand would appear closer in the situation where the intersection point of two circles is calculated. Because in fact, it is measuring the distance from two different points on the hand, namely the left and right side of the flat hand. This will become a significant problem when objects are close to the sensor.

2.1.2 Phase-based Sensors

The second method for gesture recognition is the phase-based recognition. This approach does not calculate distance or position with regard to the sensor, it looks at the order in which LED's or sensors activate. In figure 2.2 it can be seen that D2 will be inducing a signal first, then D3 and lastly D1. This pattern will indicate a swipe to the left. In this case the LED's will also be activated alternately in order to know from which LED the signal is picked up in the sensing element. The phase-based method will only be able

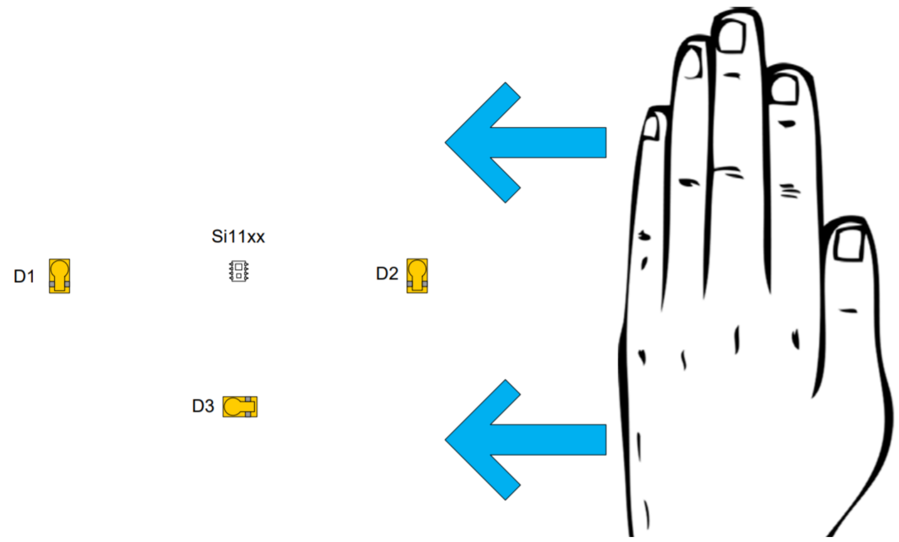


Figure 2.2: Visualization of a leftward swipe detected with the phase based method.

to detect a gesture that enters the sensing area and exits it somewhere else, it does not keep into account any movement in the middle of the sensor. This limits the method to only measure the direction by using the entry and exit of the hand. This does imply that combinations of multiple gestures can be recognized, for example a swipe right followed by a swipe left. In order to do more complicated pattern recognitions than swiping left, right, up or down, it is inevitable to use machine learning. Pattern recognition is hard to implement without the help of computer calculations.

2.1.3 Imaging Sensors

The third and last method for gesture recognition is performed with an image sensor. It uses sensors that have the ability to create an IR image through the use of multiple photodiodes. If it is an active IR sensor, it is supported with extra IR LED lighting. The light hits an array of photodiodes in the sensing element, creating a current that is related to an array of data/image. This current will eventually become pixel data after it is amplified and translated. The resolution is dependent on the number of photodiodes in the sensor. The pixel depth is dependent on the number of bits that the device can handle. The bit is translated into an intensity value in the image, thus forming one of the pixels. Most of the small sensors used create low resolution images (less than 5000 pixels). This is because most of the sensors are so small that the physical space is too limited. The ADPS-9500 for example, is a sensor (dimensions 2.83 x 6.83 x 3.72 mm) capable of capturing an image of 60 x 60 pixels. These kind of images can be used in an image recognition algorithm, capable of recognizing the different gestures. Though the use of these images broadens the scope of recognizable gestures, it also puts quite a demand on data transfer and workload for the controller. The challenge when using image-based sensing is to keep these disadvantages to a minimum.

2.2 Related Work

Studies that focus on gesture recognition using single sensor setups, with only a few sensing elements, mostly rely on specific photodiode structures. Zivkovich (2014) [2] used a sensor consisting of five photodiodes and an IR LED. The structure used here is diamond-shaped and it is supported by the walls on the side of the chip. Differences can be measured between each of the photodiodes, when a movement is made over the sensor, these differences can be highlighted or strengthened by sensor package's slanting walls (see figure 2.3). Different movements over the this area will induce patterns in the sensor measurements (phase-based). These patterns are taught to the controller via machine learning. It is found that most of the gestures are

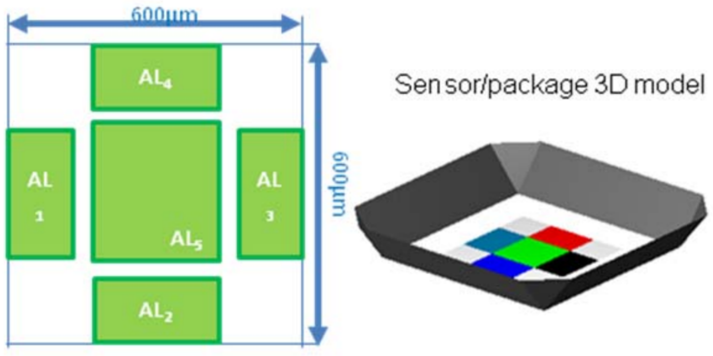


Figure 2.3: Model showing the layout of the photodiodes and the walls on the side of the layout.(Zivkovich 2014) [2]

taught reasonably good, an accuracy of 93% is measured as seen in the confusion matrix in figure 2.4. On top of that, the addition of an optical lens improved accuracy up to approximately 99%. Machine learning is done using an ADA-Boost classifier and a total of 600 measurements per gesture (300 testing, 300 training).

No optics (full gesture set)					
	Push	Left	Right	Top	Bottom
Push	290	1	1	5	3
Left	2	288	1	4	5
Right	5	2	280	6	7
Top	10	5	7	275	3
Bottom	9	11	13	5	262

Figure 2.4: Confusion matrix from the experiments done by Zivkovich (2014). [2]

Binnie, Armitage and Wojtczuk (2017) [18] used the vertical, horizontal and away from/towards motion above an IR sensor. Four sensing elements are placed on the chip (2mm x 2mm) in a symmetrical fashion. When executing a hand swipe above the sensor, a signal is induced in all four sensing elements. The order in which peaks appear in the signal of all four individual sensors indicates the type of swipe that is used (phase-based). For this algorithm little over 2000 hand gestures are used to test the algorithm, resulting in 76% accurate, 20% undecisive and 4% erroneous results. However, some test subjects are able to achieve results above 95% accuracy, which indicates that the results are very subject dependent and can perhaps improve by training.

Another method is shown by Das, Jakulin and Nigel (2016) [3], where a least squares regression is used to determine whether the gestures are horizontal or vertical. In addition to that, IR light sensors are used to recognize rotational changes in objects or gestures. A triangular layout is used for the sensors, as seen in figure 2.5. From the experiments it is

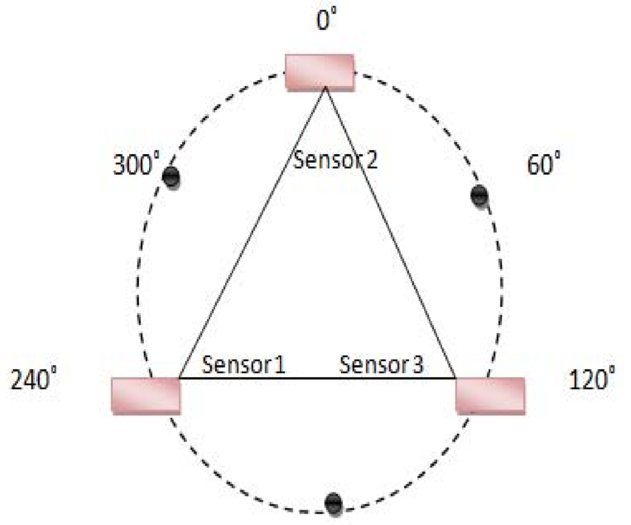


Figure 2.5: Sensor layout used for rotational gesture determination in steps of 60 degrees (Das et al. 2016) [3]

seen that rotational movements can be recognized with the measurements of the three sensors. Specifically the order in which signals are measured is important in this setup (phase-based). The study does provide proof of concept and thus provides perspective for rotational gestures. However, the size of the sensor is not given in the study which is important, since the sensors have to be placed on small earphones.

Cheng, Chen, Razdan and Buller (2011) [4] used a decision tree structure to determine the type of gesture made. In this tree parameters like variance, channel delay and slopes are considered (see figure 2.6). The decision tree can be seen as a collection of choices cancelling out or selecting specific types

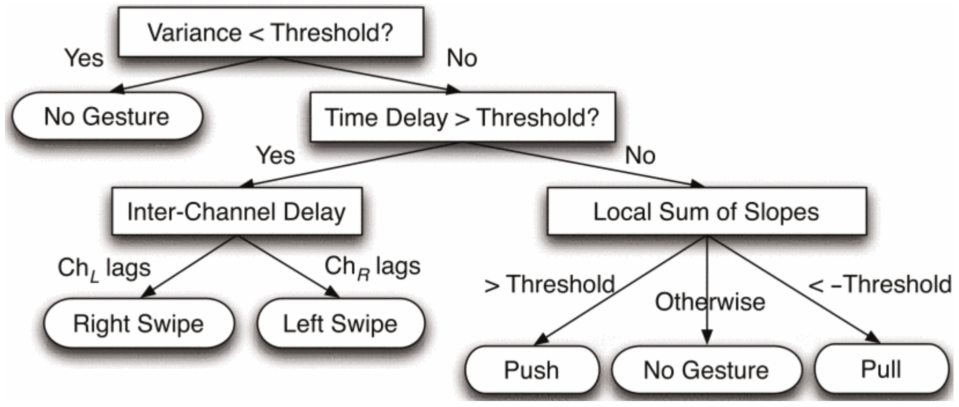


Figure 2.6: Decision tree from Cheng (2011) [4]

of gestures one at a time. The inter channel time delay for example, makes clear whether the gesture is a right or left swipe. Because left and right swipes indicate a motion from one side of the measured plane to the other. The local sum of slopes indicates how fast a hand is moving towards or away from a sensor. This slope can be determined with linear regression, where a positive and negative slope indicate either moving away or towards the sensor. A database containing 2000 hand gesture signals is used to determine the accuracy of the algorithm, which is 98%. Accuracy is measured over swiping right and left, and moving away from and towards the sensor. The sensor used is a Silicon Labs Si1120 IR proximity sensor, which consists of one sensing element and one IR LED. This could be why swiping up and down are excluded from testing, since these are very similar to swiping right and left and are thus hard to distinguish.

Yu et al. (2021) [19] used an IR sensor setup with a raspberry pi to recognize 8 different gestures. To denoise the signal, the discrete wavelet transform (DWT) is used. Wavelet thresholding uses the wavelet coefficients to decide whether a part of the signal is noise or not. When the coefficient is beneath the threshold, it is set to zero (for specifics on this Lord et al 2012 [20]). A thresholding scheme is also used to detect the start and end for each gesture. The threshold here is chosen to be 10%, which made sure that the gesture signal is not affected. The 10 percent threshold is determined from both the situation at time of measurement and the amplitudes over the entire signal (thus a segment is taken after gesture is executed). After that, standardization of the segment is done with the Z-scores method. This will not alter the shape of the signal, but it will make the signal zero mean and the standard deviation one, so it can be compared to similar signals for classification. A KNN machine learning method is chosen for making the classification standards of each gesture. The measurements are done between 20 and 35 cm from the sensor. The setup of these sensors however, is to large to be applied in a small earphone. Next to the differrence in size, the Raspberry Pi is a microprocessor that has more processing power than the average microcontroller. The study did prove that there are numerous different gestures possible, other than the mostly used up, down, left, right, away from and towards. The accuracy of the method using IR sensing is above 90% for most participants and for all gestures, also in conditions where there is no ambient light.

Chronopoulos (2019) [21] used a diamond-shaped time of flight sensor layout of four VL6180X from STMicroelectronics. The sensors are equiped with an VCSEL laser capable of emitting 850 nm pulses. It also contains an ambient light sensor sensitive to a wavelength of 450 to 700 nm. A FIFO approach (First-In-First-Out, regarding order of signals in different diodes) is used for determining gesture direction and velocity, which can also be determined due to the sensor layout. The use of a quadrant or diamond structure is common for these gesture recognizing applications (see for example Binnie, Armitage and Wojtczuk (2017) [18]) however the use of a single VL6180X sensor is more practical. A PCB (printed circuit board) as produced by Chronopoulos (2019) [21] is too complex, the different signals would have to be merged or sent to a controller over many wires. A single sensor setup is simpler and more practical.

Al, Estrela and Martinez-Hernandez (2020) [22] used the VL6180X and APDS-9960 to determine horizontal and vertical gestures (4 total) for control of a robotic arm. The approach used is an ANN (Artificial Neural Network) machine learning technique. Data is gathered from 4 IR sensors on the APDS-9960 for measurement of relative movement. The VLX6180 is used for proximity detection. The sensors are placed on a 3D-printed bracelet and used for real time testing. For the 4 gestures an accuracy is found of 99.7%. The downside of this is that two sensors are used and just as in Chronopoulos (2019) [21], complexity is high, because two signals have to be analyzed in one microcontroller, while coming from different sensors. It is also possible to use only the APDS-9960, of which implementations can already be seen in prototypes of different projects, like a wheelchair made by Saini, Bamane, Bhanpurwala and Hirlekar (2020) [23] that can be controlled with hand gestures.

Tateno, Zhu and Meng (2019) [24] used a sensor containing 32 x 24 thermophile elements. For static gestures, a technique called Gaussian mixture modelling (GMM) is used. This technique assumes that each gesture has its own Gaussian distribution in the pixel space, through these distributions gestures can be recognized after a series of pre-processing. The gesture is measured with a passive sensor, meaning a longer wavelength of IR is measured. Accuracy of recognizing a set of 7 gestures (both moving and non-moving) is 87,5%.

A study by Kim et al (2015) [5] wanted to measure gesture in 3D space with sensors that each had an adapted field of view. Some studies, like that of Zivkovich (2014) [2], did use side walls to make each sensor unique in its signal when sliding the hand over the device. Kim et al. (2015) [5] accomplished the same by using blockers. This adapted the field of view of each sensor in such a way that they all are nearly unique (see figure 2.7). This technique could also optimize the way in which machine learning would tackle the problem of gesture recognition, since more unique sensors would provide more unique patterns and thus more distinguishable/recognizable gestures. The sensor created by Kim et al. (2015) [5] is capable of detecting directional swipes over the device, with an accuracy of over 99,5%.

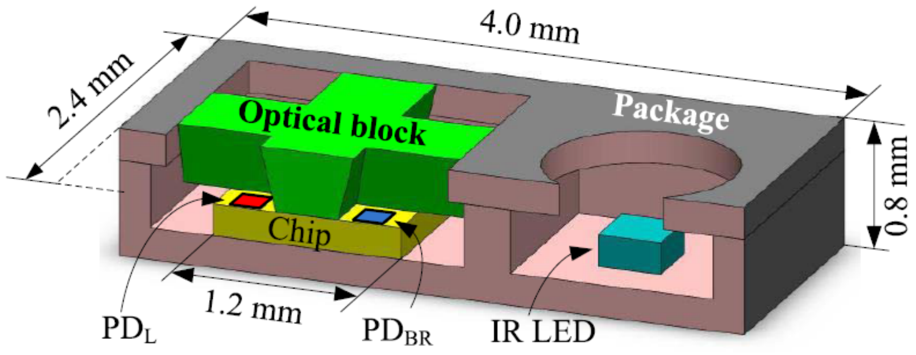


Figure 2.7: Optical block used by Kim et al (2015) [5]

2.3 Summary

This chapter highlighted a number of studies already performed with gesture sensors and their respective accuracies and/or conclusions. The topic of gesture recognition is popular and a rise in the number of studies has taken place since 2010. Most studies use 4 to 6 gestures and reports of accuracies of more than 95% are common. Most studies work with moving gestures and active sensors, which requires frequent LED lighting and thus more power. Multi-sensor setups are also used frequently and require more power. Nowadays, with the availability of image sensors and/or multiple sensing elements on one sensor, this seems needlessly complex. One sensor should be sufficient for the task, preventing extra wiring and signal merging in the controller. The ADPS-9960, for example, is already set up to handle 6 different gestures.

At the moment, the phase-based approach seems to be the most popular technique. Next to that, machine learning is the most common tool used to create models for gesture recognition. The phase-based approach is used for moving gestures, that requires a lot of LED flashing and this increases power consumption. A promising and currently not investigated option, is the use of imaging sensors. Non-moving gestures would only need one flash of the LED (or one image of a passive sensor) in exchange for higher processing demands. This would leave a challenge open to find a trigger for creating an image, like a small physical button, or a detection by the same sensor in a low power state.

Chapter 3

Hardware Description and Selection

3.1 Examples of Sensors

There are many sensors capable of the techniques mentioned in section 2.1. For the application in earphones however, the search will be limited to small sensors that can be fitted on the device and have a limited amount of power consumption. The latter is to make sure that the battery life of the cordless earphones is not affected too much. In the studies found, a small amount of authors mention the specific sensor that is used. And if it is mentioned, the chance is that the sensors are not applicable in earphones, due to size or processing needs. So a different search is conducted in online electronics stores, for small sensors with length and width in the order of millimetres. The number of applicable sensors shrinks even more when considering that these sensors should also satisfy a low power consumption criteria. A number of sensors is found and to get an idea of the type of sensors that fits the application, these sensors are mentioned in the following sections. The links to their specification sheets can be found in the literature list.

3.1.1 VL6180V1NR/1

The first sensor is the VL6180V1NR/1 [6], it is made by STMicroelectronics and is average when it comes to size (4.8 x 2.8 x 1.0 mm). The device is a typical time-of-flight sensor that has an extra option for ALS. The two functions cannot be used at the same time, measurements will take place alternately. It has a good performance when it comes to sleep and awake current. Since the sensor uses the time-of-flight measurement, it is very accurate and does not depend on a relative amounts of light coming back to the sensor. It is equipped with a vertical-cavity surface-emitting laser (VCSEL) for emission of light with a wavelength of 850 nm. The sensor only has one sensing element and one light source, so there is no possibility for position measurements, only distance from the sensor is indicated. For position either two sensors, or two LED's would be needed, both increasing power and size of the device. Without this the sensor would only be able to detect gestures: "towards" and "away from", which is too minimalistic for the UI.

Figure 3.1: VL6180 sensor made by STMicroelectronics.

3.1.2 TMD2635

The TMD2635 [7] is a sensor produced by AMS. The advantage to this sensor is that it is one of the smaller and cheaper sensors available (1.0 x 2.0 x 0.5 mm). It has a dual sensor layout, sensors are placed 0,547 mm apart on one half of the device. On the other half a VCSEL is placed, capable of emitting light with a wavelength of 940 nm. The advantages to this sensor are that it is simple, cheap and has crosstalk and ambient light cancellation to reduce noise. However, theoretically it would not be possible to measure more than four gestures with this sensor, and it is still questionable, if those gestures would be sufficiently distinguishable.

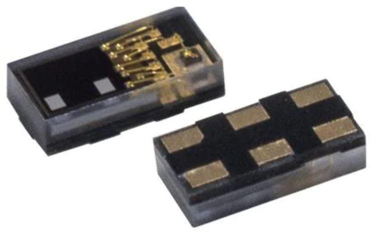


Figure 3.2: TMD2635 by AMS.

3.1.3 VCNL36826S

This is a sensor made by Vishay Electronics [8]. This device is small (2.55 x 2.05 x 1.0 mm) and simple. It uses a VCSEL that requires lower current compared to the LED's used by other sensors. The VCNL36826S is a single sensor device, but due to its size it might be possible (but unfavourable due to complexity) to use multiple at the same time, in order to increase the information gained for gesture recognition. The device also has useful features like sunlight cancellation.

Figure 3.3: VCNL36826S by Vishnay Electronics.

3.1.4 Si1120

The Si1120 [9] is an averagely large sensor (3 x 3 mm) that excels in its sensing range. Where most sensors have a sensing range of 0 to 100 mm or 0 to 200 mm, the Si1120 has a range of 0 to 500 mm. The awake current of the device is relatively low, but it is clear that the VCSEL consumes most of the power, which is very likely when supporting ranges up to 500 mm. The light source can therefore be supported by an independent power source. An advantage is that the device is also equipped with an ambient light sensor. A disadvantage is that the spectral response of this sensor is relatively wide, the IR sensor is also sensitive to signals from the ambient light spectrum as shown in the graphs of the data sheet.

Figure 3.4: Si1120 by Silicon Labs.

3.1.5 GP2AP054A00F

This sensor is made by Sharp Electronics [10]. The device is average in size (4.0 × 2.1 × 1.25 mm) and has low current consumption, especially when the LED is in its lowest power consuming state. For the sensor itself, current consumption is on the high side, but as with most active IR sensors, the LED is responsible for a majority of that large amount. The device has a dual IR sensor layout and an additional ambient light sensor. The sensor is by far the lowest in price compared to the other sensors, but still comes with good functionality, like a programmable LED current. Gesture recognition software is also available for this sensor, giving directional information of a swiping gesture.

