Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Clap detection with Microcontroller

Cyril Arnould,

Nicolas de Palézieux dit Falconnet,

Dennis Meier,

Daniel Wälchli

GA-2013-03

Institut für Technische Informatik
und Kommunkationsnetze

Supervisors:

Dr. Beat Pfister and Tofigh Naghibi

June 12, 2013

Abstract

This paper documents our group work on clap detection, covering the goals we set and how we achieved them. We first briefly describe our lab setup, then switch straight to explaining the resulting algorithms, their detection error tradeoff curves and go over their hardware implementation. In the final part, we lay bare the trains of thought we had on our algorithms, their advantages and possible problems. We list the struggles we encountered and the lessons learned.

Table of contents

1 Introduction

Clap detectors have been around for quite some time to switch lights on and off and probably for other gimmicks. On the internet there are different suggestions on how to detect claps and even complete circuit diagrams for analog, digital and hybrid clap detectors (e.g. [5], [6] and [7]). We wanted to see if we could implement our own clap detector using the AMIV Bastli AVR-board [2], which would require us to deal with both hardware and software aspects of the problem. We did, however, lay a special emphasis on the software and algorithmic aspects.

1.1 Problem Description

The problem is to find a simple, robust method to detect single claps in sound and implement that method on a simple microcontroller. More specifically, the problem can be split into three parts:

- **Develop a suitable clap detection algorithm** by creating a collection of example-recordings with and without claps in different environments and exploiting the found features of claps.
- **Test the performance** of the algorithm by creating a DET (detection error tradeoff) curve [1] and then modify and improve the algorithm using the gained knowledge.
- **Simplify and implement** the algorithm on a simple microcontroller. The reduction in reliability due to the simplification process must be determined and represented in another DET curve.

The original (german) problem description for the project can be found in appendix B.

1.2 Goals

Our goal was to have a working clap detector on a microcontroller with a reasonable error rate (both missed detections and false-positives) at the end of the project while fulfilling the requirements in the problem description. We also wanted to try different algorithms to be able to compare them.

2 Materials & methods

2.1 Procedure

To reach our goals and fulfill the problem description requirements, the following steps needed to be taken:

- Buy and assemble the hardware.
- Do research on clap detection.
- Create a collection of example signals using the hardware.
- Develop algorithms to detect claps using Matlab.
- Rewrite the algorithms in C for the microcontroller.
- Test the algorithms in practice and improve them with the gained knowledge.
- Write the project report.
- Create an introductory and a final presentation.

2.2 Organisation

Since our team had four members with different workload requirements and Daniel was abroad during a large part of the project, we had to organize ourselves accordingly:

Member	Cyril	Daniel	Dennis	Nicolas
Workload	33%	17%	17%	33%
Respons-ibilities	Hardware (HW) assembly Putting lab HW setup into operation HW & SW debugging Create & manage example signals database C algorithm implementation	Internet Research Algorithm design Matlab algorithm implementation Final presentation	Proj.-Coordination Lab HW setup SW debugging Final report finish	Putting lab HW setup into operation Matlab algorithm implementation Matlab test-bench implementation Matlab DET-curve and parameter optimization scripts

We also split the work on the final report in a similar fashion:

Beginning	Abstract		Introduction Materials & Methods	
Results	Rejected Algorithms	Algorithm 2 Algorithm 3 Figures Proofreading		Algorithm 1 Algorithm 3 Rejected Algorithms DET Curves Hardware Implementation
Discussion	General Concerns The Algorithms Hardware limitations		A slow start	The Algorithms

2.3 The lab setup

The lab hardware setup is depicted in figure 1. Its centerpiece is the AMIV Bastli **AVR-Board** (corresponding files can be found at [2] and its circuit diagram in Appendix A) with an ATmega32 microcontroller as its main component (for the datasheet see [4]). The microcontroller gets programmed in C from the Linux **computer** via the usbasp [3] **programmer**, it can transmit data to the PC over a serial connection using its UART and receive the **microphone's** [9] signal on one of its A/D-converter pins. On the computer, we used Matlab to process the received information.