Figure 3.5: GP2AP054A00F by Sharp Electronics.

3.1.6 APDS-9960

The APDS-9960 [11] is a sensor made by Broadcom Limited. The size of the device is average (3.94 x 2.36 x 1.35 mm) and the cost is average to cheap. The advantage of this sensor is that it has four photodiodes placed in a '+' formation. This makes the detection of swipes in horizontal and vertical directions very precise. Software implementation for detection of these movements is already available for the APDS-9960. The detection is done with phase-based recognition, like described in section 2.1.2 (but now with 4 photodiodes). The sensor can also measure ambient light intensity, by using the same '+' formation, but with IR blockers (like in Kim et al 2015 [5]). There is a programmable LED on the device, capable of emitting light with a wavelength of 850 nm. Since this device has four 'directional' photodiodes, the most basic gestures can be accurately detected. A challenge for this sensor would be to extend the possibilities with more gestures, however, just like sensors with two photodiodes, it is not sure if this is possible with the phase-and position-based methods.

3.1.7 APDS-9500

The APDS-9500 [12] is one of the sensors found capable of supporting the imaging technique from section 2.1.3. Even though the sensor gives a lot more information then most others, the size does not increase dramatically

Figure 3.6: APDS-9960 by Broadcom Limited.

(2.83 x 6.83 x 3.72 mm) and awake current is average. Sleep current however, can increase above average up to 10 µA. For this device the output is a 60 x 60 pixel array, with every pixel given on a 9-bit grayscale. There are settings for cropping or averaging pixels to decrease the resolution of the image. The APDS-9500 is also available with gesture recognition software. Next to the basic swiping gestures it can also detect motion away from and towards the sensor, clockwise and counter clockwise movements and it can detect a waving motion. The question when using this sensor is, whether the gesture mode will be used or the imaging mode. The downside to this sensor is that it is large size and high in sleep current. When using the sensor in gesture mode, the LED is constantly lighting up, which will affect the power consumption to a large extent. The downside to imaging mode is that it has to process a relative large amount of information, however the LED only has to light up once for that information to be given.

Figure 3.7: APDS-9500 by Broadcom Limited.

3.1.8 Grid Eye

The Grid Eye [13] is a sensor made by AMG electronics. The Grid Eye is a passive IR sensor capable of capturing an 8 by 8 pixel image. Next to lowering the power consumption due to the absence of an LED, the Grid Eye's frame rate also does not affect the power supply. Passive IR sensors are much more expensive than active IR sensors, so if the sensor is used for research it would be for future prospects, since the increase in price of the earphones would be too large. Next to a future price drop, it would also be good if the size of the sensor decreases (11.6 x 8 x 4.3 mm). It can be tested whether the image mode of an 8 x 8 sensor is sufficient enough to apply gesture recognition with.

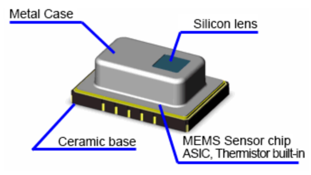


Figure 3.8: AMG Grid Eye by Panasonic.

3.2 Sensor Communication Protocols

There are different communication protocols for electrical devices, there is USB, UART, I2C, SPI and more. Devices need these protocols in order to know how to communicate from one device to another. For example, what part of the device is the message for? Or what does the one part want from the other, reading or writing? These issues are all described and tackled in the communication protocol. All devices discussed in section 3.1 have the option for I2C, but there are also ones available that can use SPI. Both these communication protocols will be discussed in this section.

3.2.1 I2C

I2C or Inter Integrated Circuit was originally made by Philips for one of their TV systems. It is the most common of the communication protocols when it comes to sensors. For I2C there is a serial clock line (SCL) and a serial data line (SDA). There is a possibility for one or more master and/or slaves, whenever a master wants to read from or write to one of the slaves, it will send the 7-bit address of that particular slave over the SDA (which will be sampled as indicated by the SCL). Since this is a BUS protocol, all of the slaves and masters can listen in on the request, but only the slave who's address is mentioned will send an acknowledgement back in order to let the master know that it received the call to action. The number of slaves is limited by the number of unique addresses due to the fact that all slaves are listening in on the requests. If an acknowledgement has taken place the transfer of data can start. Whenever the SCL is pulled high again, the SDA will be sampled, until a stop command is given. On the SCL there are a few commands possible:

- Start
- Stop
- Read
- Write
- Acknowledge
- Not Acknowledge

The data over the SDA will contain the bits with information, this could be a sensor output, user command etc. A visualization of the communication protocol can be seen in figure 3.9.

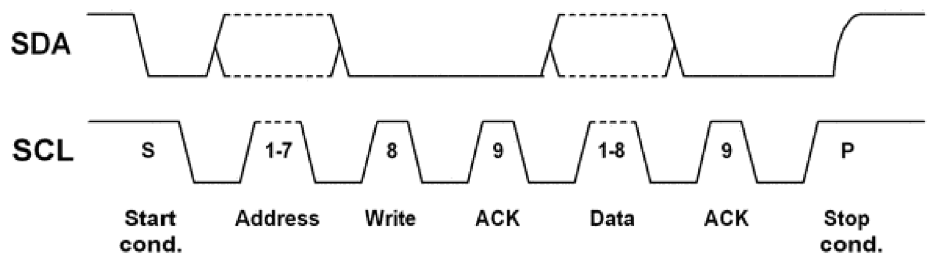


Figure 3.9: I2C based communication and addressing.

The advantages of I2C communication are:

- Only uses two pins/lines and is thus not a complicated circuit.
- Built-in addressing.
- Also supports ACK (Acknowledge) and NACK (Not Acknowledge) for better error handling.
- Can Handle multiple masters.

There are also disadvantages to the I2C interface:

- Limited data transfer (usually 400 kbps is max).
- Data collision risk needs to be excluded.
- Slave addresses should be unique.

3.2.2 SPI

SPI (serial peripheral interface) is a very common communication protocol created by Motorola in the 1980's. It is used in for example SD cards, TV's, microcontrollers and more. The interface is more complicated than that for I2C even though it only has the possibility of one master. The complexity is higher due to the pins it uses, the 4 pins are as follows:

- SCLK : Serial Clock, controlled by the master, telling the slave when to sample the data.
- MOSI : Master-Out Slave-In, data going from master to slave.
- MISO : Master-In Slave-Out, data going from slave to master.
- SS : Slave Select, tells the slave when to listen in on the MOSI, there is one SS for each slave in the circuit.

An example of the pins connected to multiple slaves is shown in figure 3.10. The major difference between I2C and SPI, is that SPI can use its data lines

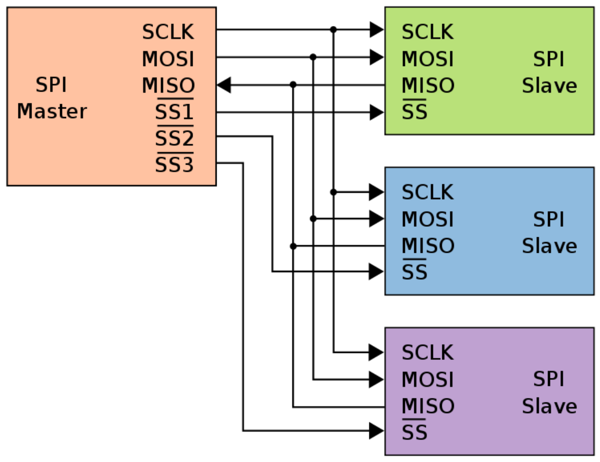


Figure 3.10: SPI BUS containing 1 master and 3 slaves.

to simultaneously send data from and to the slave. Due to this, SPI is also a faster protocol. Another difference is that SPI has 4 different operational modes, aptly named mode zero, one, two and three. The first two modes have the so called clock polarity of 0, meaning that the clock value is 0 when idle. The last two modes have a clock polarity of 1, meaning that the clock is 1 when idle. The other setting, making mode 0 different from mode 1, and mode 2 different from mode 3, is called clock phase and indicates where the data is sampled on the clock. This is either at the rising or the falling edge of the clock, indicated by a clock phase of 0 and 1 respectively. When the master wants to address one of the slaves it pulls the SS line low, telling the slave it should now start paying attention to its MOSI. The signal will be sampled at the clock rate until the line is brought up again. Advantages of the SPI communication protocol are:

- Higher speed than I2C.
- Uses less Power than I2C.
- Full-duplex (can send and receive simultaneously.
- Ubiquitous (used in many applications).

Disadvantages of SPI are:

- Only supports one master.
- More Pins are used.
- No acknowledge to verify the signal sent.
- Made for short distances.
- Variations in modes make SPI more complicated.

3.3 Summary

The hardware summarized in section 3.1 differs in a various of ways. Most of the smaller sensors contain only one or two sensing elements (section 3.1.1 to 3.1.5) , making them very compact, but also very restricted in what gestures they can measure. The availability of two sensing elements for example can not possibly distinguish more than 4 gestures. And some studies in chapter 2 use four to six gestures, but the sensors with one or two sensing elements are insufficient for this amount. This means that a lot of sensors available on the market can be regarded as unfit for the application.

The APDS sensors can acquire a lot of information relative to their size and already have software for gesture recognition implemented with I2C, integrated in the device. In order to extract image information from the APDS-9500, communication over SPI is required. If available, I2C is the preferred communication protocol due to its simplicity compared to SPI. However, when high speed is required, SPI is unavoidable. When looking for an image sensor the APDS-9500 outperforms the AMG in resolution and size. However, the advantages of the Grid Eye are that it uses I2C and is very low in power consumption due to its passive IR measurement. And for processing demands, a 64 pixel image is more attractive compared to a 900 pixel image.

Sensor	**VL6180V1NR/1**	**TMD2635M**	**VCNL36826S**	**Si1120**
Manufacturer	STMicroelectronics	AMS	Vishay	Silicon Labs
Size	4.8 x 2.8 x 1.0 mm	1.0 x 2.0 x 0.5mm	2.55 x 2.05 x 1.0 mm	3 x 3 x - mm
Weight	21 mg	477 mg	N.R.	17.6 mg
Supply Voltage	2.6 to 3.0 V	1.8 V	2.62 to 3.6 V	3.3 V
Sleep Current	<1 μA	0.7 μA (& Idle = 30 μA)	Not Available	90 μA (Only idle)
Awake Current	IR = 1.7 mA (Typical mean)	340 μA (+VCSEL)	200 μA (+VCSEL)	10 μA (+VCSEL)
Nr. of IR Photodiodes	1	2	1	1
Nr. of IR Emitters	1 (VCSEL)	1 (VCSEL)	1 (VCSEL)	1 (VCSEL)
Used IR Wavelength	850 nm	940 nm	940 nm	850 nm
Extra Ambient light Sensor	Yes	No	No	Yes
I2C Communication	Yes	Yes	Yes	Yes
Price	€3.37	€1.02	€2.13	€2.68
Notes	· Performance not Dependent on Reflection Properties Object. · Cannot use ALS and IR at the same Time.	· Crosstalk and Ambient Light Cancellation. · Dual Diode. · Gesture Implementation Exists.	· Low VSCEL Current (6 to 20 mA). · Sunlight Cancellation.	· Broad Sensitivity in spectrum (see sheet [9]).

Table 3.1: First overview of sensor specifications of all sensors mentioned in section 3.1.

Sensor	**GP2AP054A00F**	**APDS-9960**	**APDS-9500**	**Grid Eye**
Manufacturer	Sharp Electronics	Broadcom	Broadcom	Panasonic
Size	$4.0 \times 2.1 \times 1.25$ mm	3.94 x 2.36 x 1.35 mm	2.55 x 2.05 x 1.0 mm	11.6 x 8 x 4.3 mm
Weight	N.R.	30 mg	N.R.	N.R.
Supply Voltage	2.2 to 5.5 V	2,4 to 3,6 V	2,8 to 3,6 V	3.3 V
Sleep Current	N.R.	Idle = 30 μA	1 to 10 μA	0.2 mA (No LED)
Awake Current	320 μA (+LED)	200 μA (+LED)	200 μA (+LED)	4.5 mA (No LED)
Nr. of IR Photodiodes	1	4	60 x 60	8 x 8
Nr. of IR Emitters	1	1	1	0
Used IR Wavelength	940 nm	850 nm	940 nm	Middle to Far IR spectrum
Extra Ambient light Sensor	Yes	No	No	No
I2C Communication	Yes	Yes	Yes (and SPI)	Yes
Price	€0.67	€1.75	€4.98	$30
Notes	· Programmable LED. · Gesture Implementation Exists.	· Programmable LED. · Gesture Implementation Exists. · Four Diodes. · Ambient Light Sensor uses same Diodes.	· Ambient Light Cancellation. · Programmable Pixel Output. · Gesture Implementation Exists.	· No LED (low power) · Range Limited by Resolution/ Image Size.

Table 3.2: Second overview of sensor specifications of all sensors mentioned in section 3.1.

Chapter 4

Proposed Method and Requirements

4.1 Choice of Method and Sensor

As seen in section 2.2 the accuracy of some existing algorithms is already high and over 95%. Improvements can be made towards power consumption and scalability of the sensor. In order to lower power consumption the most profit can be made by adjusting the use of IR LED's. This means lowering the brightness of the LED, reducing power-on time of the LED, or removing the LED entirely (passive IR). The image mode discussed in section 2.1.3 is not used frequently in the studies on gesture recognition. When using imaging, one large advantage is that only one blink of the LED is needed to gather information on a passive gesture. If for example one finger is pointing to the right, this should be recognizable on an IR image with the human eye. This should also be the case for a passive gesture in for example an image of the APDS-9500 or Grid Eye. One of the rising technologies, is that of machine learning. Machine learning is broadly applied from regression problems to image recognition. The last could prove useful in the recognizing of gestures in the IR images. If a human eye can distinguish these gestures from one another, then there is certainly a possibility that a machine learning algorithm can do the same. Take for example recognition algorithms for dogs, cats and boats. These are all things that are already implemented. The risk here is that the algorithm becomes too complex for a small microcontroller (mounted on an earphone) to handle. On the other hand, when working with a low resolution image like those of the Grid Eye and ADPS-9500, few pixels are there to work with, this reduces the computational demand compared to regular images used for image recognition/machine learning. The AMG sensor is a good alternative compared with the APDS-9500, when looking at the low power consumption. Even though it might be slightly too large, it would provide proof of concept if a successful algorithm is trained for image recognition. If an 8 by 8 pixel frame contains sufficient information for a machine learning algorithm to recognize different gestures, this will also be the case for the 30x30 or 60x60 image, like those acquired by the APDS-9500. As a result of the previous chapters the following steps will be taken:

- As a reference for currently available gesture sensors the APDS-9500 will be used. Its gesture mode can be tested, and accuracy determined with multiple subjects.
- Extract gesture images from the AMG Grid Eye and use the data for an image recognition algorithm. This algorithm can be trained and tested with raw or pre-processed data. The aim is to improve the accuracy and/or the number of gestures recognizable with the APDS-9500.
- If the images from the AMG Grid Eye are not outperforming the APDS-9500 in accuracy or number of gestures, an alternative, higher resolution IR camera (FLUKE 279 FC) can be used for training. In order to see whether a better resolution can outperform the APDS-9500.

4.2 Choice of Microcontroller

The sensors measurements have to be processed, this needs to be handled by a microcontroller. A microcontroller is an electronic device that takes care of predefined tasks. They can be found for example in washing machines, keyboards and microwaves. An input will be given to a microcontroller, in this case the sensor data. The pre-programmed microcontroller will be responsible for running the software for the sensor, generating a clock and/or data sampling rates for example. To know what microcontroller suits the use of this application most, a search is conducted on multiple online electronic hardware shops. There are many microcontrollers available on todays market. A few criteria are made to narrow down the possibilities:

- I2C connection (for both sensors).
- SPI connection (potentially for APDS-9500).
- Low Power Consumption.
- High Computational Power for possible image processing.

When searching for these specifications, the Artemis Redboard by Sparkfun stood out. It is equipped with an Apollo 3 microcontroller, which met all the requirements. With its high computational power, high clock rate (48 MHz) and low power demand it will able to effectively run a recognition algorithm on an image. The Redboard Artemis has an Arduino uno layout, which is very user-friendly. This made the Artemis Redboard an ideal candidate for this project.

4.3 Choice of Software

When working with machine learning, the most advanced language to use is Python. Python is user friendly, well documented and has the best libraries built around machine learning. Tensorflow is one of these libraries, which can be used in combination with Keras, an overlaying library for Tensorflow. Tensorflow also has dependencies like Numpy for linear algebra needed for the project. Python will be setup in a Linux environment (VMWare Virtual Machine), and its scripts are ran via the terminal. Sublime text editor is used for the coding. The machine learning process requires images from the sensor, gathered with the microcontroller. The software used for the microcontroller is C++. The Arduino application will be used to run, write and edit the code. In total 5 programs have to be written:

- A machine learning Algorithm: this takes a large database of images as an input and outputs a model. This output model/algorithm takes as an input one image and as an output the estimated gesture that is performed in that image.

- C++ script for the AMG Grid Eye: this script will retrieve the image pixels from the AMG and send them to the computer.
- A Matlab script: this script will wait for data entering the serial port on the computer. When available, the script will load the data into an image format and store it in the given folder on the PC.
- A Matlab script for pre-processing the images. If raw images do not achieve a high accuracy, it can be tested whether results improve with artificially modified images (e.g. pixel value manipulation and/or extrapolation).
- C++ script for handling the reference gesture recognition algorithm (apds-9500). This will gather data from the sensor, which will contain the gesture made. No Matlab script is needed here, because the result can be printed on a serial monitor, and does not have to be stored on the PC.

Chapter 5

Hardware Setup and Programming

The APDS-9500 and AMG Grid Eye sensor need to be setup with the Artemis Redboard for acquiring the data. For both sensors I2C connections are required. The language for the microcontroller is C++ and this is written in the Arduino IDE. The setup for the FLUKE 279 FC is performed with Smartview software made available by the manufacturer. No extra software had to be written for the FLUKE, the instructions for extracting images or gestures from all devices can be found in Appendix B, C, D and E.

5.1 AMG Grid-Eye Setup

In this chapter the steps taken to setup the AMG Grid Eye are given. To start capturing images, the following hardware is has to be gathered first:

- AMG Grid Eye
- Artemis Redboard
- Computer with USB-A connection
- USB-A to USB-C cable

- Jumper Cables
- Optional: BreadBoard + Logic analyser (only for debugging purposes)

As for the software, Matlab [25] and the Arduino application [26] are needed. Programs to run for this application are an AMG arduino script and matlab script. The AMG Arduino script is an adaption to one of the scripts by Sparkfun (manufacturer Artemis RedBoard). This script will let the micro-controller read out the I2C registers for all the pixels in the sensor. It will then send the values to the computer via the serial line. The important parts of the script are the header (see appendix G.1, containing the function that reads out the I2C registers (2 per pixel) and the Arduino program responsible for sending the data to the PC, as seen with comments in appendix G.2. The sensor sends data via the serial line towards the PC, a Matlab program is written to listen in on the signal and create images from the segments of data. This Matlab script can be found in appendix G.3.

The AMG Grid Eye is a sensor that uses I2C communication, the advantage is that few connections are needed to establish a signal. The sensor information and recommended circuit (see figure 5.1) can be found on the data sheet [13].

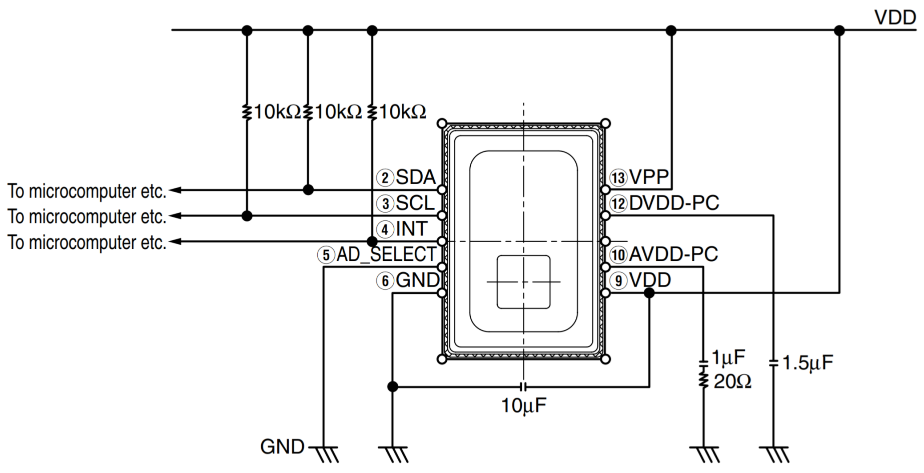


Figure 5.1: Connections for the pins in the AMG Grid Eye.