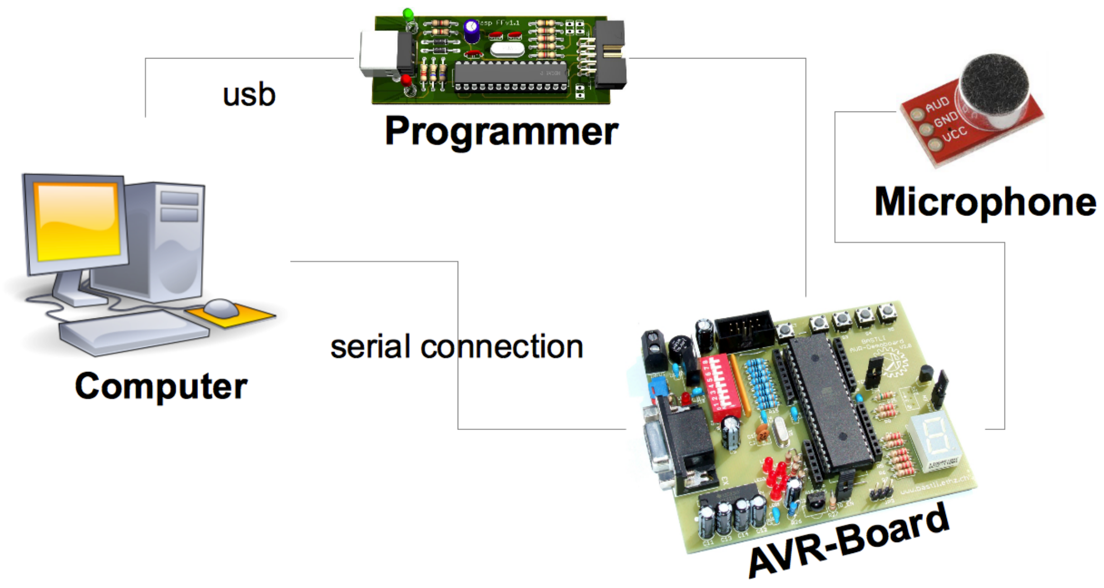


Fig. 1: Lab hardware setup

Example recording database

In order to test our algorithms in Matlab, we needed an extensive database of sample recordings. To make sure these recordings would contain data as similar as possible to the one the microcontroller would be receiving in operation, we recorded data via the microcontroller and the electret microphone. The sound input was read and saved in one Matlab script, which captured user-generated metadata as well.

We decided that we want an easy way of displaying as much information as possible contained in the database. We also needed a function to load the recordings into Matlab. We designed this function to differentiate between recordings containing claps, background noise and sounds that were imitating claps.

Parameter optimization

It quickly became apparent that we would need several Matlab scripts to efficiently test a wide range of parameters. Initially we were planning on simply having the three algorithms and one script which runs all the tests, but ended up with three different implementations, testbenches and super-testbenches for each algorithm.

The testbenches are functions which run the algorithm for each combination of parameters and signals they receive as an input and return an array containing information on how each parameter combination performed on the different signals. The output contained information about parameter combinations that lead to too many detections, too few detections, detections at wrong locations and of course correct detections.

The super-testbenches were scripts that ran the testbenches and then evaluated the information they received from them. It then output to the console which parameter combinations led to the least false positives, which lead to the most successes and which led to the least missed detections. That way we were able to determine the optimized parameters discussed find in the results section.

3 Results

3.1 Algorithms

The complexity of the detection algorithm was very limited by our hardware's processing capabilities. An obvious way to analyze an audio signal is to calculate its discrete fourier transform [10]. In the case of clap detection the fourier transform could be used to look for sudden increases in energy across a wide frequency spectrum. But due to the limitations of the hardware this kind of analysis was not an option for us, therefore our algorithms are limited to analyses in the time domain. The algorithms which were developed, including some which were rejected early on, are presented below.

Algorithm 1: A simple threshold decision rule

Our initial and most simple detection algorithm simply compares the current noise level to a threshold value. If the current amplitude is above this threshold a clap event is registered.

During a clap there are many high sample values, but there are also several samples which are close to zero. In order to prevent this from resulting in many events being detected at every high sample value, a short term average (consisting of the previous m samples, instead of the current sample alone) was compared to the threshold, effectively low pass filtering the signal.

The threshold which has to be surpassed by the short term average depends on the noise floor and is adapted continuously. A long term average is approximated by averaging the previous n samples, where n is a number significantly larger than the number of samples over which the short term average is taken (n >> m). The threshold which has to be surpassed for a clap event to be registered is a multiple of this long term average.

We have found the lengths of the short and long term windows as $m = 20$ and $n = 400$ samples together with a *decision_threshold* = 20 to be quite optimal for our hardware setup.

```
for: every sample in the signal
     short_term_average = mean of last m samples;
     long_term_average = mean of last n samples;
     clap_likeness = short_term_average/long_term_average
     if(clap_likeness > decision_threshold)
               if(not during_clap)
                      during_clap = 1;
                      signal clap event;
               end
     else
               during_clap = 0;
     end
end
```

Pseudocode for algorithm 1

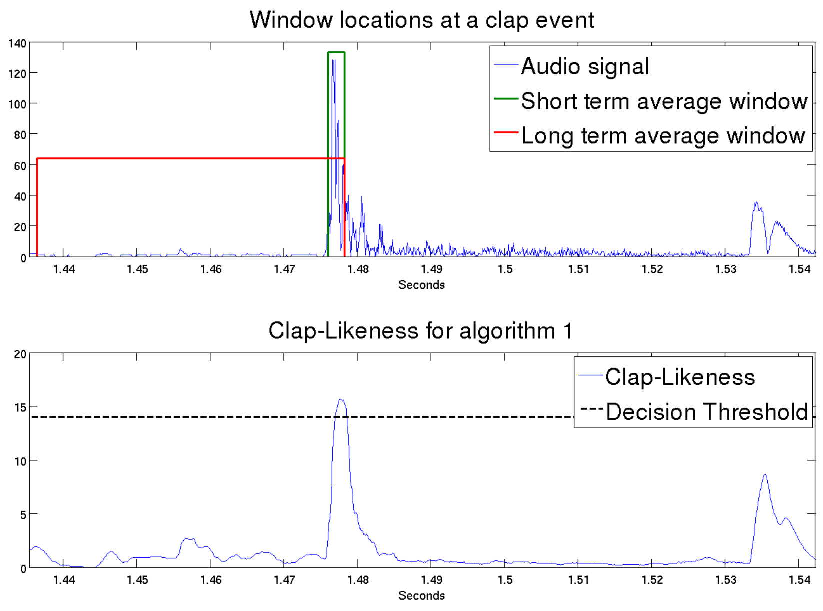


Fig: 2: Diagram showing how algorithm 1 works. The short term average (average of all samples inside the red box) divided by the long term average (average of all samples inside the green box) is the clap-likeness value. If it is higher than the decision threshold a clap event is determined.