The printed circuit board is provided by Dopple, and is based on the data sheet from the sensor. In order to make the device operational, 5 pins are needed. These are the connections for each possible pin:

1:	VDD	$\rightarrow$	3.3V
2:	SDA	$\rightarrow$	SDA
3:	VVP	$\rightarrow$	N.C. (Not Connected)
4:	SCL	$\rightarrow$	SCL
5:	AVDD	$\rightarrow$	N.C.
6:	INT	$\rightarrow$	N.C.
7:	DVDD	$\rightarrow$	N.C.
8:	AD-SEL	$\rightarrow$	GND
9:	GND	$\rightarrow$	GND
10:	GND	$\rightarrow$	N.C.

When the connections are established, a small I2C scan can be performed, if correct it should state a device address like 0x73. Now the Arduino code can be compiled and uploaded to the Artemis RedBoard microcontroller. When this is done the data can be checked on the serial monitor, if data is visibly printed onto the monitor, the last step is to run the Matlab script. The image will then be extracted every 2 seconds and saved into location specified in a string variable in the Matlab code. After saving the raw data, there is an option for using the Matlab script written for pre-processing. This performs extrapolation and contrast increase of all the images in the specified database folder, as seen in appendix G.4.

5.2 APDS-9500 Setup

For the APDS-9500, a script is written to extract gesture information. This is done with the help of Arduino documentation and the APDS-9500 data sheet [12]. The Connections between the board and the sensor are derived from the sensor data sheet and the schematics of the sensors PCB:

1:	INT	$\rightarrow$	N.C.
2:	VIO	$\rightarrow$	N.C.
3:	GND	$\rightarrow$	GND
4:	MCLK	$\rightarrow$	N.C.
5:	MOSI	$\rightarrow$	N.C.
6:	ICS	$\rightarrow$	N.C.
7:	SDA	$\rightarrow$	SDA
8:	LEDR2	$\rightarrow$	N.C.
9:	SCL	$\rightarrow$	SCL
10:	SCK	$\rightarrow$	N.C.
11:	LEDA	$\rightarrow$	3.3 V
12:	VDO	$\rightarrow$	3.3 V

To go through the process step-by-step, it is first needed to see if the connection is good and the sensor is powered on when provided with the power source. An I2C scan is performed to check the connection. As a response, the correct address is sent back by the board. Next, to check if the sensor is powered on, a read command had to be sent to the right register, the response (0x20) indicates that the sensor is turned on correctly. The I2C communication protocol can only be used for device settings and the non-image mode that requires a lower data rate. The settings entail for example, sleep mode settings, interrupt settings and more. To write to a register, a communication function is used (given in appendix G.5). As first input, this function is given the general I2C address of the sensor, so the master/microcontroller knows what slave to communicate with. The second and third input are the registers address of which the content needs to be adjusted and the adjustment value respectively. An example of a call is given in appendix G.5.

It is not only necessary to write to registers, but also to read from registers like the interrupt register for gesture information. The register read function can be seen in appendix G.5.

The interrupt register contains information on why the sensor is interrupted and the sensor will be reset after it is read. In the case of gesture mode, the register can be read to retrieve information on the gesture that is detected. The main part of the script handles the reading of the register and the interpretation of what is read (given in appendix G.6).

The results after performance of a gesture can be seen in the Arduino's serial monitor prompt. A specific manual to set up the software and hardware for the APDS-9500 is given in appendix C.

Chapter 6

Machine Learning in Gesture Recognition

6.1 What is Machine Learning

Machine learning is model construction based on real data. Computers can detect patterns in data and learn to recognize these patterns in samples of data that are not used for model creation. There are different types of machine learning, there is supervised learning, unsupervised learning and reinforced learning:

Supervised Learning: When using supervised learning the input and output are provided to the machine. As an example: storage space in a USB and its price are correlated, which would impose a regression problem (relation between numeric input and output). Based on the knowledge of both storage space and price, a model will be built. Supervised learning can also be applied to a classification problem. For example, determining the manufacturer of the USB, based on its price, size and storage space. This will create a tag category called: "manufacturer", and during learning, the algorithm will try to couple the specific values to a manufacturer.

Unsupervised Learning: In unsupervised learning the model is given no knowledge of the possible outputs. As an example: An algorithm can be made by video streaming services based on one of many typical viewer profiles. This is how recommendations on streaming services are created: the machine assumes that viewers who watched similar series or movies, have similar taste. It will try to categorize viewers without knowledge on the movies, just the behaviour of persons watching.

Reinforced Learning: In reinforced learning no information will be given in the learning process. This is more like learning in real life, by trial and error. It can be compared with moving through a maze. The machine will take different routes to get to the middle, but will receive a time penalty every time it makes a wrong decision. It is implemented in for example the algorithms for self-driving cars and chess playing programs.

Machine learning has a broad scope of applications as well as a broad range of complexity. Simple linear regressions for example, can be regarded as machine learning, but so can the neural networks used for image recognition. The goal for machine learning in general, is to create a model capable of making (real-life) predictions with a minimal amount of error. The minimization of the error, is attempted with every iteration of the learning process, until (and if) the model converges and stops improving its accuracy on the training data. If data is trained too short, no converging will have taken place and the model will be less accurate. When the model is trained too long, there is a risk of overfitting the model. This means that the training has come to a point where it will involve outliers in the model, specifically to increase the accuracy on the training data. However, these outliers are not prevalent in real/new data, and the algorithm will perform worse in real life than on the training data. Noise or a blurred image can be seen as an example of an outlier. Overfitting also makes the model more complex then it has to be, causing predictions to take longer and increasing the parameters and storage space used for the code. No measures can be taken for training too short, but there are measures that can be taken against overfitting, these will be discussed in section 6.6.

The goal in this project is to identify the different gestures made in front of the sensor. The most effective way to use machine learning here is to define a number of gestures and labelling them as that gesture, which is the process used in supervised learning. Different gestures will induce different patterns to recognize. The challenge is to find out what sets of gestures can be used in a practical manner for the UI.

6.2 What is Image Recognition

Image classification is used increasingly in today's society. Since not that long ago, it has been applied to unlock phones with an image of an owners face. It has also been used for converting handwritten symbols and letters into a digital form. In image recognition, a mathematical model has learned to recognize specific ranges and patterns of pixels as an object/person. There are two different ways of applying image recognition. An image can either be searched for occurrences of an object/person or it can classify an entire image as such an object/person. Since this research is based on recognizing different gestures and it is not important where the gesture is taking place in the image, the entire image may be labelled as that gesture. If done correctly, no other objects will be present in the image and the focus will lie solely on the gesture in the image.

6.3 Convolutional Neural Networks

In the field of image recognition, convolutional neural networks (CNN) are used. The term CNN is made up of two parts: "convolution" and "neural networks". Here, the neural network is an imitation of the brains neural network, where each neuron decides if a signal/input is sent through to the next neuron. In the case of the CNN's used for image recognition, layers of neurons can be chosen by the engineer prior to training/learning. There are different kinds of layers, these will be discussed in section 6.4. The second term used in CNN's is convolution. Convolution is defined as a mathematical method to convert two different signals into a third signal. The first signal is the input image (or in case of multiple layers it can also be the output of another neuron), and the second is the filter that is applied in the current neuron. After the filter is applied in a neuron, that neuron will send the signal through to the next layer of neurons. A chain of such neurons and filters is illustrated in figure 6.1, together with the visualization of the effect that the filters can have on an image.

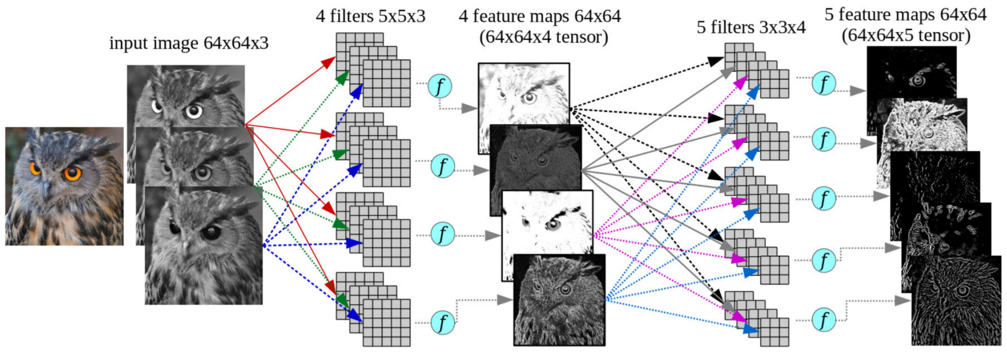


Figure 6.1: Visualization of an RGB image going through two convolutional layers.

6.4 Neural Network Layers

In neural networks the architecture consists of layers of neurons. In these layers, there are differences in for example size of the used filters or stride the filter takes after each iteration. A stride of 1 is minimal and a stride of 2 would mean that the filter will always have an uneven pixel in its top left corner (e.g. pixel 1,3,5 etc.). There is also a difference in weights and biases belonging to each neuron, because, one feature can be more important than the other. Weights and biases are the parameters that will be determined through the training process and are essential to the accuracy of the algorithm. Each neuron in a layer can have different settings for a weight, but the main task of one neuron is always equal to that of the other neurons in its layer. For CNN's the types of layer can be a convolutional layer, activation layer, pooling layer or fully connected layer. Each of these layers will be explained in the following subsections.

6.4.1 Convolutional Layers

The convolutional layer is partly discussed in section 6.3, convolution is a mathematical operation that uses two signals to create one signal. A neuron in the convolutional layer has its own filter, which can be seen as the first signal. The convolutional filter is an array of values that is to be moved over the image, which is the second signal. The frame of pixels in the image that the filter is on, is known as the local receptive field. During the training process, the filter moves over the image according to the stride settings that the training program has been given and thus changing the local receptive field with each iteration. When this process is completed, a new signal/image is created (the third signal) that shows whether the filtered features exists in the image or not. One of the differences between each neuron, is that each can have its own filter. These filters can differ in dimensions, but are equal for neurons from the same layer (e.g. 2x2, 4x3, 5x5 pixels etc.). Weights and biases are different between all layers and neurons in those layers. The filter coefficients (or weights) are the values used to create a linear combinations with the pixels in the image. Convolutional layers are commonly followed by activation layers.

6.4.2 Activation Layers

Activation layers are transform values, usually between 0 and 1 or -1 and 1. One of the most popular and effective activation functions is the Rectified linear unit (ReLU), which converts all negative values to 0. These values are insignificant and not occurring in the original image, but can be created when for example putting an image in through convolution layer. All positive values remain the same. A visualization of ReLU can be seen in figure 6.2. Next to ReLU, there are also other layers that can be used like sigmoid, in which lower values are made less significant and values above a certain threshold are made more significant, while also restricting all values to be either positive or zero. Other examples of activation functions are PreLU and tanh, however ReLU is the most commonly used in CNN's. Activation functions can be seen as a way to optimize and keep control over the learning process. This role in the learning process will be explained in section 6.5 about backpropagation.

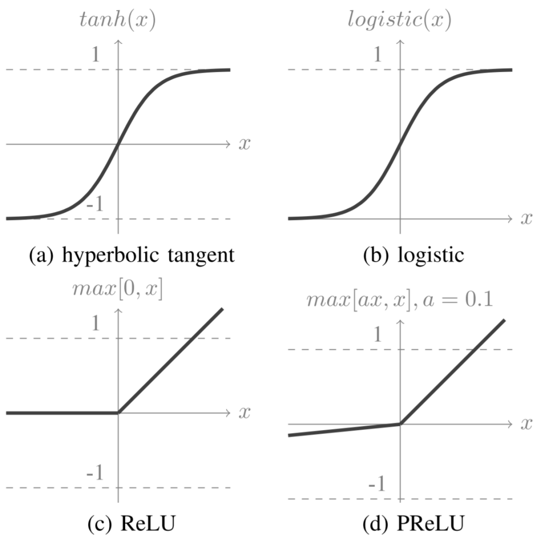


Figure 6.2: Visualization of activation functions.

6.4.3 Pooling Layers

Pooling layers are commonly applied after a few convolutional layers. The main goal of pooling is reducing the size of the images (officially called tensors) passing through the upcoming layers. There is minimum, maximum and average pooling. To take for example maximum pooling, in which an x by y pixel frame is slid over the image. Each step has a maximum pixel value inside that frame, this is chosen as a new pixel for the output image, decreasing the size of the input image significantly. There is however, also a 3rd dimension which is visualized in figure 6.1. This dimension will increase when put through a number of convolution layers, making the reduction of the other dimensions very convenient, since it compensates for the increasing demand in computational power. Another reason for applying these pooling layers, is to be able to process the images in different scale spaces. For an 8 by 8 image, pooling is unlikely to be necessary, since the images are already low resolution and probably lose valuable information when pooled.

6.4.4 Fully Connected Layers

These layers are put at the end of the neural network, converting the array of inputs into a single weight scalar value. This is done by first converting the array(s) of x by y pixels into a 1 · x · y vector. After this, a linear combination of this vector is taken with the weights in the fully connected layer. This will produce the neurons weight as seen in equation 6.1.

$$P(y = c|x; w; b) = softmax_c(X^T w + b) = \frac{e^{X^T w_c + b_c}}{\sum_j e^{X^T w_j + b_j}} \tag{6.1}$$

Where P is the probability vector, giving the probabilities of the image belonging to each of the possible classes. c is the input vector, w is the weight of a neuron and b is the bias belonging to each of the neurons. The weights will indicate the chances of the image belonging to the neurons class with a number between 0 and 1. A visualization of the fully connected layer can be seen in figure 6.3.

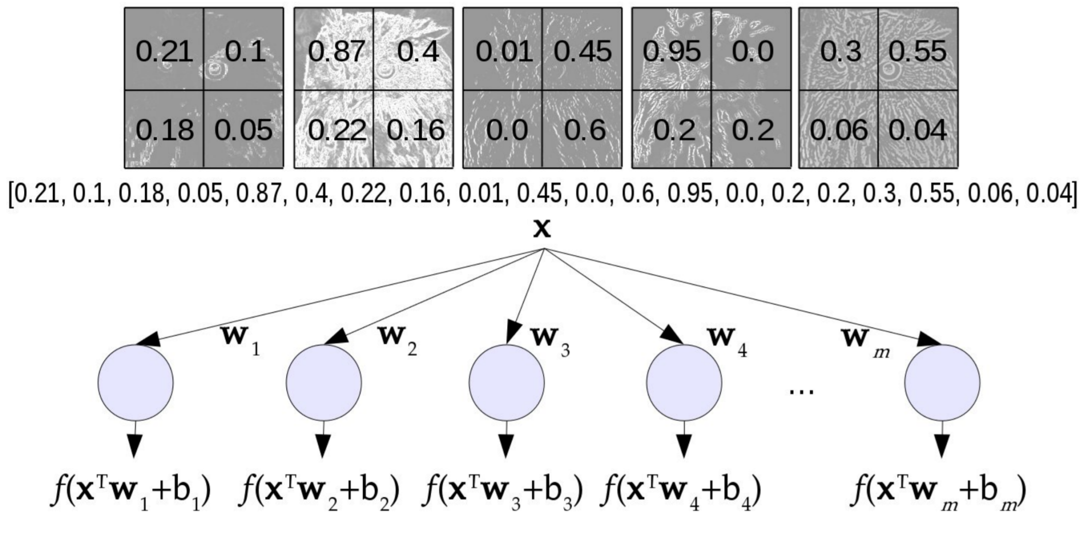


Figure 6.3: Visualization of the feature maps, shown earlier in figure 6.1, going through a fully connected layer.

6.5 How Training Takes Place

In order to create an understanding of how a model is trained, the concepts of back propagation and the loss function have to be understood. The loss function indicates how much error takes place as a result of your current weights and biases. The formula for the loss function used in classification problems is the cross-entropy loss function:

$$L = -\sum_{i=1}^{n} t_i log(p_i) \tag{6.2}$$

Where t_i is the true classification outcome, p_i is the estimated outcome from the softmax layer and n is the number of different classes. Note that it does not matter whether the input is a vector or a scalar, the result of the loss function is always a number between one and zero. And it is an average of multiple training examples (unless batch size, explained later in section 6.6, is one). The goal is to make the loss function approach zero, by tweaking the weights and the biases of the model. A loss function of one indicates a 100 percent certainty of having estimated the right class. As is common in optimization problems, a derivative needs to be calculated. The approach of training, is to estimate the slope of the loss function when considering one weight or bias. When the slope is known, a small step can be taken in the downward direction of that slope in order to decrease the loss. That small step is regulated by the learning rate as shown in equation 6.3 and 6.4.

$$w_n = w_o - \mu \cdot \frac{\delta L}{\delta w_o} \tag{6.3}$$

$$b_n = b_o - \mu \cdot \frac{\delta L}{\delta b_o} \tag{6.4}$$

Where w_n is the new weight and w_o is the old weight. In equation 6.4 weights are replaced by biases. N is the learning rate and L is the loss function. The learning rate is determined by the user, it cannot be too large or the model will iterate over too large steps, preventing the model from converging. On the other hand, when the learning rate is too small the model will take very long to converge. The old weights and biases are known, which leaves us with the derivative of the loss function over the chosen weight. The problem here is that there are so many weights and biases that it is hard to see individual contributions and determine in which way to adjust the model for optimization. This is where backpropagation comes in. In backpropagation, the starting point lies at the end of the neural network. Take for example the simple neural network in figure 6.4, where the loss function can be calculated with the current models output $\hat{y}$.

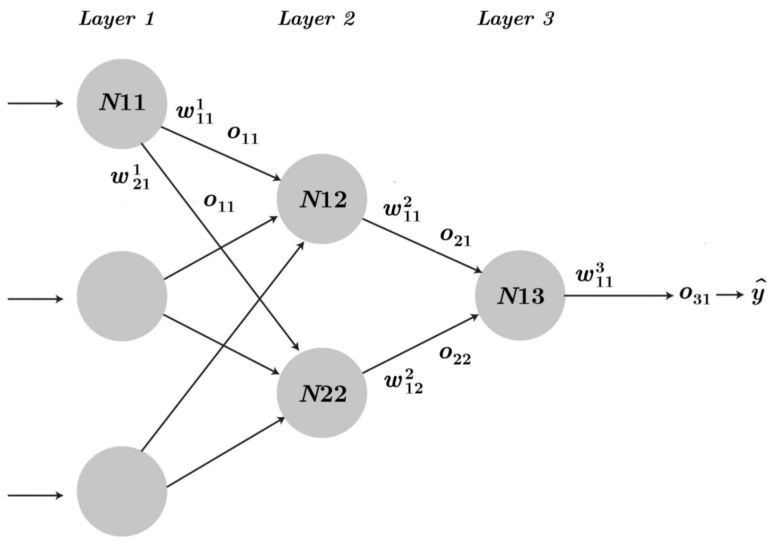


Figure 6.4: Drawing of the last 3 layers of a neural network, with weights and outputs indicated with w and O respectively.

The derivative of the loss function with regard to the weight of the last neuron is easiest to derive, as seen in equation 6.5.

$$\frac{\delta L}{\delta w^3_{11}} = \frac{\delta L}{\delta O_{31}} \cdot \frac{\delta O_{31}}{\delta w^3_{11}} \tag{6.5}$$

As seen in equation 6.5, the chain rule is applied to the equation. There is no direct effect of the weight on the loss function. However there is a measurable influence of the output O_{31} on the loss function and an influence of w^3_{11} on O_{31}. This approach is used throughout the entire chain. Take for example w^2_{11}, of which loss function derivative is given in equation 6.6.

$$\frac{\delta L}{\delta w^2_{11}} = \frac{\delta L}{\delta O_{31}} \cdot \frac{\delta O_{31}}{\delta O_{21}} \cdot \frac{\delta O_{21}}{\delta w^2_{11}} \tag{6.6}$$

As can be seen, the examples are only taken for neurons that have one route to the final output. When considering neurons that have multiple ways to get to O_{31}, all of them are to be added. Take for example w^1_{11} which has two ways of getting towards O_{31} ($O_{11} \to O_{21} \to O_{31}$ and $O_{11} \to O_{22} \to O_{31}$), which can be seen in equation 6.7.

$$\frac{\delta L}{\delta w^1_{11}} = \frac{\delta L}{\delta O_{31}} \cdot \frac{\delta O_{31}}{\delta O_{21}} \cdot \frac{\delta O_{21}}{\delta O_{11}} \cdot \frac{\delta O_{11}}{\delta w^1_{11}} + \frac{\delta L}{\delta O_{31}} \cdot \frac{\delta O_{31}}{\delta O_{22}} \cdot \frac{\delta O_{22}}{\delta O_{11}} \cdot \frac{\delta O_{11}}{\delta w^1_{11}} \tag{6.7}$$

The number of derivatives inside one network can increase exponentially, this is only a simple example. The principle remains to follow the chains of connection back towards the neuron in question, which causes a lot of the derivatives to be used for multiple calculations. In this manner backpropagation and gradient descent are used to make the model converge towards a point of minimal error.

The same method is valid for calculating the biases of the neurons. Equation 6.8 is a copy of equation 6.5, where all the weights w are replaced with biases b.

$$\frac{\delta L}{\delta b_{3,1}} = \frac{\delta L}{\delta O_{3,1}} \cdot \frac{\delta O_{3,1}}{\delta b_{3,1}} \tag{6.8}$$

6.6 What are the Different Hyperparameters

Before training, some parameters are set for the user, like the initial weights/filters, these are based on a Gaussian distribution. However, a lot of choices can still be made, these choices have a large effect on the accuracy of the model. The choices that can be made will be mentioned here, as well as what should be kept in mind when making these choices.
Epochs: The number of times that the training data is used to train the model. After every epoch the entire training set is applied to the model. Training with too few epochs can cause your model to stop training when its not optimal. Training over too many epochs can cause overfitting the model, when the right measures are not taken.
Batch Size: Inside an epoch there will be smaller steps of training, each step taking into account one batch size of data. If an entire dataset is used, computation of derivatives will become very time intensive. Using small batches will not affect the result as long as training is done long enough. Estimations of the derivative are still sufficient for finding a good minimum. Because estimations of the derivative are used, randomness is added to the

model training. Decreasing the chances of overfitting a model. The total number of times that weights and biases are changed in training is equal to database size/ batch size ∗ epochs.
Learning rate: Learning rate indicates what step size is taken in adjusting the weights and biases of the model at each iteration. Small values will increase training time needed, but too large sizes can prevent the model from converging. This because the steps will be too large to stay inside a local minimum.
Optimizer: An optimizer can be seen as an extra tool for the training process. They are not required, but are very useful. It adjusts parameters while training. ADAM (Adaptive Moment Estimation) for example is the name of a popular optimizer that adjusts the learning rate based on the steps taken in accuracy during the training process. It combines two qualities of other optimizers called ADAGrad and RMSprop. A comparisson can be found in Çelebiler (2021) [27].
Activation function: The Activation function has already been discussed in section 6.4.2. Because of the use of activation functions/layers, the value of the derivative of the loss function will adjusted. When working with the derivative of a loss function, there is a chance for that derivative to increase drastically. Activation functions keep the derivative from doing this and supports finding a minimum in a steadier fashion.
Dropout Rate/Layer: A dropout layer is another measure against overfitting and making a model robust against noise. The function of the dropout layer is to randomly set input values (for the layer it is behind) to zero, in order to create a disturbance in the process. The goal is to make the model familiar with disturbances and better at guessing even if the input data contains anomalies. If the dropout is too high it can negatively affect the training. Dropout is given with a value that describes the chance for a neuron to drop out during one training iteration.

6.7 Relation Between Model Size and Structure

Accuracy is not the only measure of how good the algorithm fits the application. Model size and parameters are important for two reasons. The first is that microcontrollers only have limited RAM space for uploading code. The second is that less parameters cause less processing demand and thus faster recognition. Of the four layers, convolutional layers and fully connected layers have the most effect on model size. They have ”learnable” parameters that play a significant role in the model size and processing demands. Pooling layers cause a decrease in parameters of the fully connected layers, because the fully connected layers parameters are dependent on tensor size. The number of convolutional layer parameters are dependent on filter size and the amount of neurons in the current and previous layer.

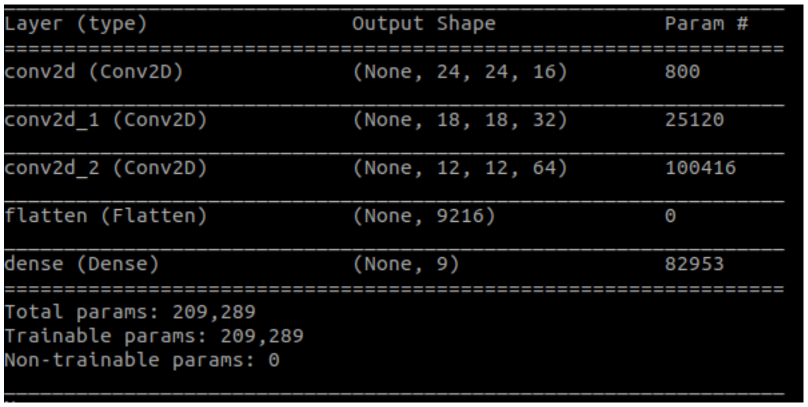
For convolutional layers part of the number of parameters is equal to the filter size times the number of tensors it is given by the previous layer. These tensors are given as input to all the neurons in the current layer, and on top of that, each neuron in the current layer has its own bias. So the formula for parameters in the convolutional layers can be described as

$$p = ((l \cdot h \cdot w) + 1) \cdot c) \tag{6.9}$$

Where l and c are the number of neurons in the previous and current layer respectively. p is the number of filter weights in the current layer and h is the current filter height and w is the current filter width. The 1 in the equation, stands for the bias in each neuron. As an example figure 6.5 is given. Where a 30 x 30 image is put into three convolutional layers of size 7 x 7. The numbers of parameters in Conv2d_1 is equal to $((7 \cdot 7 \cdot 16) + 1) \cdot 32 = 25120$.

The last convolutional layer passes an x amount of tensors (based on its number of neurons) to the fully connected layer. For each pixel in this tensor the fully connected layer contains a weight. So as an example, if the last convolutional layer contains 8 neurons and each neuron outputs a 6 x 6 image towards the fully connected layer, then the number of parameters used in the fully connected layer is (8 x 6 x 6 =) 288. Finally, the fully connected layer contains one more parameter for each possible outcome, the bias for each neuron. So if there are 5 possible outcomes, the total number of parameters is now 293.

Convolutional layers have an output tensor equal to the image size minus the filter size - 1. When for example a 7 x 7 filter is applied to an 30 x 30 image, the output tensor will be 24 x 24. This means that larger filter sizes in convolutional layers have a decreasing effect on the parameters of the fully connected layer, however, the number of parameters in the convolutional layer is increased. For a neural network with for example three layers, it is better to use bigger filter sizes, because model size decreases more due to the lowering of parameters in the fully connected layer, compared to the increase in parameters in the filters. Because the fully connected layer is highly dependent on the size of the tensors it is given as input.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 16)        800
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 18, 18, 32)        25120
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        100416
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 9)                 82953
=================================================================
Total params: 209,289
Trainable params: 209,289
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.5: Example of the numbers of parameters used in a model containing 3 convolutional layers with 7x7 filters, and a fully connected layer that outputs 9 options.

Chapter 7

Keras: Structure of the Machine Learning Algorithm

Keras is an API for machine learning available as a Python library. It provides functions that are able to load data, pre-process data, build models, train models, save/load models and more. Every function has its own parameters that can be chosen by the engineer (see section 6.6). The models built in this project are based on the available documentation on the Keras website [28]. As long as Python and the necessary libraries are installed on the computer, the scripts for training can be run.

7.1 Image Recognition Training

For image recognition, a database with training and validation images has to be retrieved. Keras is made such that a folder can be indicated as the "main database folder". This folder contains sub-folders with images, each subfolder consists only of images of one specific category that is to be recognized. A function is used to appoint a folder and read in the data with specific hyper parameters (code in appendix G.1). Next, the data should be preprocessed by keras for use, for example shuffling the images and loading them in the right format. A model will then be built consisting of a predetermined amount of layers. Every layer has its own filter size, neuron size and activation function. Layers can be added in the keras API by using the "addlayer" function. Every call to this function adds one layer to the end of the network. In appendix H.2, it can be seen that options are built in for multiple structures. Starting with a convolutional layer, with as input the images. This is followed by more convolutional layers, which each, can be followed up by a pooling layer. All these layers can be set with the hyperparameter settings given in the code in appendix H.3. After setting specific hyper parameters to test, the model can be trained with the "fit" command. This will run a training algorithm for an X amount of epochs times an Y amount of batches, where $Y = DB\_size/Batch\_Size$. This will be done over all the possible combinations of hyperparameters given.

7.2 Optimizing the Model

Machine learning is an optimization problem on its own, while the hyperparameters should also be optimized to create the best model. The process of finding the best fitting model, is highly dependent on these parameters. Choosing a too high batch size for example, will lead to perfect estimations of the derivative of the loss function. This will pose a problem for the training, since it will be highly likely that the model converges to a local minimum that is not optimally accurate. A script is written to train multiple models with different parameters (given in appendix H.3). Each of the listed parameters will be combined with each other to create unique models. The parameters and functions are listed in a file named *train_models_with_multiple_parameters.py*. An example of code is shown in appendix H.2. The number of possible structures in terms of hyperparameters and layers is endless. Especially when taking into account multi-layer structures, with each layer having the option to contain different sizes and filters. Optimization is done by testing a broad range of parameters and layer structures and zooming in on the parameters that do well. As a result, each model is saved in a specified folder. The hyperparameters, confusion matrix and accuracy of the model is written to a ".txt" file. The Python script remembers the best performing model and gives that as a result when its finished running. Based on the best models, new assumptions can be made and new training sessions can be initiated.

7.3 Testing the Model

As mentioned in section 6 the image database is split into 3 unique sets of data (See figure 7.1). The first is the training set, this is the set over which is trained, meaning that weights and biases will be adjusted based on the derivatives/loss function from this data. Most of the database will be used for this purpose. The second is the validation set, this is used as an extra control mechanism against overfitting. When only using the training data to determine accuracy, the model might unnoticedly converge to solutions that will fit the training data and not the general/realistic cases. Whenever the Accuracy of the validation split starts falling behind compared to the training accuracy, it should raise suspicion. The last set is the test set, this test is kept away from the training process at all times. Only in the end, the model will be used to make an accuracy measurement with the test data. The test set is independent and the same for all models.

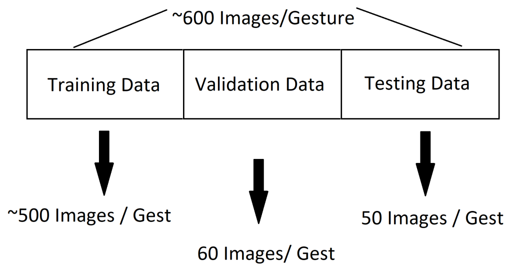


Figure 7.1: One of the ways data is split in the training of the artificial neural network. The test split is created manually.

In this case it is chosen to use a test dataset of 50 images per gesture. At the end of all model training, images are classified by the model 1-by-1. The classification results are shown in a confusion matrix. This confusion matrix shows the gestures contained in the images in its row and the estimations made by the model in its columns. Thus, if 100% accuracy is reached, only the diagonal will be filled with numbers higher than zero. The code for looping through the classification, creating the confusion matrix, calculating the accuracy and remembering the best model is done by a part of the Python script given in appendix H.4. The total flow of data and products from the sensors to the machine learning algorithm can be seen in figure 7.2.

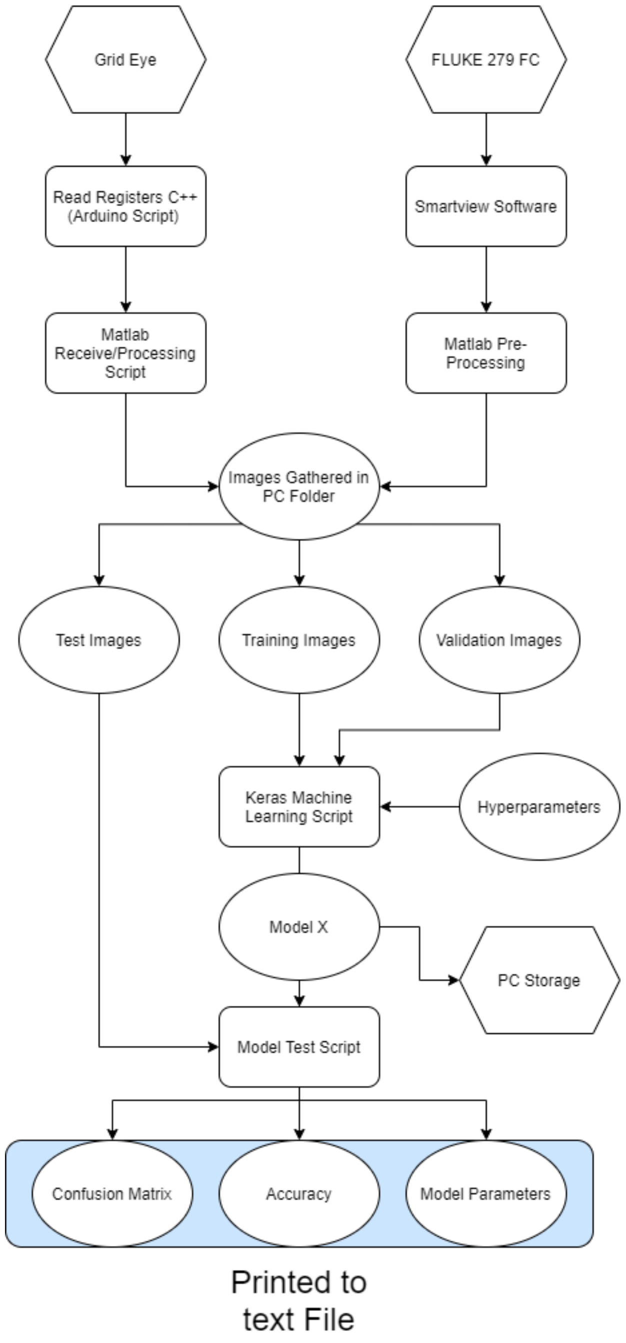


Figure 7.2: Total flow as discussed in current and previous chapters, displaying the gathering of the images to the outputs of the model.

Chapter 8

Method of Data and Gesture Acquisition

8.1 APDS-9500

The APDS-9500 is used in its gesture mode. The sensor is used as a reference for the accuracy of existing gesture sensors, thus no machine learning is used. The sensor is set up as described in chapter 5, the C++ code is uploaded into the Apollo 3 microcontroller with the use of the Arduino application. The serial monitor will tell the researcher what gesture is measured, by printing out that measured gesture. For the APDS-9500 in gesture mode, 8 gestures are available.

1. Swipe up
2. Swipe down
3. Swipe left
4. Swipe Right
5. Move towards sensor
6. Away from sensor
7. Clockwise rotation

8. Counter clockwise rotation

However, the accuracy for the last 3 gestures is not as good as the first 5. Rotational movements of hands are very hard to identify for the sensor, most attempts result in wrong identification. The other gesture that mostly results in wrong sensor identification, is moving away from the sensor. In order to move away from the sensor it is first required to get close to the sensor, this mostly results in a premature estimation of another (wrong) gesture (e.g. swipe right). This is why only the first 5 gestures are used for the tests. A total of 8 participants completed every gesture 10 times (50 total). The participants ages range from 25 to 60 years. Instructions for swiping are given by the researcher to the participants. Each participant is given a minute to get familiar with the sensor and its output. The gestures are randomized and executed in about 2 minutes by each participant. Feedback is given, only when there is a difference in the requested and measured gesture. Results are placed in a confusion matrix and accuracy is measured as the percentage of correctly identified gestures.

8.2 AMG

The Grid Eye is set up as described in chapter 5. For the Grid Eye, 6 participants are asked to make 5 different gestures in front of the sensor. These 5 gestures are the horizontal finger, vertical thumb, O-sign, pinkindex-sign and a flat hand. The gestures are depicted in figure 8.1.

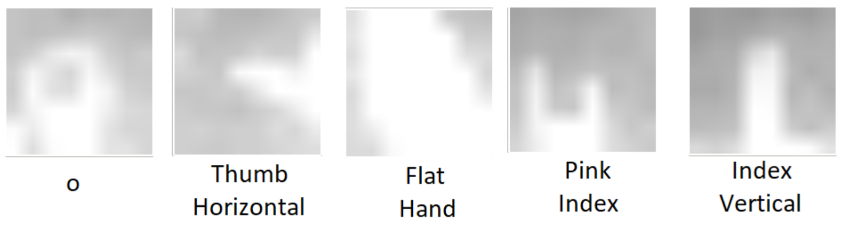


Figure 8.1: Examples of 8 by 8 images made with the AMG Grid Eye, the ".png" format smoothens the pixels out to make it look more natural.

The participants ages range from 25 to 60 years. Instructions are given to keep the hand still in the desired position, small variations/movements

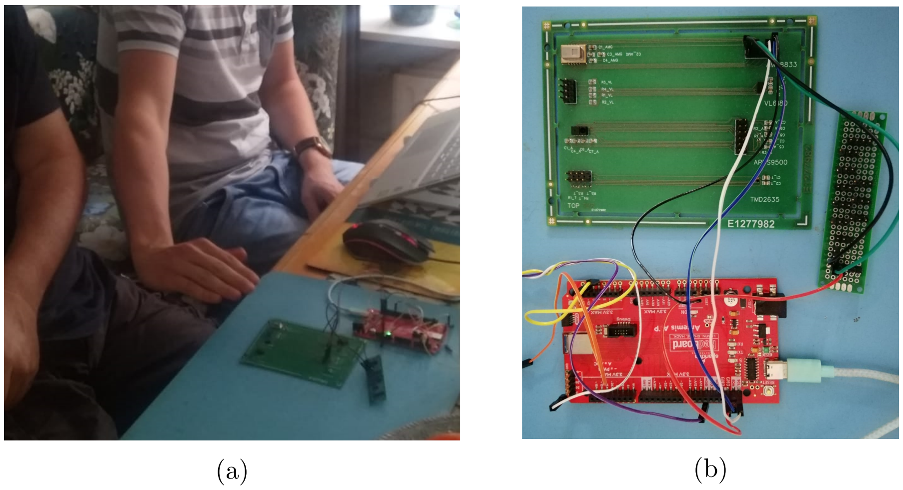


Figure 8.2: The setup for the experiments with the AMG Grid Eye. A) Researcher explaining the different gestures to the participant. B) Artemis Redboard with Apollo 3 connected to the AMG sensor (top left of the green PCB)

are allowed after a few images, these have to be done directly after an image is taken, in order to not blur the next image (2 seconds later) with movement artefacts. The setup can be seen in figure 8.2.

As explained in chapter 7, the Python machine learning algorithm is used, in combination with a database of about 600 images per gesture. Different arrays of hyperparameters are varied in the following ranges:

- Number of Epochs = 10 to 200
- Batch Size = 10 to 2400 (10, 50, 100, 200, 400, 600 etc...)
- Validation Split = 0.1 or 0.2
- nr of Conv. Layers = 0, 1 or 2
- Size of Conv. Layers = 8, 16, 32, 64 (in ascending order)
- Max Pooling = None
- filter Size = (2,2) or (3,3)
- Activation Function = relu, sigmoid, tanh, elu, selu

- Optimizer = Adam, ADAGrad, RMSprop, Nadam, SGD, ADAdelta
- Dropout Rate = 0 or 0,2

When a variation is ineffective, the training is aborted in order to remove the ineffective parameter. These models are all tested with an independent test database of 50 images/gesture. Hyperparameters, model number, confusion matrix and accuracy are printed to a text file for later analysis. The best 10 models are taken from the trained model folder. This is done for the raw AMG data, but also differently pre-processed versions of the raw database are used as a comparison and attempt to improve the models accuracy. The first form of pre-processing is a linear extrapolation in order to increase image size from 8 by 8 to 12 by 12. The second is a contrast increase, by spreading the entire variation in the image pixels over 8 bits. The last form of pre-processing is first increasing the contrast and then an extrapolation.

8.3 FLUKE 279 FC

The FLUKE 279 FC is used as an passive IR camera. In order to see whether an increase in pixels (to an amount similar to that of the APDS-9500 in image mode) has a significant effect on the accuracy of the model. It is mounted on a multimeter and captures images that it can send to the PC using the Smartview software package. For the entire instructions to use the FLUKE and store images on the PC see Appendix E. The FLUKE has higher resolution and thus the capability of making more gestures recognizable. In order to do a good comparison with the lesser quality images from the Grid Eye, the same 5 gestures are used. In addition to these gestures, 4 more gestures are added to see whether more pixels can result in more classification options, with a similar accuracy. Example images of all the gestures used with the FLUKE, are shown in figure 8.3.

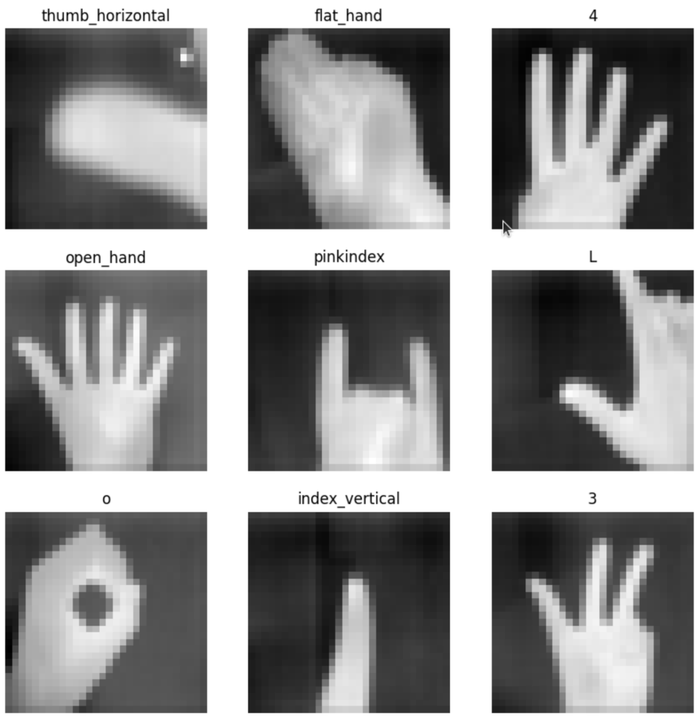


Figure 8.3: The 9 different gestures as captured with the FLUKE 279 FC.