Algorithm 2: The relation of the amplitude to the clap duration

A more complex algorithm is our second one. Like algorithm 1 it low pass filters the signal by computing a short term average. This short term average has to exceed a threshold in order to trigger further evaluation of the microphone input. The evaluation includes measuring the duration of how long the constant threshold is exceeded by the low pass filtered signal. Then it analyzes the ratio of the square of the maximum amplitude to the duration. The aim of this process is to obtain better results since we are looking for high amplitudes during a short duration (single peaks). In addition to that, the algorithm strictly ignores high peaks which last longer than a given duration, because a clap is in no way a long-lasting sound.

Optimized parameters for algorithm 2 on our hardware have been found to be: *threshold* = 20 , *decision_threshold* = 40, *short_term_duration* = 20 samples and *max_allowed_clap_duration* = 300 samples.

```
for: every sample in the signal
      short_term_average = mean of last 20 samples;
      while( short_term_average > threshold )
            max_val = max(short_term_average - threshold);
            clap_duration++;
      end
      if( clap_duration > max_allowed_clap_duration)
            clap_duration = 0;
            break;

      if( maximum duration above threshold not exceeded )
           clap_likeness = max_val^2/clap_duration;
           if(clap_likeness > decision_threshold)
                  signal clap event;
           end
      end
end
```

Pseudocode for algorithm 2.

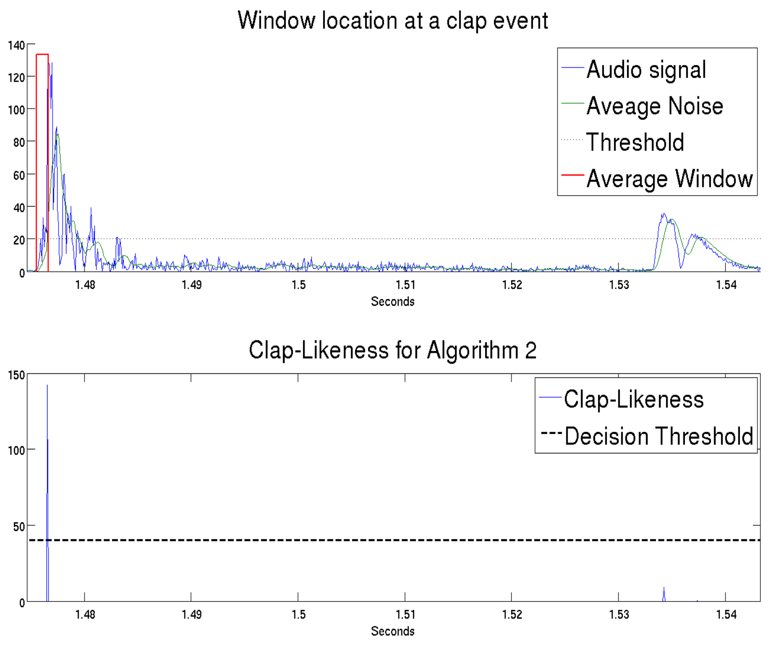


Fig. 3: Diagram showing how algorithm 2 works.

Algorithm 3: Amplitude to clap duration ratio with an adaptive threshold

The principle of algorithm 3 is the same as the one of algorithm 2 but it adds a final refinement. In algorithm 3 we do not use a constant noise threshold which has to be exceeded to run the whole process. This threshold is the sum of the short term average and a given addend. In practice this means algorithm 3 takes the general noise floor into account when looking for peaks.

Optimized parameters for algorithm 3 on our hardware have been found to be: *threshold_constant* = 19, *decision_threshold* =3, *short_term_duration* = 15 samples and *long_term_duration* = 500 samples.

```
for: every sample in the signal
      short_term_average = mean of last short_term_duration samples;
      long_term_average = mean of last longt_term_duration samples;
      threshold = threshold_constant + long_term_average;
      while( short_term_average > threshold )
            max_val = max(short_term_average - threshold) since above threshold;
            clap_duration++;
            if( clap_duration > max_allowed_clap_duration )
                  clap_duration = 0;
                  max_val = 0;
                  break;
      end
      if( maximum duration above threshold not exceeded )
           clap_likeness = max_val^2/clap_duration;
           if( clap_likeness > decision_threshold )
                  signal clap event;
           end
      end
end
```

Pseudocode for algorithm 3.