At first the raw images are used. Then the images are downscaled to a lower resolution to see what the influence of this step is on the accuracy. Apart from the downscaling, no other forms of pre-processing are applied to the images of the FLUKE. Hyperparameter variations are similar to those of the Python machine learning algorithm used for the Grid Eye, however due to the increase in resolution, more options are available and applied:

- Max Pooling = (2,2) layer applied after either layer
- filter Size = (2,2) up until (10,10) varying per layer

Chapter 9

Results

9.1 APDS-9500

For the APDS-9500, data is gathered from 8 participants. Each participants results are processed in a confusion matrix. In such a matrix the rows indicate the gesture made and the columns indicate the gesture estimation by the sensor/software. That means that all the numbers along the diagonal of the matrix, are the number of correctly estimated gestures. The individual results can be found in appendix F, the summed results are given in table 9.1. In Appendix F it can be seen that most variation takes place in the "moving towards" gesture. Variation in accuracy for that gesture is between 50% and 80%, while the other gestures are 100% accurate in every participant, except

Summed results	Right	Left	Up	Down	Towards	Accuracy
Right	80	0	0	0	0	100%
Left	0	80	0	0	0	100%
Up	1	0	76	2	1	95%
Down	0	0	0	80	0	100%
Towards	0	9	16	2	53	66,25%

Table 9.1: Confusion matrix containing the results of the experiments with 8 participants each performing 50 gestures, total accuracy is 92.25%.

for 2 participants that made mistakes in the ”swipe up” gesture.

9.2 AMG

9.2.1 Training With Raw Data

At first an attempt is made to do image recognition on the raw data. Due to the sensor range, which is -20 to 80 degrees Celsius, the contrast between the hand and the environment is not that big. The incoming raw data varies between approximately 40 and 90. While the values are 8 bit and can thus range from 0 to 255. The first attempts are manually setting the hyperparameters to get a feel of the hyperparameter limits. It is seen that 2 or more convolutional layers with sizes of 64 or higher, tend to give a high training accuracy, but a low test and validation accuracy. This means that the model tends to find a solution to the problem with many neurons, however, it overfits the training data entirely. Validation splits of 0.1 or 0.2 are common in machine learning and should not absorb too much training data, in our already small database. The majority of higher accuracy models, are those with a validation split of 0.1. Batch sizes are first set to about 128, this causes the model to become best at putting all guesses on one gesture (20% accuracy). Better models start to form around batch sizes of 300. The number of epochs used is 50 at the start, but since more training did not harm the model it is chosen to use more epochs, as long as the overfitting does not become a problem. To help prevent overfitting, a dropout layer is added in between the last convolutional layer and the fully connected layer. This sets random inputs to the last layer to zero, like a self induced noise to make the model more robust.

A total of 200 models is made, the best performing model had an accuracy of 74.8%. Hyperparameters for this model are:

- Number of Epochs = 100
- Batch Size = 500
- Validation Split = 0.1
- Size of Conv. Layers = 0

- Optimizer = Adam
- Dropout Rate = 0

From this model and the other models that performed (relatively) well a few things are noticed. For the activation function, ReLU performed the best. Performance with regard to batch size is varying, so more variation in batch size can be applied. The number of convolutional layers used is 0 in the 10 best performing models, so only the fully connected layer is used. Other variables vary randomly in the top 10 of models, so they are of less influence to the result. In the next training session variation is increased in batch size and number of epochs. Convolutional layers is set to 0 and all other parameters are varied mildly. The result of this training is a best model that has an accuracy of 79.2%. So there is slight improvement compared to the latest model. The new best models are analysed again. For the best model these parameters are used:

- Number of Epochs = 100
- Batch Size = 700
- Validation Split = 0.1
- Size of Conv. Layers = 0
- Optimizer = Adam
- Dropout Rate = 0.2

The differences between the latest model and this one can be found in the batch size and the dropout rate. More variation to these hyper parameters did not improve the result. Another 20 training sessions are performed with the same parameters as the currently best model. This is done because the randomness added to the training by the dropout, batch sizes and shuffling (of the DB) might hold back small improvements. However, this did not improve the model. The model remains the best performing model for the raw data set and its confusion matrix can be seen in table 9.2.

Summed results	Index Vertical	Flat Hand	Pinkindex	O	Thumb Horizontal	Accuracy
Index Vertical	11	5	6	28	0	22%
Flat Hand	0	47	0	3	0	94%
Pinkindex	0	0	48	2	0	96%
O	0	4	3	43	0	86%
Thumb Horizontal	0	1	0	0	49	98%

Table 9.2: Confusion matrix of best 8 by 8 AMG raw data model. Accuracy = 79.2%.

9.2.2 Training With Extrapolated Data

As an attempt to improve recognition, (bi-)linear extrapolation is applied over the images in the databases. 8 by 8 images are bi-linearly extrapolated to 12 by 12 images. This way it is possible to apply a 2 by 2 or 3 by three filter, without decreasing image resolution into a too small amount of pixels. This is also a way of artificially creating more information for the algorithm to train on. The same variations as used with the raw database are trained over these 12 by 12 images. As a result over 200 models are trained of which the best model achieves an accuracy of 88.75% . Its parameters are as follows:

- Number of Epochs = 100
- Batch Size = 600
- Validation Split = 0.1
- Size of Conv. Layers = 0
- Optimizer = Adam
- Dropout Rate = 0.2

The top 10 of extrapolated models showed similarities with this model. The majority of the models used validation splits of 0.1. The models do differ, in dropout rate (0, 0.1 and 0.2). The number of epochs used is mostly 100 and the Batch size is optimal around 500. New variations are applied, this time holding the convolutional layers at 0 and validation split at 0.1. This while varying the dropout rate, nr of epochs and batch size. This results in a slightly improved model with an accuracy of 89.6%. The following hyperparameters are used for the model: follows:

- Number of Epochs = 150
- Batch Size = 500
- Validation Split = 0.1
- Size of Conv. Layers = 0
- Optimizer = Adam
- Dropout Rate = 0.2

After this the same model is trained another 20 times to see if improvement due to randomness could take place. This is not the case. The confusion matrix can be seen in table 9.3.

Summed results	Index Vertical	Flat Hand	Pinkindex	O	Thumb Horizontal	Accuracy
Index Vertical	30	3	7	10	0	60%
Flat Hand	0	49	0	1	0	98%
Pinkindex	0	0	49	0	0	98%
O	0	3	1	46	0	92%
Thumb Horizontal	0	0	0	1	49	98%

Table 9.3: Confusion matrix of best 8 by 8 AMG extrapolated data model. Accuracy = 89.6%.

9.2.3 Increasing Image Contrast

Next to extrapolation, an attempt to increase contrast between the hand and the environment is made. The lowest value pixel and highest are stretched to be 0 and 255 respectively. This data is used to train models with the same hyperparameters as before. Of the best performing model, which achieves an accuracy of 91.6%, hyperparameters are shown here:

- Number of Epochs = 150
- Batch Size = 700
- Validation Split = 0.1
- Size of Conv. Layers = 0
- Optimizer = Adam
- Dropout Rate = 0.2

Noticeable here is, that there are a lot of models with accuracies near 90% (top 20 model accuracy > 89%). Similarities between the well performing

Summed results	Index Vertical	Flat Hand	Pinkindex	O	Thumb Horizontal	Accuracy
Index Vertical	34	1	13	2	0	68%
Flat Hand	0	50	0	0	0	100%
Pinkindex	0	0	50	0	0	100%
O	0	2	1	47	0	94%
Thumb Horizontal	0	0	1	1	48	96%

Table 9.4: Confusion matrix of best 8x8 AMG model where contrast of the images is increased. Accuracy = 91.6%.

models are the validation split of 0.1 and the use of only the fully connected layer. A slight majority of about 70% uses 150 epochs to train the model. Batch sizes vary around 500 again, however this time a lot of good performing models do well with a batch size of 300. ADAM is the best performing optimizer and the dropout rate varies to be either 0 or 0.2. Another training session is started with varying batch sizes, epochs and dropout rates. While keeping validation split, number of convolutional layers and optimizer the same. This results in no improvements, even when the model is checked for random improvement for 20 equal hyperparameter trainings. The confusion matrix of the best model is shown in table 9.4.

9.2.4 Extrapolation and Increase in Contrast

Also a combination for both of extrapolation and contrast improvement is taken. The same variations are applied to these models as before. The best model achieves an accuracy of 92.4% with the following hyperparameters

- Number of Epochs = 150
- Batch Size = 700

Summed results	Index Vertical	Flat Hand	Pinkindex	O	Thumb Horizontal	Accuracy
Index Vertical	37	0	11	2	0	74%
Flat Hand	0	47	0	2	1	94%
Pinkindex	0	0	50	0	0	100%
O	1	0	2	47	0	94%
Thumb Horizontal	0	0	0	0	50	100%

Table 9.5: Confusion matrix of best 8 by 8 AMG model where contrast of the images is first increased and then the image is extrapolated to 12 by 12 pixels. Accuracy = 92.4%.

- Validation Split = 0.2
- Size of Conv. Layers = 0
- Optimizer = Adam
- Dropout Rate = 0.2

In the best models the batch size varies between 500 and 700, the validation split also varies between 0.1 and 0.2. Number of convolutional layers is 0 again and the best number of epochs is 150. Based on these results a new training session is started, varying the validation split, batch size and dropout rate. While keeping the number of epochs, optimizer and number of convolutional layers equal. No improvement is made in the models accuracy in this training session, as well as in retraining the model with same parameters 20 times. The confusion matrix of the best performing model is shown below in table 9.5

9.3 FLUKE 279 FC

9.3.1 FLUKE 279 FC With 5 Gestures

The FLUKE is used on one subject. The subject is asked to perform the 5 gestures that are also used in the Grid Eye's experiments. Next to this, four extra gestures are performed to be able to train with different/more gestures than previously applied. A total of 5400 images are taken for the machine learning database (600 images times 9 gestures). The gestures are rescaled using a Matlab script, the applied resolution is 30 by 30 pixels (see figure 8.3). This resolution is chosen in order to mimic some of the qualities in small imaging sensors available today (like the APDS-9500). First it is tested whether the images from the FLUKE are better for predicting gestures than the Grid Eye's. The same 5 gestures are applied in the same quantity. Batch sizes are varied and up to 3 convolutional layers are applied with an option of pooling on the first layer (similar to Grid Eye training). Accuracy is determined by 50 test images/gesture that are separated from the training process. The best model achieves an accuracy of 97.2% with the following parameters:

- Number of Epochs = 100
- Batch Size = 500
- Validation Split = 0.1
- Size of Conv. Layers = 16
- Pooling Layers = None
- Activation Function = ReLu
- Optimizer = Adam
- Dropout Rate = 0

Variation is applied to the number of batches and the number of epochs is lowered, because the performance is also higher than 90% when using up to 20 epochs. The size of the convolutional layers is working well with one convolutional layer of 16 neurons. Variations are applied with multiple layers and layer sizes as well. The best network remains to be the same as above. Its confusion matrix is in 9.6.

Summed results	Index Vertical	Flat Hand	Pinkindex	O	Thumb Horizontal	Accuracy
Index Vertical	47	0	3	0	0	94%
Flat Hand	0	50	0	0	0	100%
Pinkindex	0	0	50	0	0	100%
O	0	0	0	50	0	100%
Thumb Horizontal	0	0	1	3	46	92%

Table 9.6: Confusion matrix of best 30 by 30 FLUKE data model, based on same 5 gestures used with the Grid Eye. Accuracy = 97.2%.

9.3.2 FLUKE 279 FC With 9 Gestures

After this, the attempt is made to train 9 different gestures shown in figure 8.3. Training takes place with the same parameters as before, the main difference in this model is that the fully connected layer has 9 outputs instead of 5. The best model achieves an accuracy of 99.3% with the following parameters:

- Number of Epochs = 20
- Batch Size = 400
- Validation Split = 0.1
- Size of Conv. Layers = [16, 32, 64]
- Pooling Layer = None
- Optimizer = Adam
- Size of filters = [(6, 6),(6,6),(6,6)]

- Activation Function = ReLu
- Dropout Rate = 0

Because this model functions well when using multiple convolutional layers, a lot more variation is possible. Filter sizes can be varied and pooling can be applied after different convolutional layers. To check the influence of different batch sizes, a variation is applied with different activation functions on a 3 layer model as seen in figure 9.1.

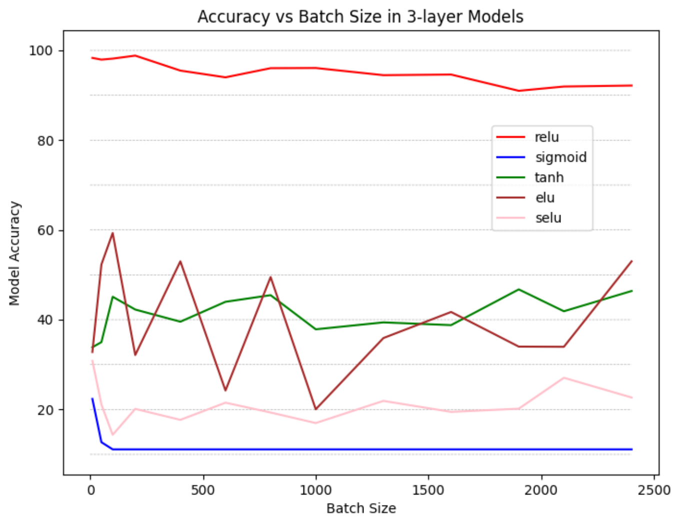


Figure 9.1: A graph displaying the accuracy plotted against batch size, with different activation functions. For the well performing activation functions, small sizes affect the accuracy however it drops slightly when increasing.

After this a comparison is made between filter sizes and different optimizers. This is applied to a three layer model as seen in figure 9.2. Even though there is not that much improvement possible regarding accuracy, an attempt can be made to minimize the models parameters, while also keep focussing on accuracy. Due to the large convolutional layers, this model model had a size of 2,7 MB. As explained in section 6.7 the model can be decreased in size by using bigger filters or pooling (both reducing input size in the next layer). A proof of that claim, which is already made in section 6.7, is given in graph 9.3. This graph represents the decrease in model size

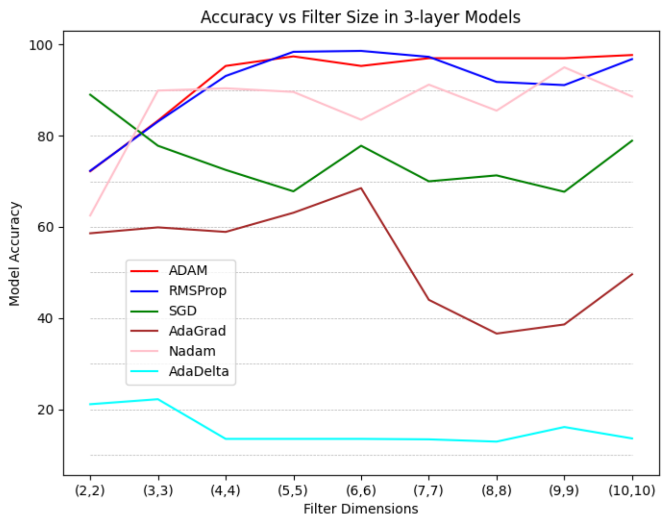


Figure 9.2: A graph displaying the accuracy as a function of filter size with different optimizers. Filter sizes are equal on all 3 layers of the model, layer sizes are 8, 16 and 32 respectively.

when increasing filter sizes of all its three layers. After varying these layers in pooling, size and filters, it is seen that model size decreased, while remaining highly accurate in some of the models. The smallest size model that still achieves an accuracy of over 98% is one of 203.4 kB. It achieves an accuracy of 98.2%. Its hyperparameters are shown here:

- Number of Epochs = 25
- Batch Size = 400
- Validation Split = 0.1
- Size of Conv. Layers = [8, 16]
- Pooling Layer = (1,0)
- Optimizer = Adam
- Size of filters = [(5,5)(6, 6)]
- Activation Function = ReLu

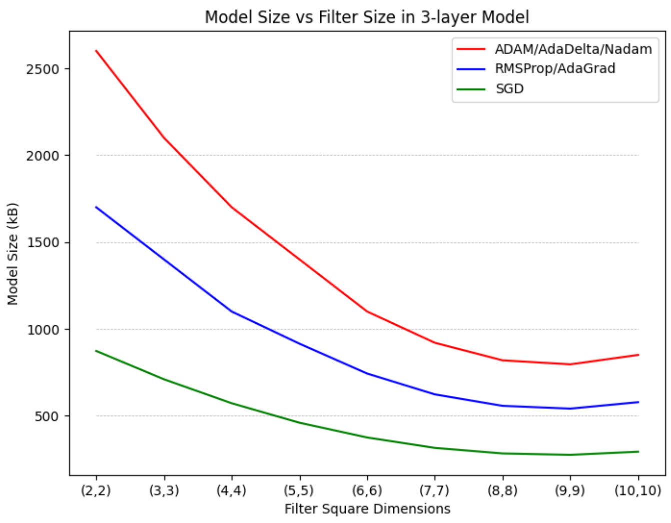


Figure 9.3: A graph displaying the decrease in model size/parameters, when increasing convolutional filter size for different optimizers. This is applied to a model with 3 layers of size 8,16 and 32. The gain in the end of the graph is the point where the trade of between increase in filter parameters and decrease in fully connected layer parameters flips.

As can be seen in these parameters, a pooling layer is applied after the first convolutional layer of size 8, followed by a convolutional layer of size 16. Its confusion matrix can be seen in table 9.7.

Summed results	Thumb Horizontal	Flat Hand	Four	Open Hand	Pinkindex	L	O	Index Vertical	Three	Accuracy
Thumb Horizontal	49	0	0	1	0	0	0	0	0	98%
Flat Hand	0	50	0	0	0	0	0	0	0	100%
Four	0	0	49	0	0	1	0	0	0	98%
Open Hand	0	0	0	50	0	0	0	0	0	100%
Pink-index	0	0	0	0	50	0	0	0	0	100%
L	0	0	0	1	0	49	0	0	0	98%
O	0	0	0	1	0	0	49	0	0	98%
Index Vertical	0	0	0	0	1	0	0	48	1	96%
Three	0	0	1	1	0	0	0	0	48	96%

Table 9.7: Confusion matrix of best 30 x 30 FLUKE images for 9 gestures. Accuracy = 98.2%.

The convoluted image, which is the output of a convolutional layer, is supposed to lay bear some features of the image. Namely the features it is trained to detect with the training data. Such an image is called a feature map. To give an idea of the appearance of such an image, a Python script is written for displaying one. A feature map of the first of the two convolutional layers from the model mentioned as the best FLUKE model is given in figure 9.4.

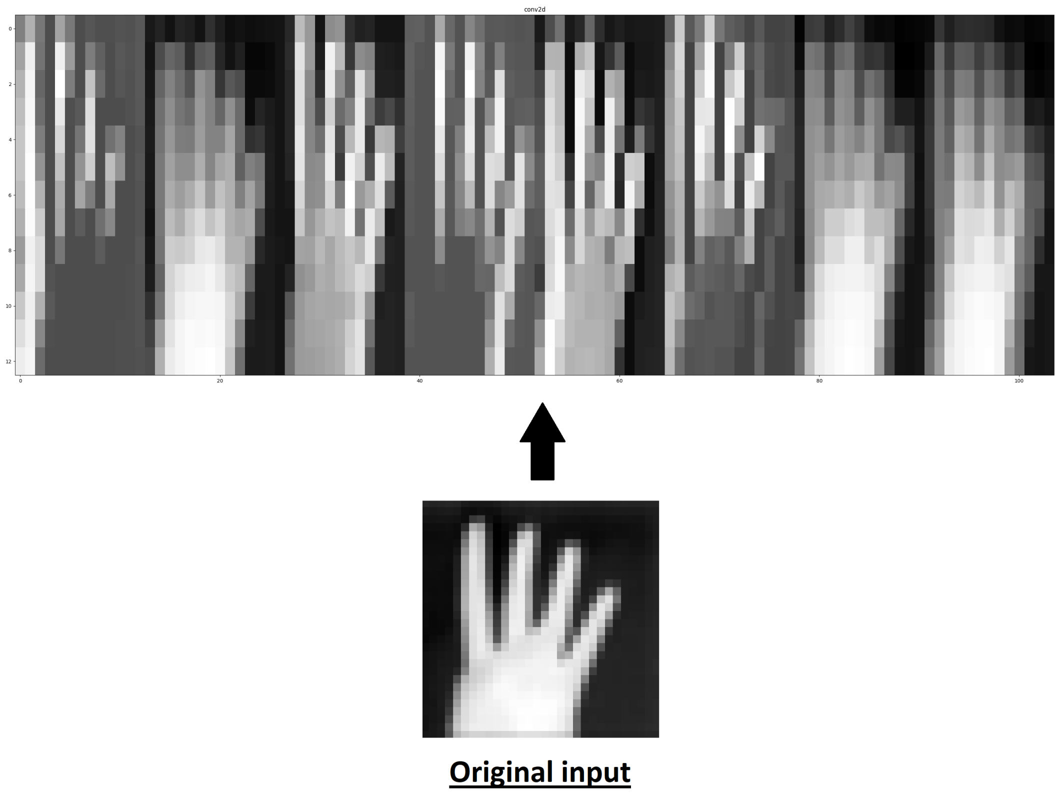


Figure 9.4: A feature map of the gesture shown as the original image, when going through a convolutional layer with a size of 8 neurons. The feature map shows that in some images, like on the most left, fingers are highlighted.

9.4 Varying the Environmental Conditions

Measurements with participants are all taken indoor in similar conditions. It is clear, however that environmental conditions can influence the passive IR images. In order to know in what quantity the images are distorted, both the FLUKE and Grid Eye are used to do measurements in 4 different conditions.

1. In front of a black background
2. In front of a white background
3. In direct sunlight
4. In front of a warm object The resulting pictures can be seen in

in figure 9.5 the results are shown as an 8 by 8 image from the Grid Eye and a 30 by 30 image from the FLUKE.

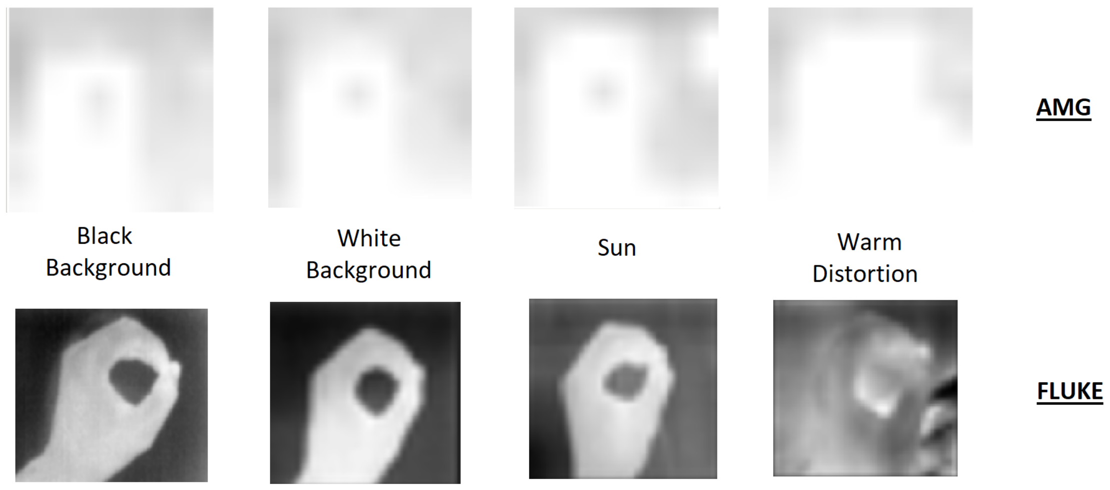


Figure 9.5: Images taken with the FLUKE and AMG in different conditions.

Chapter 10

Discussion

The thesis started with a number of research questions, which have all been addressed in the course of the report:

- What are the current techniques for gesture recognition and how accurate are these techniques? **Chapter 2**
- What sensors are available for recognition and can be implemented while holding for the criteria for power consumption, processing power, size and price? **Chapter 3**
- What method can be chosen to meet those criteria and how will it be implemented? **Chapter 4 to 8**
- How accurate is the chosen method and how does it compare to current techniques? **Chapter 9**

In chapter 2 it is concluded, that in order to distinguish more than 4 gestures, multiple sensing elements are required. This leaves only a few options open for use, of which two are the APDS-9500 and the AMG Grid Eye. The software of the APDS-9500 is not computationally intensive and has relative high accuracy on 4 of the 8 gestures (nearly 100%). The fifth gesture is rather unreliable (about 70%) and the advice would be to not use the APDS-9500 with this gesture. Also because the sensor requires an LED that is constantly lit, necessary for capturing moving gestures. This is a burden on the power supply of a small device.

The advantage of using image recognition in this application is that it only needs one LED flash (or in the case of passive IR none) to extract all the information needed for gesture classification. The Grid Eye data is not able to surpass the accuracy of the APDS-9500, even when adding the 5th and worst performing gesture to the APDS-9500 accuracy calculation. The experiments with the Grid Eye do prove that with a very low resolution raw image, recognition of gestures is possible up to 80 percent. And that with pre-processed images and extra information that is artificially generated (extrapolation) a model can improve.

When using low resolution data, the model does best when trained without convolutional layers. This is remarkable, because it neglects the layers that a neural network is all about. This can be because of two reasons, the first being, that convolution lowers image resolution even more and takes out more valuable information for the fully connected layer, than it adds by convoluting. The second is that the fully connected layer is usually fed with low resolution images. And that, specially in the case of the high contrast images of the Grid Eye, the input is quite similar to what it usually receives from higher resolution images, after a number of convolutions and poolings. With images from the FLUKE it is seen that convolutional layers become more valuable due to the increase in information/pixels given to the model.

A few points can be made in favour of the suggestion that the AMG has too few pixels to work with for this application.

1. Extrapolation on Grid Eye's data improves accuracy.
2. FLUKE data performs better with its higher resolution.
3. The Grid Eye performs better without convolution (either too few pixels for filtering or enough pixels for fully connected layer).

Neither of these suggestions would make a strong case when standing alone, because they could be refuted in other ways. The FLUKE 279 FC for example, could just have different qualities other then an increased resolution, that explains the difference in performance. However, because with all three points taken into account, especially that of extrapolation, there is enough proof that the 8 by 8 image from the Grid Eye is insufficient in resolution for this application.

As for the different hyperparameters: the batch size and dropout layer are added to prevent the model from converging into local minima. It seems that the batch size itself is already sufficient for achieving this. This can be seen by the insignificance regarding the accuracy when applying, or not applying a dropout layer in the best performing models. In chapter 9 it can be seen however, that the Grid Eye's models that came out on top for each database, had a dropout rate of 0.2 except for one model. This can indicate that the dropout rate gives a final nudge towards a better performance, but only when accuracy is not near 100% already, as is with the FLUKE's best model. The lowest possible batch size did differ between the Grid Eye and the FLUKE data. For the FLUKE the model still performs well with batch sizes up to 10, the Grid Eye however, already starts performing worse with batches lower than 300. This could be because less information makes it harder to converge to a good local minimum in the grid, especially when using less accurate estimates of that information.

With respect to the validation split, a smaller split seems to be better at 0.1. However, this is highly dependent on the database. Since the measurements require a lot of time, a limited database is built. Making training data very valuable. A validation split is nice to have, but it should take a minimum amount of images/data when dealing with small databases.

As for the FLUKE images, high accuarcy is achieved (98.2%) in combination with a model size (203 kB) that fits on a modern microcontroller. The FLUKE needs less training (reduced to about 20 epochs), this is most likely due to the increase in data given to the model. The data analyzed over an epoch with images that have 900 pixels is more valuable than one over images with 64 pixels. It should also be taken into account that the FLUKE is used with one test subject, while the Grid Eye is used on six. This means that there is a possibility of less variation in the FLUKE data, making it easier to find optimal solutions.

It is still questionable whether this can be brought into practice as gesture recognition mounted on earphones, specially regarding passive IR sensors. Passive IR sensors have one major source of noise, which is heat from other objects than the hand. This might be avoided if the range of the sensors is somehow kept to a minimum. If not, then active IR sensors can provide a solution. Due to the restricted time the LED has to light up for 1 image, it is very energy efficient. It would be necessary to create a database with these sorts of images, from for example the APDS-9500. If the contrast of the images is good enough, as shown with the improvement of the Grid Eye's processed data, then the images should train well. The FLUKE has already proven that a 30 by 30 pixel resolution is sufficient. So the result will probably be better with APDS-9500 even if its only due to the higher number of pixels. It should also be investigated whether there are other forms of noise for the passive IR sensor than heat, and what the most significant sources of noise are for an active IR sensor.

Chapter 11

Conclusion

Gesture recognition is used in order to create a user interface for earphones. IR sensors are very cheap, small and practical for a UI where no physical contact is required. As a reference to current gesture recognition the APDS-9500 is used. Eight participants are asked to perform 5 gestures multiple times, resulting in an accuracy of 92.25%.

As a newly proposed method, Python and the Keras library are implemented to create a neural network for image/gesture recognition. Python and Keras use a machine learning approach to create models with trained parameters. In order to do this a 3000 image gesture database consisting of 5 gestures from 6 participants is used for the training of a neural network. Each gesture containing 50 test images, 60 validation images and 490 training images. The goal is to train an image classification model for gestures. Variation is applied to the hyperparameters, which are the parameters used to train the models with. Training results in an accuracy of 92.36% for images from the AMG Grid Eye that are extrapolated and where contrast is increased. For raw images from the Grid Eye the accuracy is 79.2%. Which means that an increase of camera contrast and resolution increases the performance of the models. Even when that increase is is artificially generated from the same data as used prior to the pre-processing. The same gestures are tested with the same database size, but with with another passive IR camera: the FLUKE 279 FC. This results in an accuracy of 97.2%. When using higher resolution data from the FLUKE, it is noticed that less epochs and lower batch sizes are required. This is most likely because it is easier to find local minima when there is more information to recognize gestures with.

For downscaled 30 by 30 images of the FLUKE 279 FC, a database containing 9 gestures with each 600 images is used. This training results in a maximum accuracy of 99.3%. Because model accuracy is high enough priorities are shifted to minimizing model size. In attempts to keep model size to a minimum accuracy has become 98.2%, while the model size is 203 kB.

This study proves that a 30 x 30 image contains enough information for gesture recognition with 9 gestures, using a small neural network. This even outperforms the currently available APDS-9500 in accuracy and number of gestures. It also proved that in general, an increase in contrast or pixels, even when artificially created, can cause an increase in accuracy of neural networks. To such a degree that 5 different gestures in 8 by 8 pixel images are to be recognized with an accuracy of over 90%.

11.1 Future work

This study only used passive IR sensors for gesture recognition. Because this poses a risk in hot environments, it would be useful to investigate further option in the area of active IR sensors. Power consumption of active IR sensors will still be limited by the fact that one flash of the LED is needed to acquire all the information for one gesture. Which leads to the second necessity, namely that of when an image is triggered. The LED should not be flashing all the time, but only after triggered out of its sleep mode. This trigger could be a single physical button or a pattern measured by the sensor in a low power consuming state.

Next, it should also be tested whether the use of multiple subjects and thus different hands influences the accuracy of the model. Next to the fact that it influences the accuracy, it should also be investigated how a model performs on data from a subject that is not used for training, testing or validation. Since, in realistic situations, customers will most likely not have taken part in the data acquisition.

It could also prove useful to know where the drop in accuracy as a function of resolution becomes too steep. There is still a large difference between 8 by 8 images (64 pixels) and 30 by 30 image (900 pixels). The number of pixels plays a large role in the size of the model, if trying to keep the model size to a minimum, it is essential to keep image resolution to a minimum.

Lastly, the implementation should be tested in a microcontroller and preferably mounted onto something similar to an earphone. There are repositories available for converting models created in Python and Keras to C++ code. The network itself is a combination of different transformations and filters, that in essence needs no special libraries to run in C++. Eventually an image is the input to the microcontroller and the output will be the gesture recognized by the neural network.

Bibliography

[1] Y. Li, “Hand gesture recognition using kinect,” in *2012 IEEE International Conference on Computer Science and Automation Engineering*, 2012, pp. 196–199. DOI: 10.1109/ICSESS.2012.6269439.

[2] Z. Zivkovic, “Air gesture control using 5-pixel light sensor,” in *2014 IEEE International Conference on Consumer Electronics (ICCE)*, 2014, pp. 67–68. DOI: 10.1109/ICCE.2014.6775911.

[3] S. R. Das, T. E. Jakulin, and K. G. J. Nigel, “Linear and rotational air gesture detection using optical sensors setup in automotive infotainment system,” in *2016 International Conference on Communication and Signal Processing (ICCSP)*, 2016, pp. 1164–1169. DOI: 10.1109/ICCSP.2016.7754335.

[4] H. Cheng, A. M. Chen, A. Razdan, and E. Buller, “Contactless gesture recognition system using proximity sensors,” in *2011 IEEE International Conference on Consumer Electronics (ICCE)*, 2011, pp. 149–150. DOI: 10.1109/ICCE.2011.5722510.

[5] J. S. Kim, S. J. Yun, D. J. Seol, H. J. Park, and Y. S. Kim, “An ir proximity-based 3d motion gesture sensor for low-power portable applications,” *IEEE Sensors Journal*, vol. 15, no. 12, pp. 7009–7016, 2015. DOI: 10.1109/JSEN.2015.2471845.

[6] S. Electronics, *Vl6180*, Original datasheet of the VL6180, 2021. [Online]. Available: https://www.st.com/resource/en/datasheet/vl6180.pdf.

[7] AMS, *Tmd2635*, Original datasheet of the TMD2635, 2020. [Online]. Available: https://ams.com/documents/20143/36005/TMD2635_DS000674_4-00.pdf/3bff4671-799a-efb5-4c07-6bae3ec2bddb.

[8] V. Electronics, *Vcnl36826s*, Original datasheet of the VCNL36826S, 2020. [Online]. Available: `https://www.vishay.com/docs/84964/vcnl36826s.pdf`.

[9] S. Labs, *Si1120*, Original datasheet of the Si1120, 2010. [Online]. Available: `https://www.silabs.com/documents/public/data-sheets/Si1120.pdf`.

[10] S. Electronics, *Gp2ap054a00f*, Original datasheet of the GP2AP054A00F, 2021. [Online]. Available: `https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2ap054a00f_e.pdf`.

[11] Broadcom, *Apds-9960*, Original datasheet of the APDS-9960, 2021. [Online]. Available: `https://docs.broadcom.com/docs/AV02-4191EN`.

[12] ——, *Apds-9500*, Original datasheet of the APDS-9500, 2021. [Online]. Available: `https://docs.broadcom.com/docs/AV02-4584EN`.

[13] Panasonic, *Grid eye*, Original datasheet of the Grid Eye, 2021. [Online]. Available: `https://mediap.industry.panasonic.eu/assets/imported/industrial.panasonic.com/cdbs/www-data/pdf/ADI8000/ADI8000C66.pdf`.

[14] *Global earphones and headphones market size report, 2020-2027.* https://www.grandviewresearch.com/industry-analysis/earphone-and-headphone-market, 2020.

[15] *Broadcom website*, https://www.broadcom.com/, 2020.

[16] E. Søderholm, *Touchless human-computer interface for in-ear multipurpose wearable health device*, 2021.

[17] P. Wojtczuk, D. Binnie, A. Armitage, T. Chamberlain, and C. Giebeler, "A touchless passive infrared gesture sensor," in *Proceedings of the Adjunct Publication of the 26th Annual ACM Symposium on User Interface Software and Technology*, Association for Computing Machinery, 2013, pp. 67–68, ISBN: 9781450324069. DOI: `10.1145/2508468.2514713`. [Online]. Available: `https://doi.org/10.1145/2508468.2514713`.

[18] T. David Binnie, A. F. Armitage, and P. Wojtczuk, "A passive infrared gesture recognition system," in *2017 IEEE SENSORS*, 2017, pp. 1–3. DOI: `10.1109/ICSENS.2017.8234402`.

[19] L. Yu, H. Abuella, M. Z. Islam, J. F. O'Hara, C. Crick, and S. Ekin, "Gesture recognition using reflected visible and infrared lightwave signals," *IEEE Transactions on Human-Machine Systems*, vol. 51, no. 1, pp. 44–55, 2021. DOI: `10.1109/THMS.2020.3043302`.

[20] J. W. Lord, M. P. Rast, C. Mckinlay, J. Clyne, and P. D. Mininni, "Wavelet decomposition of forced turbulence: Applicability of the iterative donoho-johnstone threshold," *Physics of Fluids*, vol. 24, no. 2, p. 025 102, 2012. DOI: `10.1063/1.3683556`. [Online]. Available: `https://doi.org/10.1063/1.3683556`.

[21] C. Chronopoulos, "Quadrant: A multichannel, time-of-flight based hand tracking interface for computer music," *None*, 2019.

[22] G. A. Al, P. Estrela, and U. Martinez-Hernandez, "Towards an intuitive human-robot interaction based on hand gesture recognition and proximity sensors," in *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2020, pp. 330–335. DOI: `10.1109/MFI49285.2020.9235264`.

[23] P. Saini, N. Bamane, M. Bhanpurwala, and V. H. S. Vaishali, "Smart gesture recognition for disabled people," *SSRN*, 2020.

[24] S. Tateno, Y. Zhu, and F. Meng, "Hand gesture recognition system for in-car device control based on infrared array sensor," in *2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 2019, pp. 701–706. DOI: `10.23919/SICE.2019.8859832`.

[25] MATLAB, *version R2020a*. Natick, Massachusetts: The MathWorks Inc., 2020. [Online]. Available: `https://nl.mathworks.com/`.

[26] Arduino, *version 1.8.15*. BCMI, 2020. [Online]. Available: `https://www.arduino.cc/`.

[27] K. Çelebiler, "Heart disease diagnosis using neural networks on electrocardiogram datasets," Ph.D. dissertation, Jun. 2021. DOI: `10.13140/RG.2.2.27247.15521`.

[28] K. Website. "Online keras guide." (2021), [Online]. Available: `https://keras.io/guides/`.

Appendices

Appendix A

Abbreviations

ADAM:	Adaptive Moment Estimation
ALS:	Ambient Light Sensing
ANN:	Artificial Neural Network
CNN:	Convolutional Neural Network
DWT:	Discrete Wavelet Transform
FIFO:	First-in First-Out
GMM:	Gaussian Mixture Modeling
IR:	Infra-Red
I2C:	Inter Integrated Circuit
LED:	Light Emitting Diode
N.C.:	Not Connected
PCB:	Printed Circuit Board
SPI:	Serial Peripheral Interface
UI:	User Interface

Appendix B

Connect and Setup the Artemis Redboard

This is a step-wise instruction to connect the Artemis Redboard to the computer. The goal is to be able to record measurements for the database, so enough training data is available for the creation of a neural network. The Sparkfun website has a manual that indicates that this board can be set to Arduino in the Arduino application, this is not the case for neither Windows nor Linux on my PC. The uploading of code seems to fail, for unknown reasons, that is why I had to search for solutions and came to this, to save people time in the future I wrote a manual of how I did it. This is only for a Windows operating system!

1. Connect the Artemis Redboard to the pc with an USB-C to USB-A cable.
2. Download the Arduino application from arduino.cc, start it.
3. Go to file >> preferences
4. Go to additional board managers urls and paste the following: https://raw.githubusercontent.com/sparkfun/Arduino_Apollo3/master/package_sparkfun_apollo3_index.json
5. Go to tools >> manage libraries.

6. Search for Sparkfun, an option that you have just downloaded will pop up.
7. Click on install
8. Wait for the installation to finish, it might take a few minutes.
9. Go to tools >> board, hover over Sparkfun Apollo3, select Redboard Artemis.
10. Next, download the CH430 drivers, https://cdn.sparkfun.com/assets/learn_tutorials/8/4/4/CH341SER.EXE (If the link is outdated, you can google "download CH430 drivers").
11. Open the downloaded executable and click uninstall, this will avoid conflict with previous versions of the drivers.
12. Then click install, wait for the installation to finish.
13. Now go back to the Arduino app. Go to tools >> port, select the right port (COM3 for example), if not sure, you can unplug the device and see which one disappears.
14. Now you are set up to upload code into the Artemis Redboard, to verify you can use the blink example from the Arduino application itself.

Appendix C

Connect and Setup APDS-9500

This is the manual for setting up the APDS-9500 in gesture mode in combination with the Artemis Redboard. The Arduino application is used to upload the code to the Artemis Redboard and in order to do this you should have first executed the steps in appendix B to connect the Artemis Redboard to a Windows PC. If that is done, follow the next steps:

1. Connect jumper wires from Redboard to AMG sensor in the following manner:

1: INT	→	N.C.
2: VIO	→	N.C.
3: GND	→	GND
4: MCLK	→	N.C.
5: MOSI	→	N.C.
6: ICS	→	N.C.
7: SDA	→	SDA
8: LEDR2	→	N.C.
9: SCL	→	SCL
10: SCK	→	N.C.
11: LEDA	→	3.3 V
12: VDO	→	3.3 V

2. Open the Arduino application.
3. Run an I2C scan to see what the I2C address is of the sensor.
4. Open the APDS-9500 Arduino code, go to line 371 and change the I2C address to that given by the I2C scan.
5. Optional: Go through the defined settings, these are written to the sensor before starting the sensor readings. Change settings if preferred (e.g. what gestures should be measured).
6. Upload the code to the Artemis RedBoard
7. Open the serial monitor, if all runs correctly, it should start to display 0x20 as the response from the sensor, this means wake up was succesful.
8. Sensor readings can now be done and will be displayed in the serial monitor.

Appendix D

Connect and Setup AMG Grid-Eye

This is a setup guide for making use of the Grid Eye sensor by AMG in combination with the Artemis Redboard using the Arduino application. In order to do this you should have first executed the steps in appendix B to connect the Artemis Redboard to a Windows PC.