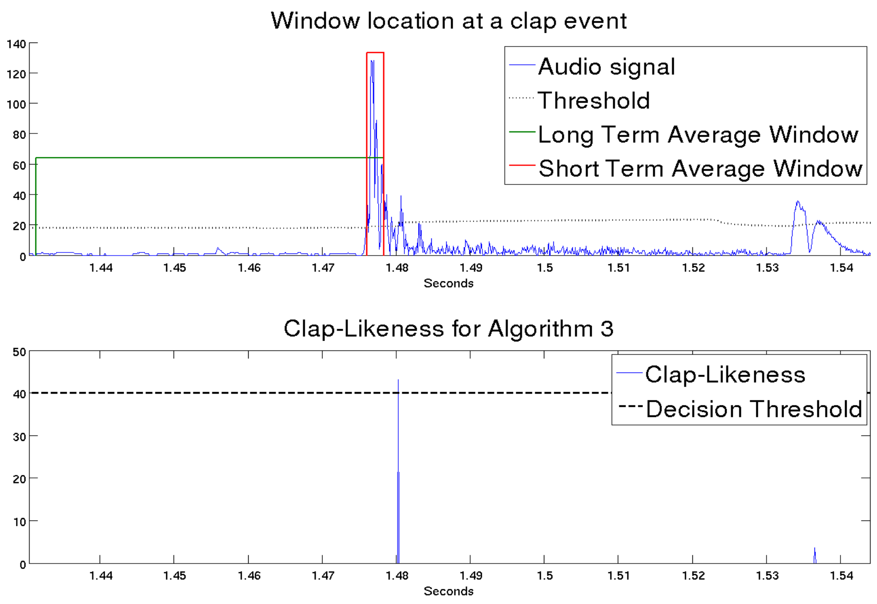


Fig. 4: Diagram showing how algorithm 3 works.

Rejected Algorithms

Measuring the zero crossing rate

When we were researching different ways of detecting claps, we came upon the information that the zero crossing rate (ZCR) of the audio signal would increase during a clap [8]. However, in our experiments the zero crossing rate was the same or even higher when a person is talking. It is possible that we simply weren't able to sample the signal fast enough to see an increase in the ZCR during a clap. Either way, we abandoned this idea shortly after that.

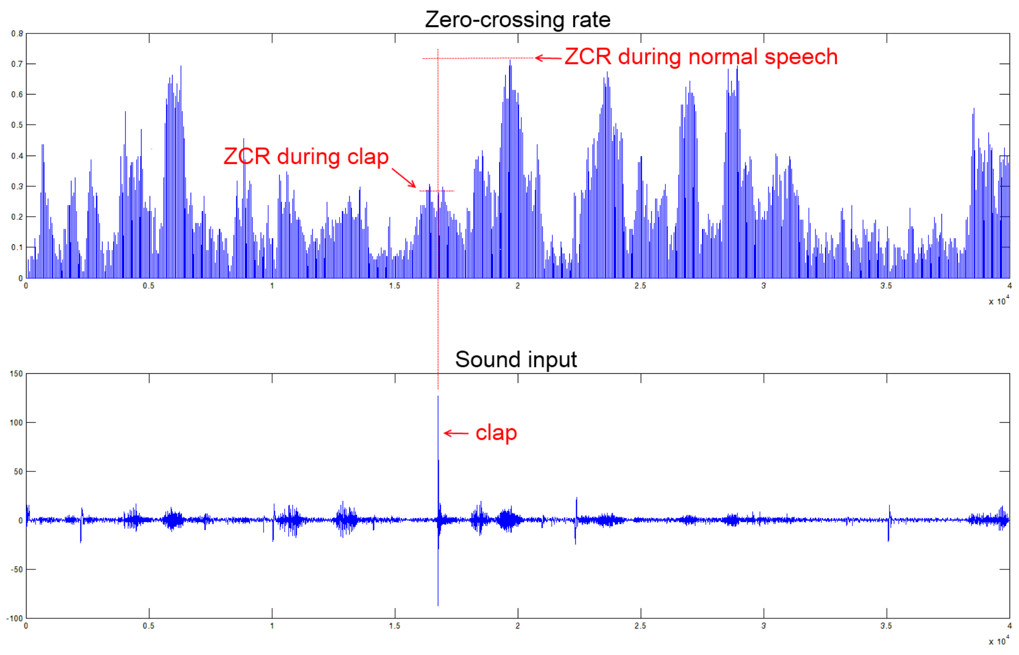


Fig. 5: Zero crossing rate of a recording containing claps and speech.

Comparison to an exponentially decaying envelope

Another characteristics of a clap recording is the approximately exponential decay of the amplitude as can be observed in figure 6. We therefore decided to design an algorithm which, after the amplitude has increased over a certain threshold, compares its decay to an exponential. The difference of the actual amplitude recorded by the microphone and the exponential envelope is summed up over time. If this error is small enough a clap event is registered. After implementing this algorithm and trying it out with several of our sample recordings it was decided that an exponentially decaying envelope was not characteristic enough of a clap. The speed at which a clap decays as well as the shape of its envelope largely depends on the surroundings of the microphone. If the microphone is placed in a room where there is a lot of echo, the amplitude of the clap decays much slower than in an acoustically dry room. Thus a detection algorithm based on the decay rate would have to be calibrated to the room in which the device is to be used, which is not something we want our device to require.