1. Take the software package "SparkFun_GridEYE_Arduino_Library-master" and place it inside you Arduino library folder. The original code is from the Sparkfun company, however, changes to the code are made to extract images from the sensor (chapter **??**).

2. Connect jumper wires from Redboard to AMG sensor in the following manner:

1: VDD	$\rightarrow$	3.3V
2: SDA	$\rightarrow$	SDA
3: VVP	$\rightarrow$	N.C.
4: SCL	$\rightarrow$	SCL
5: AVDD	$\rightarrow$	N.C.
6: INT	$\rightarrow$	N.C.
7: DVDD	$\rightarrow$	N.C.
8:AD-SEL	$\rightarrow$	GND
9: GND	$\rightarrow$	GND
10: GND	$\rightarrow$	N.C.

3. Open the Arduino application. Run an I2C scan to see what the I2C address is of the sensor.

4. Go to SparkFun_GridEYE_Arduino_Library-master/src/ and open the header file, check in line 41 if the address is the correct, if not, change to whatever the I2C scan reported.

5. Place the AMG Grid Eye C++ code in the Arduino folder. Go to the Arduino application and open the code (this is the customized code, so not available in the original Sparkfun kit)

6. Compile and upload the code to the redboard, you can now record.

7. Open the Matlab script, change the address where you want the images to be stored. Also change the range of the pictures you want to make, the names of the images stored in the specified folder are N to N_max (e.g. 1 to 100).

8. Optional: the script "process_bulk_1_0_extrapolate.m" is a method for pre-processing folders of data. A storage folder and folder to be processed can be specified. The images in the specified folder will be extrapolated and object pixels will become 1 and non-object pixels will become 0.

Appendix E

Setup the Fluke 279 FC

For setting up the FLIR and extracting the images from the device, take the following steps.

1. Download the Smartview software, open the executable and follow the steps for installation. Open the application when finished.
2. Turn on the FLUKE, set it in IR imaging mode by turning the switch all the way to the right.
3. Connect a USB-A cable to the PC and connect the micro-USB side of the cable to the FLUKE.
4. The Smartview application will recognize the device and ask for software updates if available. Wait for install, this will take about 5 minutes.
5. Click the yellow capture button on the top right for making images, then click again for saving the image to the memory (memory size is only 100 images)
6. Extract all images from the memory by clicking the "save all" button. "Save all and delete" if you want to wipe the memory of the Fluke also. Select the location to save to and press OK.
7. The files can now be opened by clicking file >> open, then selecting one or multiple images.

8. Go to edit >> all images, then uncheck the "display markers" box.

9. Go to file >> export all, then select the format for the image and the folder in which the images should be stored. Click OK and the files will be sent to the folder.

Appendix F

APDS-9500 gesture experiments

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	7	2	1
Down	0	0	0	10	0
Towards	0	0	4	0	6

Table F.1: APDS-9500 participant 1. Accuracy = 86%.

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	10	0	0
Down	0	0	0	10	0
Towards	0	2	1	0	7

Table F.2: APDS-9500 participant 2. Accuracy = 94%.

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	10	0	0
Down	0	0	0	10	0
Towards	0	4	1	0	5

Table F.3: APDS-9500 participant 3. Accuracy = 90%.

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	10	0	0
Down	0	0	0	10	0
Towards	0	1	3	1	5

Table F.4: APDS-9500 participant 4. Accuracy = 90%.

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	1	0	9	0	0
Down	0	0	0	10	0
Towards	0	0	3	0	7

Table F.5: APDS-9500 participant 5. Accuracy = 92%.

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	10	0	0
Down	0	0	0	10	0
Towards	0	0	2	0	8

Table F.6: APDS-9500 participant 6. Accuracy = 96%.

Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	10	0	0
Down	0	0	0	10	0
Towards	0	1	2	0	7

Table F.7: APDS-9500 participant 7. Accuracy = 94%.

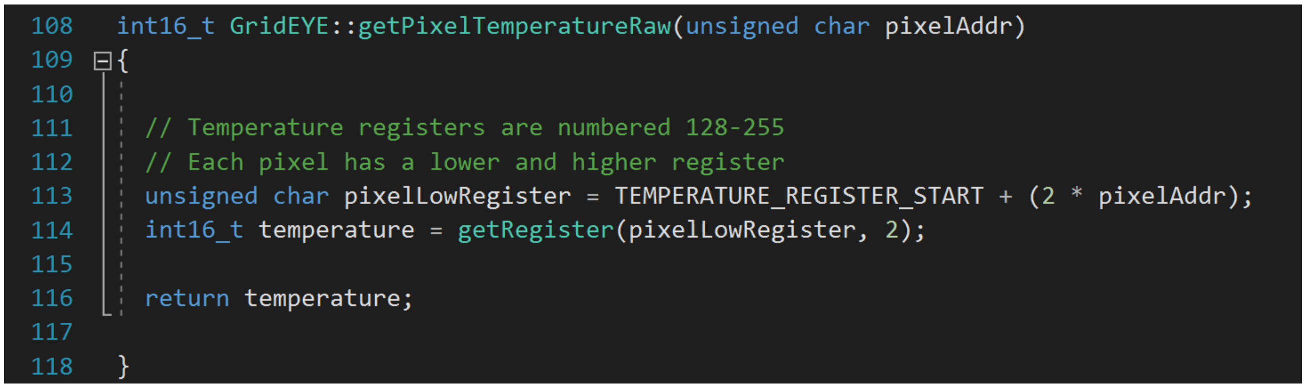
Summed results	Right	Left	Up	Down	Towards
Right	10	0	0	0	0
Left	0	10	0	0	0
Up	0	0	10	0	0
Down	0	0	0	10	0
Towards	0	1	0	1	8

Table F.8: APDS-9500 participant 8. Accuracy = 96%.

Appendix G

Important Parts of Sensor Software

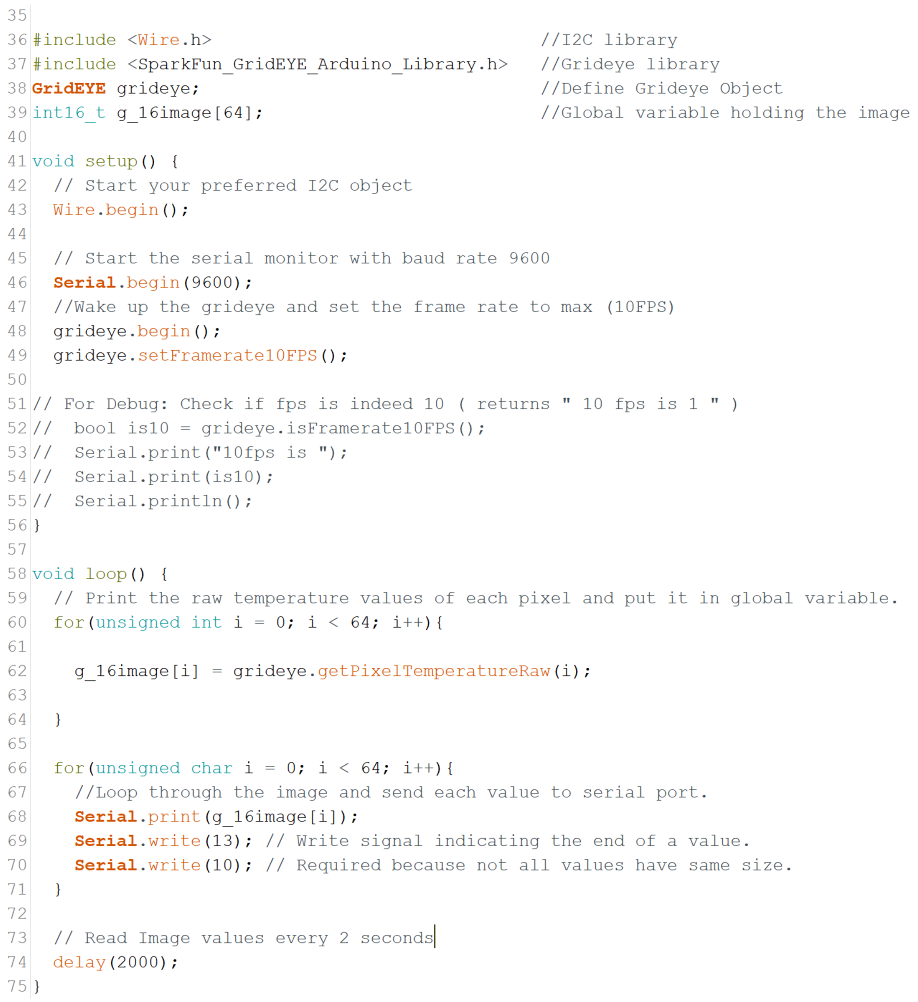
G.1 AMG Retrieve Pixel Values



```
int16_t GridEYE::getPixelTemperatureRaw(unsigned char pixelAddr)
{

    // Temperature registers are numbered 128-255
    // Each pixel has a lower and higher register
    unsigned char pixelLowRegister = TEMPERATURE_REGISTER_START + (2 * pixelAddr);
    int16_t temperature = getRegister(pixelLowRegister, 2);

    return temperature;

}
```

Figure G.1: Header function responsible for reading the pixel values in the sensor I2C registers.

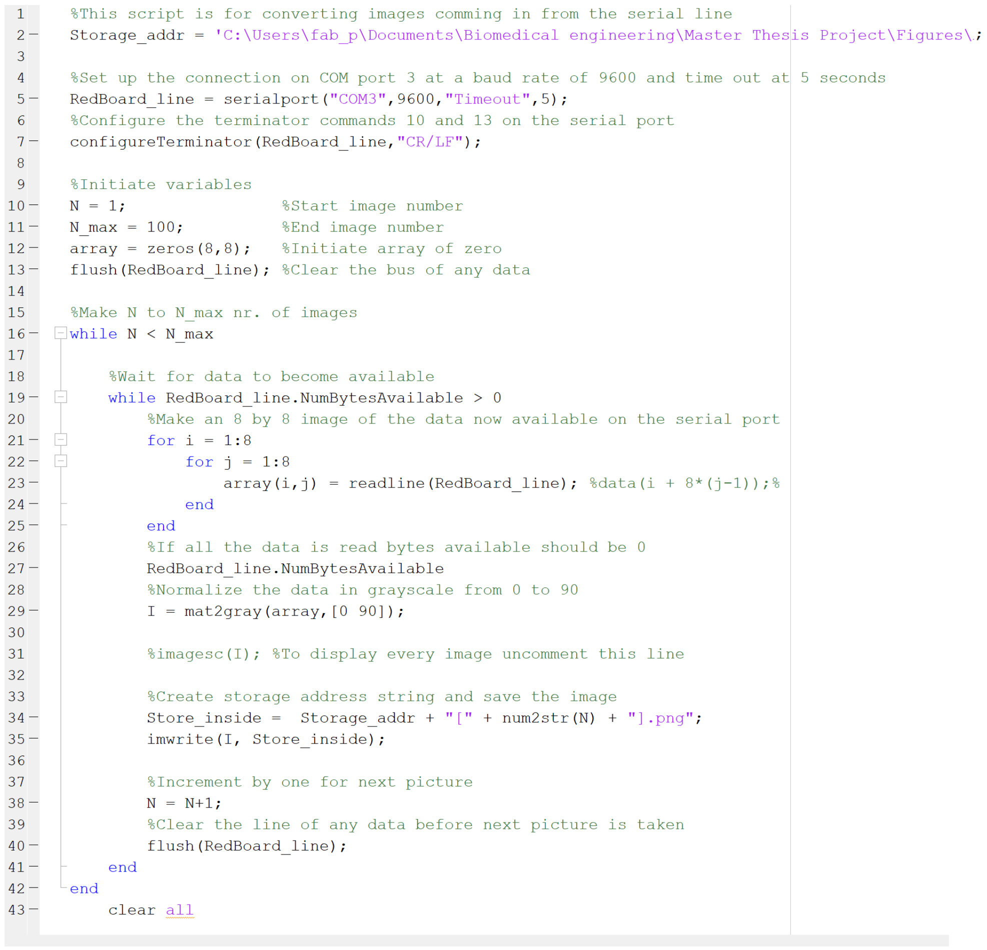
G.2 AMG C++ Code



```

#include <Wire.h>                                  //I2C library
#include <SparkFun_GridEYE_Arduino_Library.h>      //Grideye library
GridEYE grideye;                                   //Define Grideye Object
int16_t g_16image[64];                             //Global variable holding the image

void setup() {
  // Start your preferred I2C object
  Wire.begin();

  // Start the serial monitor with baud rate 9600
  Serial.begin(9600);
  //Wake up the grideye and set the frame rate to max (10FPS)
  grideye.begin();
  grideye.setFramerate10FPS();

// For Debug: Check if fps is indeed 10 ( returns " 10 fps is 1 " )
//  bool is10 = grideye.isFramerate10FPS();
//  Serial.print("10fps is ");
//  Serial.print(is10);
//  Serial.println();
}

void loop() {
  // Print the raw temperature values of each pixel and put it in global variable.
  for(unsigned int i = 0; i < 64; i++){

    g_16image[i] = grideye.getPixelTemperatureRaw(i);

  }

  for(unsigned char i = 0; i < 64; i++){
    //Loop through the image and send each value to serial port.
    Serial.print(g_16image[i]);
    Serial.write(13); // Write signal indicating the end of a value.
    Serial.write(10); // Required because not all values have same size.
  }

  // Read Image values every 2 seconds
  delay(2000);
}
```

Figure G.2: The program used for extracting images from the AMG displayed in the Arduino application.

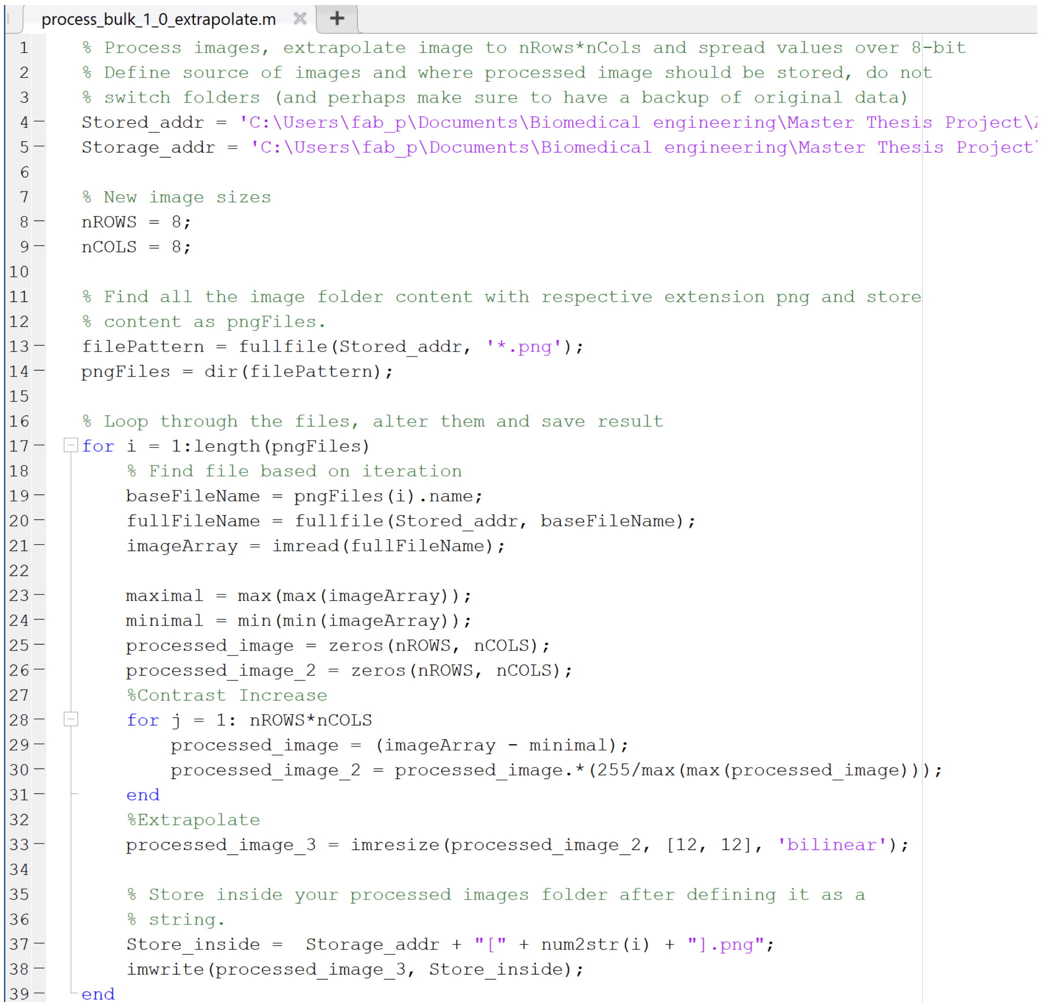
G.3 AMG Matlab Receive via Serial Line



```
%This script is for converting images comming in from the serial line
Storage_addr = 'C:\Users\fab_p\Documents\Biomedical engineering\Master Thesis Project\Figures\;

%Set up the connection on COM port 3 at a baud rate of 9600 and time out at 5 seconds
RedBoard_line = serialport("COM3",9600,"Timeout",5);
%Configure the terminator commands 10 and 13 on the serial port
configureTerminator(RedBoard_line,"CR/LF");

%Initiate variables
N = 1;                  %Start image number
N_max = 100;            %End image number
array = zeros(8,8);     %Initiate array of zero
flush(RedBoard_line);   %Clear the bus of any data

%Make N to N_max nr. of images
while N < N_max

    %Wait for data to become available
    while RedBoard_line.NumBytesAvailable > 0
        %Make an 8 by 8 image of the data now available on the serial port
        for i = 1:8
            for j = 1:8
                array(i,j) = readline(RedBoard_line); %data(i + 8*(j-1));%
            end
        end
        %If all the data is read bytes available should be 0
        RedBoard_line.NumBytesAvailable
        %Normalize the data in grayscale from 0 to 90
        I = mat2gray(array,[0 90]);

        %imagesc(I); %To display every image uncomment this line

        %Create storage address string and save the image
        Store_inside =  Storage_addr + "[" + num2str(N) + "].png";
        imwrite(I, Store_inside);

        %Increment by one for next picture
        N = N+1;
        %Clear the line of any data before next picture is taken
        flush(RedBoard_line);
    end
end
    clear all
```

Figure G.3: Matlab program responsible for receiving data sent down the serial line, converting it to an image and then saving it to the computer.

G.4 Pre-Processing Matlab



process_bulk_1_0_extrapolate.m

```
% Process images, extrapolate image to nRows*nCols and spread values over 8-bit
% Define source of images and where processed image should be stored, do not
% switch folders (and perhaps make sure to have a backup of original data)
Stored_addr = 'C:\Users\fab_p\Documents\Biomedical engineering\Master Thesis Project\
Storage_addr = 'C:\Users\fab_p\Documents\Biomedical engineering\Master Thesis Project'

% New image sizes
nROWS = 8;
nCOLS = 8;

% Find all the image folder content with respective extension png and store
% content as pngFiles.
filePattern = fullfile(Stored_addr, '*.png');
pngFiles = dir(filePattern);

% Loop through the files, alter them and save result
for i = 1:length(pngFiles)
    % Find file based on iteration
    baseFileName = pngFiles(i).name;
    fullFileName = fullfile(Stored_addr, baseFileName);
    imageArray = imread(fullFileName);

    maximal = max(max(imageArray));
    minimal = min(min(imageArray));
    processed_image = zeros(nROWS, nCOLS);
    processed_image_2 = zeros(nROWS, nCOLS);
    %Contrast Increase
    for j = 1: nROWS*nCOLS
        processed_image = (imageArray - minimal);
        processed_image_2 = processed_image.*(255/max(max(processed_image)));
    end
    %Extrapolate
    processed_image_3 = imresize(processed_image_2, [12, 12], 'bilinear');

    % Store inside your processed images folder after defining it as a
    % string.
    Store_inside =  Storage_addr + "[" + num2str(i) + "].png";
    imwrite(processed_image_3, Store_inside);
end
```

Figure G.4: Matlab script used for pre-processing all images present in a folder and storing them in folder of choice.

G.5 I2C Read, Write & Set

```
// Write data with I2C to the APDS sensor, used for giving instructions and setting values
void i2c_write_data(unsigned char I2C_ADDR, unsigned char reg, unsigned char value)
{
  Wire.beginTransmission(I2C_ADDR);       // Initiate communication with I2C
  Wire.write(reg);                        // Send write request to register
  Wire.write(value);                      // Send this value to register
  Wire.endTransmission();                 // End communication with I2C
}
```

Figure G.5: Function to write to an I2C register.

```
#define Im_INT                          0x82      // Set INT pin
#define Im_INT_Set                      0x0C
 i2c write data(0x73, Im INT, Im INT Set);
```

Figure G.6: Example of a call to I2C write function in hexadecimal form.

```
// I2C read block data
int i2c_read_data(int I2C_ADDR, int reg, int bytes_read, unsigned char *recv_data)
{
  int i = 0;

  if (!recv_data) return -1;
  Wire.beginTransmission(I2C_ADDR);           // Initiate communication with I2C
  Wire.write(reg);                            // Send read request to register
  Wire.endTransmission();                     // End the writing with I2C
  Wire.requestFrom(I2C_ADDR, bytes_read);     // Request a response from I2C address
  while (Wire.available())
  {
    if (i >= bytes_read) return i;            // While data is available collect bytes untill
    recv_data[i++] = Wire.read();             // nr. of bytes requested is met
  }
  return i;
}
```

Figure G.7: Function to read from an I2C register.