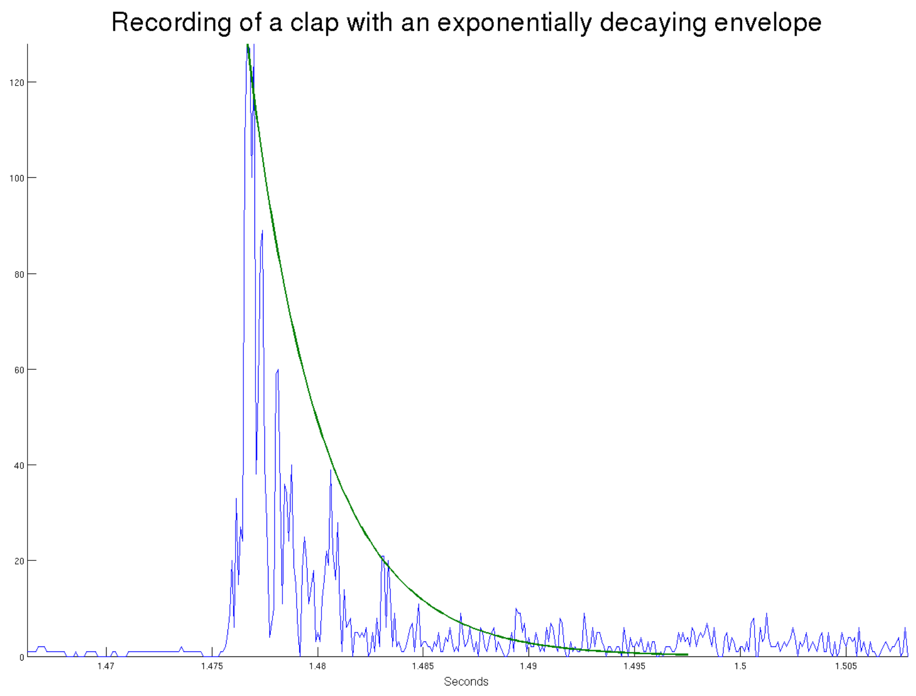


Fig. 6: A recording of a clap with an exponentially decaying envelope

3.2 Detection Error Tradeoff Curves

In order to judge the performance of the developed algorithms, detection error tradeoff (DET) curves were created. A DET curve shows the performance of a given binary decision algorithm by comparing the missed detection rate to the false positive rate at different decision thresholds.

To create the DET curves the following steps were executed:

1. The entire sample recording database was evaluated with a given detection algorithm, which assigned a *clap_likeness[.]* to each sample of every recording.
2. This *clap_likeness[.]* had several peaks, some corresponding to claps in the recordings, and some corresponding to other events.
3. With a given decision threshold it is easy to count the missed detections and false positives. All peaks in *clap_likeness[.]* corresponding to claps which are lower than the given decision threshold will be missed, while peaks corresponding to non-clap events which are higher than the decision threshold will be registered as clap events and constitute false positives.
4. Given the number of missed detections and false positives we calculated the corresponding missed-detection- and false-positive-rates.
5. By varying the decision threshold and determining these rates every time we got a (false-positive-rate, miss-rate) pair for every decision threshold.
6. We could then plot the these pairs on a graph with false-positive-rate as abscissa and miss-rate as ordinata, arriving at the final DET curve. The result for our three algorithms can be seen in figure 7.

Given the DET curve it is easy to find a decision threshold value which suits the needs of the application, depending on which type of mistake - false positives or missed detections - are more important to be avoided.

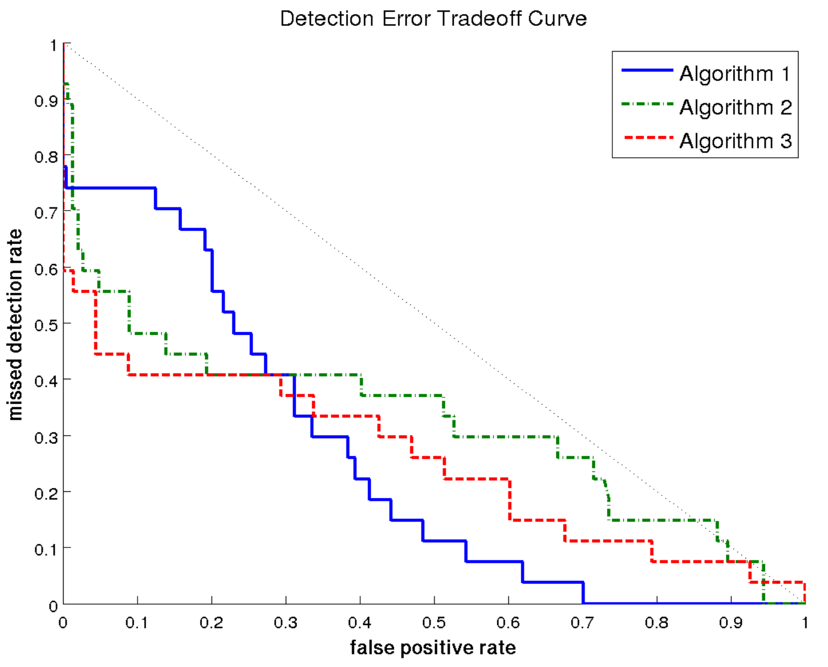


Fig. 7: Detection Error Tradeoff curve of all three algorithms.

3.3 Hardware Implementation

The microcontroller's limitations with respect to processing speed were already taken into account during the development and Matlab implementation of the algorithms. Thus, implementing the algorithms in C required little change. There was only one additional limitation that had to be considered and that generally lead to less accurate detections, which was the fact that the hardware could only handle integer variables. The compiler for the microcontroller would have been able to handle non-integer variables, but this is done software based and is very costly in terms of processor use. We were therefore forced to use integers, which in general resulted in a loss of accuracy and detection performance.

The change in performance due to the use of integers can easily be seen when comparing the DET curves of an algorithm with double and integer precision.