G.6 APDS Gesture Mode Main

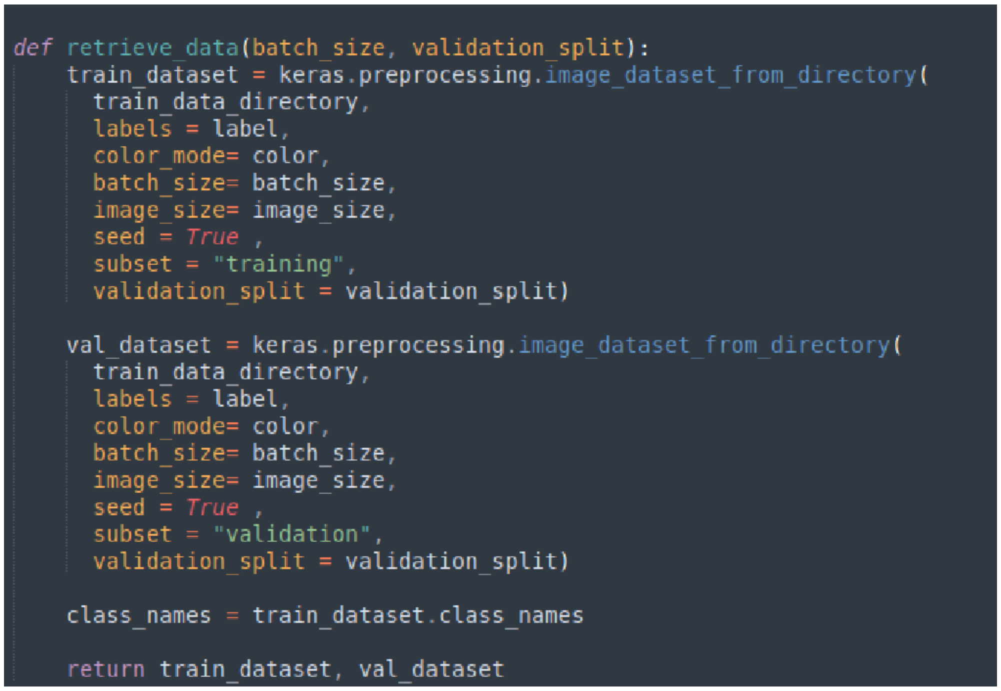
```
void loop() {
   // Loop Forever
  unsigned char I2C_ADDR = 0x73;      // I2C Address APDS
  // Read Gesture Result Registers
  i2c_read_data(I2C_ADDR, 0x43, 2, (unsigned char *)serial_data);
  g_gesture = ((serial_data[1]<<8)|(serial_data[0]&0xFF))&0x1FF;

  //Interpret the interupt status as a gesture
  if (g_gesture > 0)
  {
    if ((g_gesture&0x01) == 0x01) {
      if (orientation == ROT_0) {sprintf(dummyc,"DOWN");}
      else if (orientation == ROT_90) {sprintf(dummyc,"LEFT");}
      else if (orientation == ROT_180) {sprintf(dummyc,"UP");}
      else {sprintf(dummyc,"RIGHT");} }
    else if ((g_gesture&0x02) == 0x02) {
      if (orientation == ROT_0) {sprintf(dummyc,"UP");}
      else if (orientation == ROT_90) {sprintf(dummyc,"RIGHT");}
      else if (orientation == ROT_180) {sprintf(dummyc,"DOWN");}
      else {sprintf(dummyc,"LEFT");} }
    else if ((g_gesture&0x04) == 0x04) {
      if (orientation == ROT_0) {sprintf(dummyc,"RIGHT");}
      else if (orientation == ROT_90) {sprintf(dummyc,"DOWN");}
      else if (orientation == ROT_180) {sprintf(dummyc,"LEFT");}
      else {sprintf(dummyc,"UP");} }
    else if ((g_gesture&0x08) == 0x08) {
      if (orientation == ROT_0) {sprintf(dummyc,"LEFT");}
      else if (orientation == ROT_90) {sprintf(dummyc,"UP");}
      else if (orientation == ROT_180) {sprintf(dummyc,"RIGHT");}
      else {sprintf(dummyc,"DOWN");} }
    else if ((g_gesture&0x10) == 0x10) {sprintf(dummyc,"FORWARD");}
    else if ((g_gesture&0x20) == 0x20) {sprintf(dummyc,"BACKWARD");}
    else if ((g_gesture&0x40) == 0x40) {sprintf(dummyc,"CLOCKWISE");}
    else if ((g_gesture&0x80) == 0x80) {sprintf(dummyc,"COUNTER-CW");}
    else if ((g_gesture&0x100) == 0x100) {sprintf(dummyc,"WAVE");}
    else {sprintf(dummyc,"");}

    Serial.println(dummyc);
  }
}
```

Figure G.8: Main function for the APDS-9500 that reads the interrupt register, interprets the result and then prints it.

Appendix H

Important Parts of Machine Learning Software

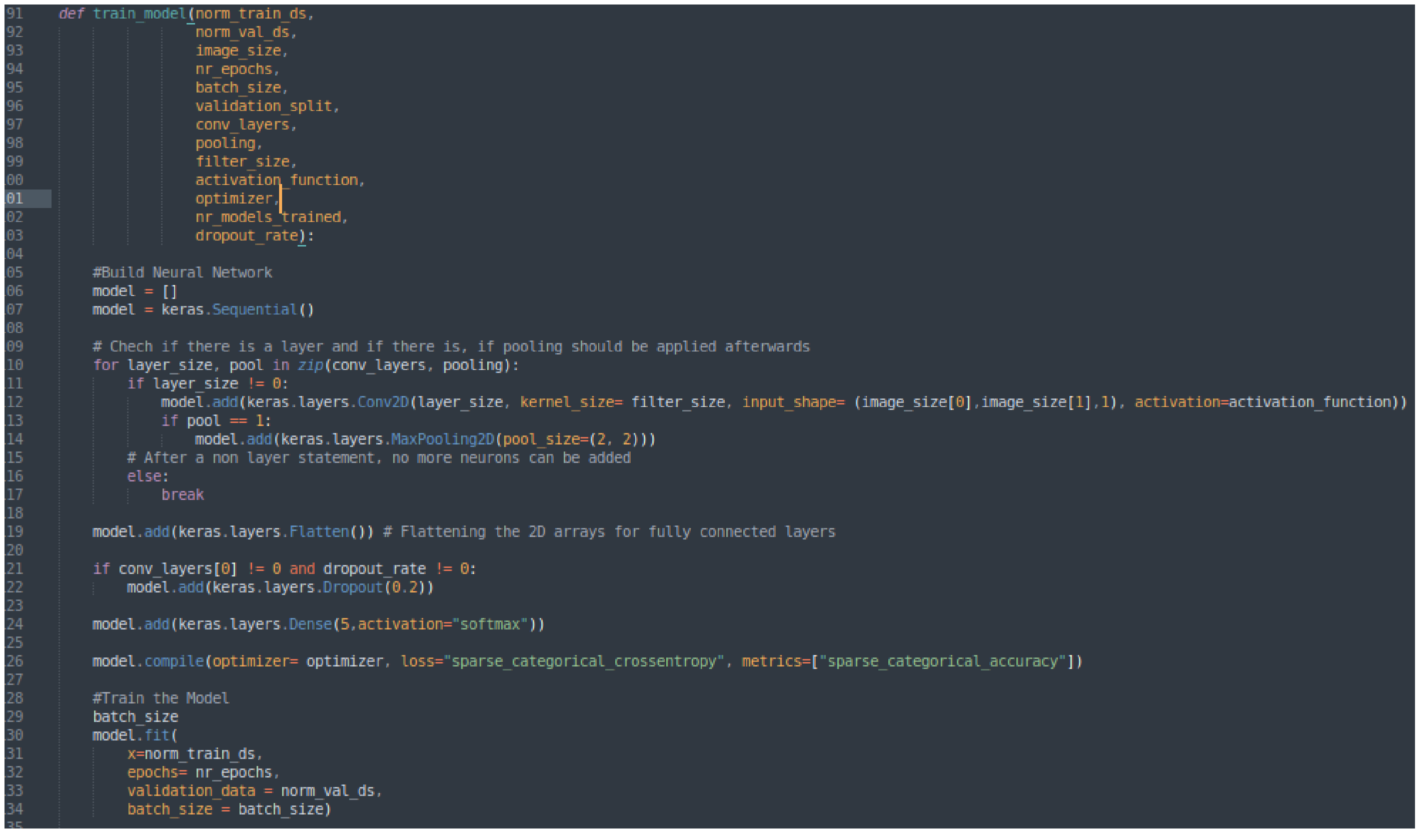
H.1 Retrieve Data



```
def retrieve_data(batch_size, validation_split):
    train_dataset = keras.preprocessing.image_dataset_from_directory(
      train_data_directory,
      labels = label,
      color_mode= color,
      batch_size= batch_size,
      image_size= image_size,
      seed = True ,
      subset = "training",
      validation_split = validation_split)

    val_dataset = keras.preprocessing.image_dataset_from_directory(
      train_data_directory,
      labels = label,
      color_mode= color,
      batch_size= batch_size,
      image_size= image_size,
      seed = True ,
      subset = "validation",
      validation_split = validation_split)

    class_names = train_dataset.class_names

    return train_dataset, val_dataset
```

Figure H.1: The code used to load in a database into the Python training program.

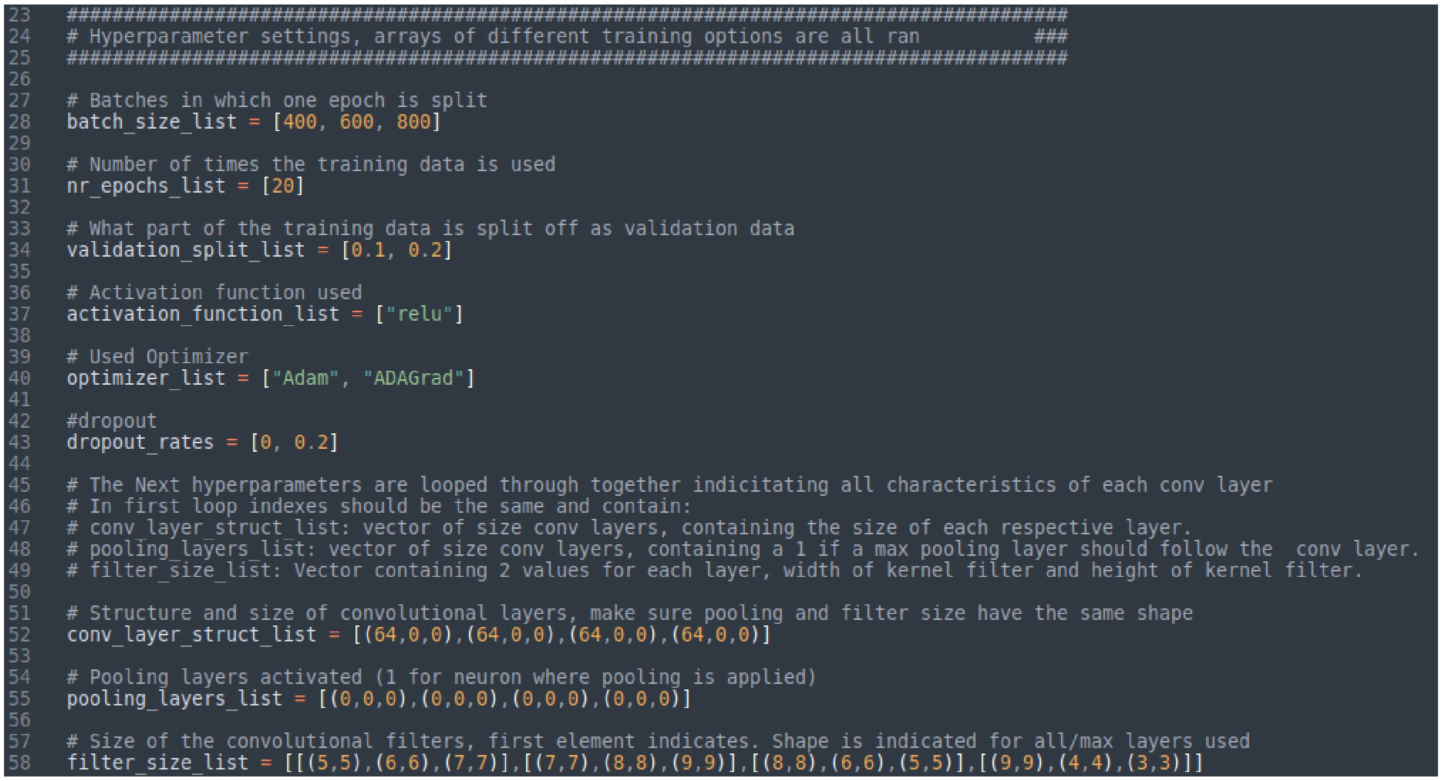
H.2 Train Function



```python
def train_model(norm_train_ds,
                norm_val_ds,
                image_size,
                nr_epochs,
                batch_size,
                validation_split,
                conv_layers,
                pooling,
                filter_size,
                activation_function,
                optimizer,
                nr_models_trained,
                dropout_rate):

    #Build Neural Network
    model = []
    model = keras.Sequential()

    # Chech if there is a layer and if there is, if pooling should be applied afterwards
    for layer_size, pool in zip(conv_layers, pooling):
        if layer_size != 0:
            model.add(keras.layers.Conv2D(layer_size, kernel_size= filter_size, input_shape= (image_size[0],image_size[1],1), activation=activation_function))
            if pool == 1:
                model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
        # After a non layer statement, no more neurons can be added
        else:
            break

    model.add(keras.layers.Flatten()) # Flattening the 2D arrays for fully connected layers

    if conv_layers[0] != 0 and dropout_rate != 0:
        model.add(keras.layers.Dropout(0.2))

    model.add(keras.layers.Dense(5,activation="softmax"))

    model.compile(optimizer= optimizer, loss="sparse_categorical_crossentropy", metrics=["sparse_categorical_accuracy"])

    #Train the Model
    batch_size
    model.fit(
        x=norm_train_ds,
        epochs= nr_epochs,
        validation_data = norm_val_ds,
        batch_size = batch_size)
```

Figure H.2: Part of the function that is used for training the models with the parameters as listed in figure H.3.

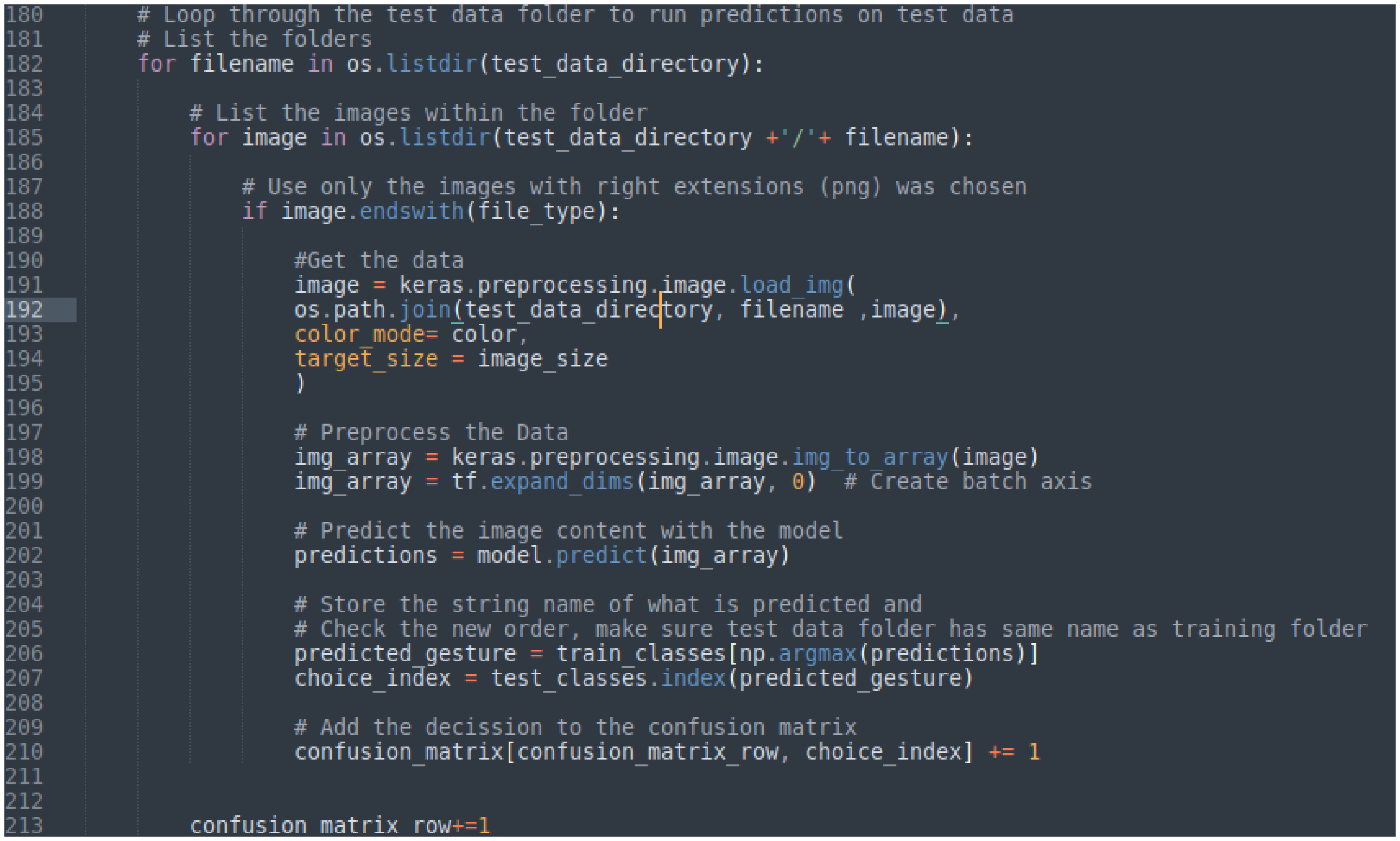
H.3 Hyperparameter Script



```
#####################################################################################
# Hyperparameter settings, arrays of different training options are all ran        ###
#####################################################################################

# Batches in which one epoch is split
batch_size_list = [400, 600, 800]

# Number of times the training data is used
nr_epochs_list = [20]

# What part of the training data is split off as validation data
validation_split_list = [0.1, 0.2]

# Activation function used
activation_function_list = ["relu"]

# Used Optimizer
optimizer_list = ["Adam", "ADAGrad"]

#dropout
dropout_rates = [0, 0.2]

# The Next hyperparameters are looped through together indicitating all characteristics of each conv layer
# In first loop indexes should be the same and contain:
# conv_layer_struct_list: vector of size conv layers, containing the size of each respective layer.
# pooling_layers_list: vector of size conv layers, containing a 1 if a max pooling layer should follow the  conv layer.
# filter_size_list: Vector containing 2 values for each layer, width of kernel filter and height of kernel filter.

# Structure and size of convolutional layers, make sure pooling and filter size have the same shape
conv_layer_struct_list = [(64,0,0),(64,0,0),(64,0,0),(64,0,0)]

# Pooling layers activated (1 for neuron where pooling is applied)
pooling_layers_list = [(0,0,0),(0,0,0),(0,0,0),(0,0,0)]

# Size of the convolutional filters, first element indicates. Shape is indicated for all/max layers used
filter_size_list = [[(5,5),(6,6),(7,7)],[(7,7),(8,8),(9,9)],[(8,8),(6,6),(5,5)],[(9,9),(4,4),(3,3)]]
```

Figure H.3: The code indicating different parameters to train with, in this example 96 models will be created.

H.4 Confusion Matrix and Accuracy Calculating Code



```
    # Loop through the test data folder to run predictions on test data
    # List the folders
    for filename in os.listdir(test_data_directory):

        # List the images within the folder
        for image in os.listdir(test_data_directory +'/'+ filename):

            # Use only the images with right extensions (png) was chosen
            if image.endswith(file_type):

                #Get the data
                image = keras.preprocessing.image.load_img(
                os.path.join(test_data_directory, filename ,image),
                color_mode= color,
                target_size = image_size
                )

                # Preprocess the Data
                img_array = keras.preprocessing.image.img_to_array(image)
                img_array = tf.expand_dims(img_array, 0)  # Create batch axis

                # Predict the image content with the model
                predictions = model.predict(img_array)

                # Store the string name of what is predicted and
                # Check the new order, make sure test data folder has same name as training folder
                predicted_gesture = train_classes[np.argmax(predictions)]
                choice_index = test_classes.index(predicted_gesture)

                # Add the decission to the confusion matrix
                confusion_matrix[confusion_matrix_row, choice_index] += 1


        confusion_matrix_row+=1
```

Figure H.4: Code responsible for classifying all the gestures in the test database folder and creating the confusion matrix.