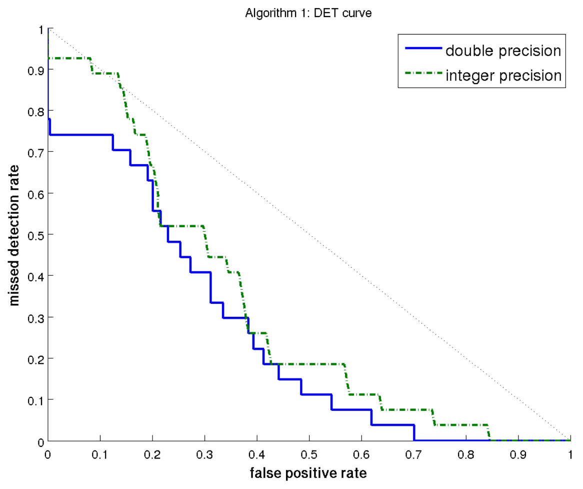


Fig. 8: DET curve for the first algorithm with double and integer precision.

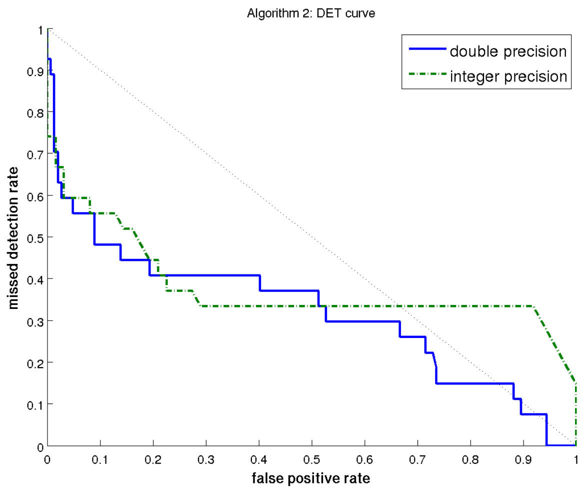


Fig. 9: DET curve for the second algorithm with double and integer precision.

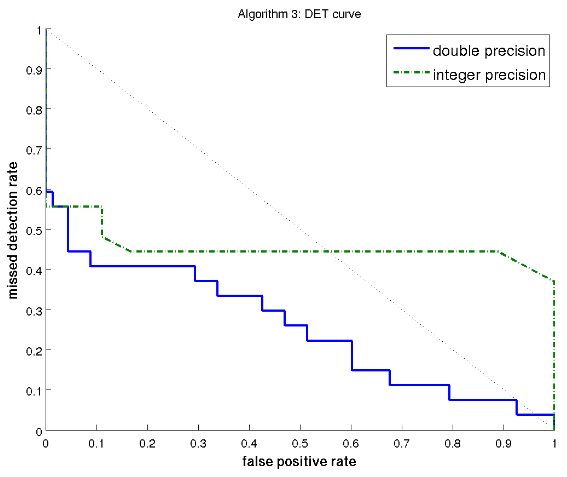


Fig. 10: DET curve for the third algorithm with double integer precision.

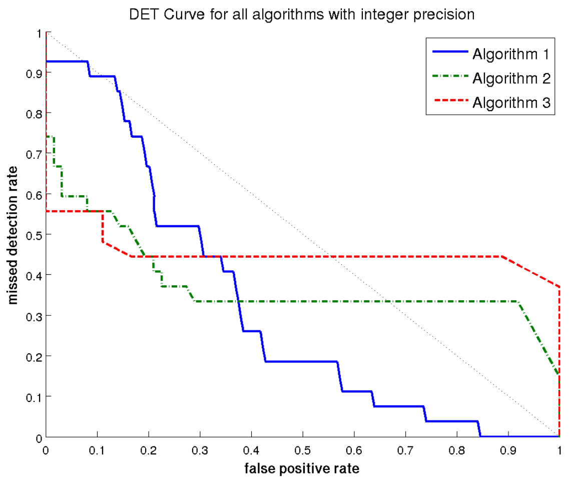


Fig: 11: Detection Error Tradeoff Curve for all three algorithms with integer precision.

4 Discussion

4.1 The Algorithms

Implementation of Algorithm 1

An important decision for Algorithm 1 was how to calculate the threshold. We decided that the threshold is a multiple of the long term average noise level, since this is similar to the logarithmic nature of sound amplification. We avoided actually using a logarithmic scale though, since it probably would have resulted in too many computations for the hardware.

The other decision was how the average would get calculated. Initially, we used every past sample in the recording to calculate the mathematically correct average. We had to abandon this approach because it required us to count all past samples, which can not be done indefinitely. We moved on to using a fixed-length window. An easy way of calculating the correct average of the window is summing up all its values once, then simply subtracting the oldest and adding the newest each time and then dividing the result by the window length.

Implementation of Algorithms 2 and 3

Since we designed algorithms 2 and 3 to behave the same apart from one key difference, most of the decisions taken were the same for both.

The major question was how to judge the clap likeness of a peak. We first tried summing up all the values which were consecutively above the threshold and dividing by time, but we quickly realized that we were simply calculating the mean value, which would lead to longer, quieter noises having similar scores to short, loud noises and thus the algorithm would end up classifying them as claps.

Instead we settled on taking the maximum value the window mean reached and dividing it by how long the noise was above the threshold. Testing showed that we achieved better results when we squared the maximum value first, which can be interpreted as putting more emphasis on the amplitude than on the length of the noise.

With both algorithms, we had to ignore noises that stayed above the threshold for too long. We therefore decided to use a maximum number of samples which the noise was allowed to be above the threshold before the algorithm decided that this couldn’t be a clap.

Specific to algorithm 3, the concerns about calculating the threshold arose again. It was important that the threshold was low in order to trigger many clap likeness peaks. That way accidentally filtering out claps that do not surpass the threshold is less likely. We tried again to use a multiple of the long term average noise level, but since this threshold was to be way lower than that of algorithm 1, we encountered problems when the average noise level was very low. The algorithm would then start analyzing each time there was the slightest of cracks, which we wanted to avoid by using a threshold.

Instead we used an added constant to calculate the threshold from the long term average noise. That way the threshold also wouldn’t exceed the limits of our A/D converter too early so that detection at higher noise levels is still possible.

Algorithm Performance

Overall, the different algorithms/-algorithm versions did behave quite as expected when compared to one another. If we take a look at the DET curves, we can see that the integer based versions generally didn't perform as well as those with double precision, and the more sophisticated the algorithms got, the better the results were in the area of interest.

Interestingly the simplest algorithm, algorithm 1, shows better performance than the more sophisticated ones in the region where we have a high false positive rate (see Fig. 7, bottom right hand corner). This is due to the fact that algorithm 1 produces many peaks which do not correspond to clap events. These peaks are generally very small compared to the peaks corresponding to clap events. Algorithms 2 and 3, on the other hand, produce peaks at fewer points, where certain conditions with regard to clap likeness are met. In this case a higher clap likeness value is attributed.
The better performance of algorithm 1 at high false positive rates is even more apparent in the case of integer precision (see Fig. 11). Here we can see that the algorithms 2 and 3 both rise very sharply when reducing the false positive rate (in the bottom right hand corner of the graph). This is due to the fact that several claps produced a clap-likeness score of 0. This loss in performance can be justified as follows: Divisions necessary to calculate the means tend to produce a lower result due to integer-division. Thus some short term means are lower and do not surpass the threshold to trigger a calculation of a clap-likeness value, whereas these same claps did result in a short term mean high enough in the case of double precision.

However, in the region of the DET plot most relevant for our application, the left hand side, where the false positive rate is low, the more sophisticated algorithms outperform algorithm 1. Here, the loss in performance of the algorithm 1 is the most severe. Algorithms 2 and 3 both show a slightly better performance with integer precision, although this is very dependent on the parameters chosen. A slight change in parameters can push the integer precision curve to the other side of the double precision curve. These small changes do not result in a noticeable change in performance when testing the algorithms on the board. Thus this apparent performance improvement of the integer precision algorithms 2 and 3 bear no real world significance. The fact that the integer precision versions of algorithms 2 and 3 performed much closer to their double precision counterparts is probably due to the fact that algorithms 2 and 3 have several parameters which yield more degrees of freedom when optimizing the algorithm for integer precision.

The algorithms also performed fast enough so that no problems arose when running them on the microcontroller. It was however too much to ask of the hardware to run the algorithms simultaneously, this quickly resulted in buffer overflows.

4.2 Suggestions for Additional Work

When we implemented the algorithms on the board and were testing them for the first time, we were taken aback by how loud we had to clap to set it off since we focused on getting rid of false positives during parameter optimization. This could have been prevented for example by choosing parameters with higher false positive rates. Future projects could also implement detecting a **double-clap** instead of just a single clap. Double sharp noises that are not claps probably occur less frequently in reality. This would allow for lower thresholds and therefore weaker double-claps to be detected without a higher false positive rate.

After we made progress with the recording process, we started focussing on how to judge the algorithms' performances. The more we understood what DET curves are about, the more we realized the important role they play. Upon implementing the algorithms on the AVR-board, we also realized that sometimes the echo from the room would be detected as a second clap. We therefore played around with a **cooldown time** after each confirmed clap, during which it is was not possible to detect a second clap. This significantly improved reliability on the board. Implementing this in Matlab and a more thorough analysis of this method would be desirable.

The parameters which we found were also limited in quality by how diverse and large our database was. In the end there were 58 sample recordings in the database. A much **bigger and more diverse database** (e.g. claps from farther away, claps in different rooms, claps by different people, etc.) would be desirable for a more general parameter optimization. Additionally, the DET curves would also have a nicer look to them with more data available.

4.3 Encountered Problems

A slow start

Setting up a working lab environment was much harder than originally expected and there were several challenges we had to overcome:

- No LEDs worked on the AVR-board (not even the power on LED) at the beginning. After we had measured the voltage of virtually every pin on the board we realized that the LEDs have been soldered in the wrong way around.
- We had very little experience working with AVR microcontrollers, so we had to take our time getting to know the possibilities we had when using the board.
- Since our lab computer was managed, we didn't have root permissions on the usbasp programmer which was needed to load code onto the microcontroller.
- When trying to make our first recordings we randomly got bytes sent twice whenever we used a baud-rate higher than 9600. Since this was still during a stage of getting familiar with the environment, it took us several weeks to figure out what the problem was before we could record our first reasonable example signal. In the end it wasn't our programming and it wasn't some kind of noise on the board, it was the USB-to-serial converter we used that couldn't handle higher baud-rates (which wasn't documented).

- Another problem we had when trying to make recordings was figuring out how to receive them on the computer. The linux program “gtkterm” was initially good enough for debugging, but it quickly became apparent that it wouldn’t be suitable to use it long-term. We first tried to write a C program to capture sound input from the serial port before we found that Matlab offered convenient ways of reading and saving serial input data.

Hardware challenges

Since it was part of the plan to use less powerful hardware, there was quite a struggle with it at times, one of the main difficulties being debugging. The graver errors like missing semicolons were thankfully caught by the compiler, but for all the other errors we had to improvise.
Much of the debugging could be done by using the UART, but at some point we noticed a bug caused by the routine we used to send C-strings, the whole main-routine started looping. We assumed that it must be an error caused by moving around the pointer of the C-string. From that point on, we only relied on the onboard-LEDs for debugging.

Some problems were caused because of us not being very familiar with the ways of avr-gcc. One example would be that the board at some point messed up when trying to perform addition and a modulo-operation on the same line of code. This led to us believing that we were not able at all of performing several operations on the same line, which then made us write overly complicated code. We later figured out that we were in fact able to write everything on the same line, the source of the initial error was not found, it could be speculated that the problem was not using volatile for variable declaration.

At first we relied on interrupt routines triggered by the A/D-converter when we were trying to set up a way to make recordings. We never got higher than the 2 kHz mark without getting overflows when measuring the sampling rate, until we got the tip from our supervisors that interrupt routines might take too long to leave enough processing time for sending the data. As it turned out, polling the ADC was the way to go.
Another issue was figuring out the rate at which we were able to sample the signal. The A/D-converter had at its lowest a sampling rate of 9615 Hz. We settled on measuring the effective sampling rate by recording a sine wave that gets paused for one second. The result we got after optimizing the C code for sending audio input confirmed that we were in fact able to reach the A/D-converter’s speed of 9615 Hz.

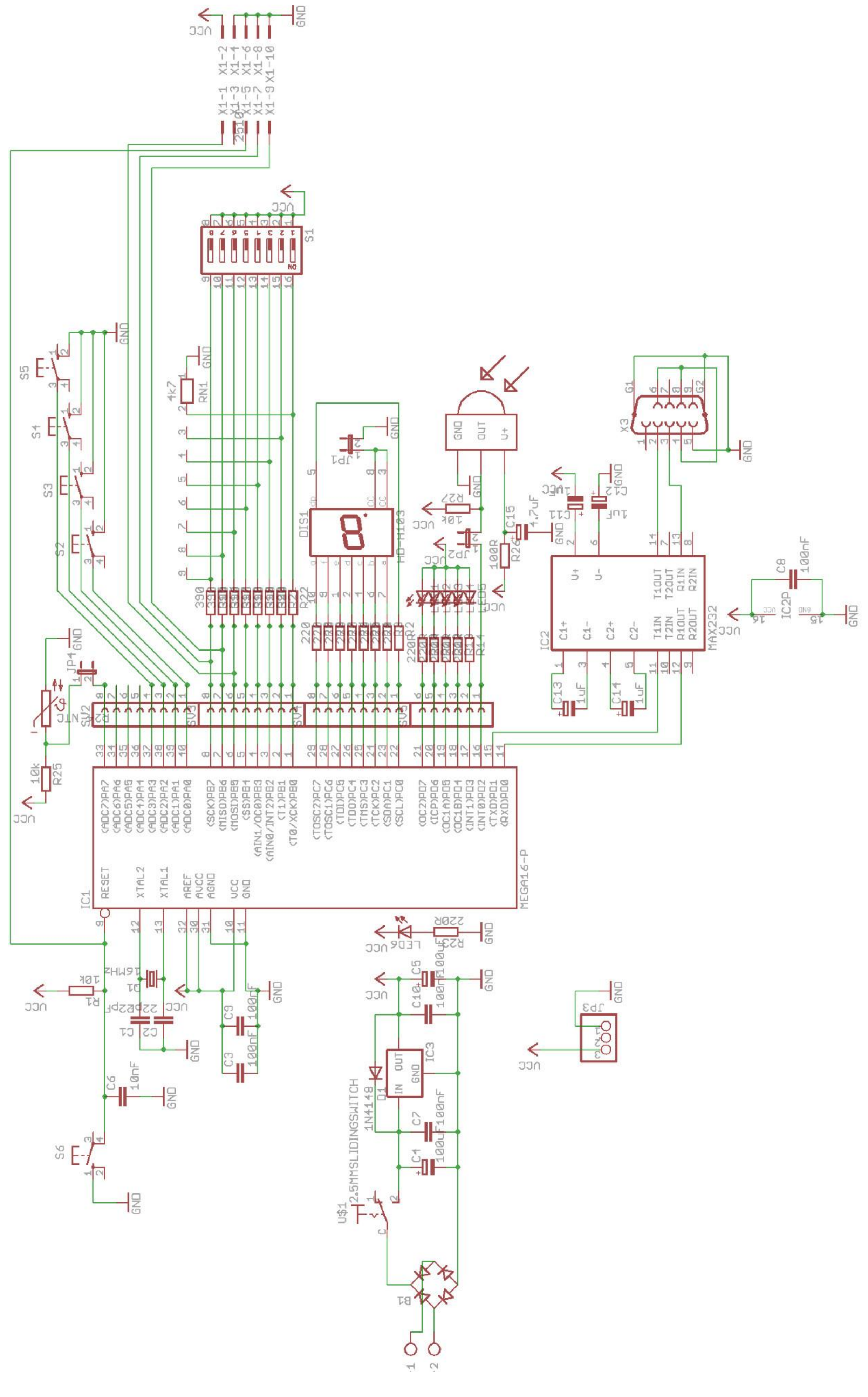
Acknowledgements

This work was done in the in the context of our group work for the bachelor in electrical engineering at ETH Zürich. We would like to thank our supervisors, Dr. Beat Pfister and Tofigh Naghibi for their advice and support.

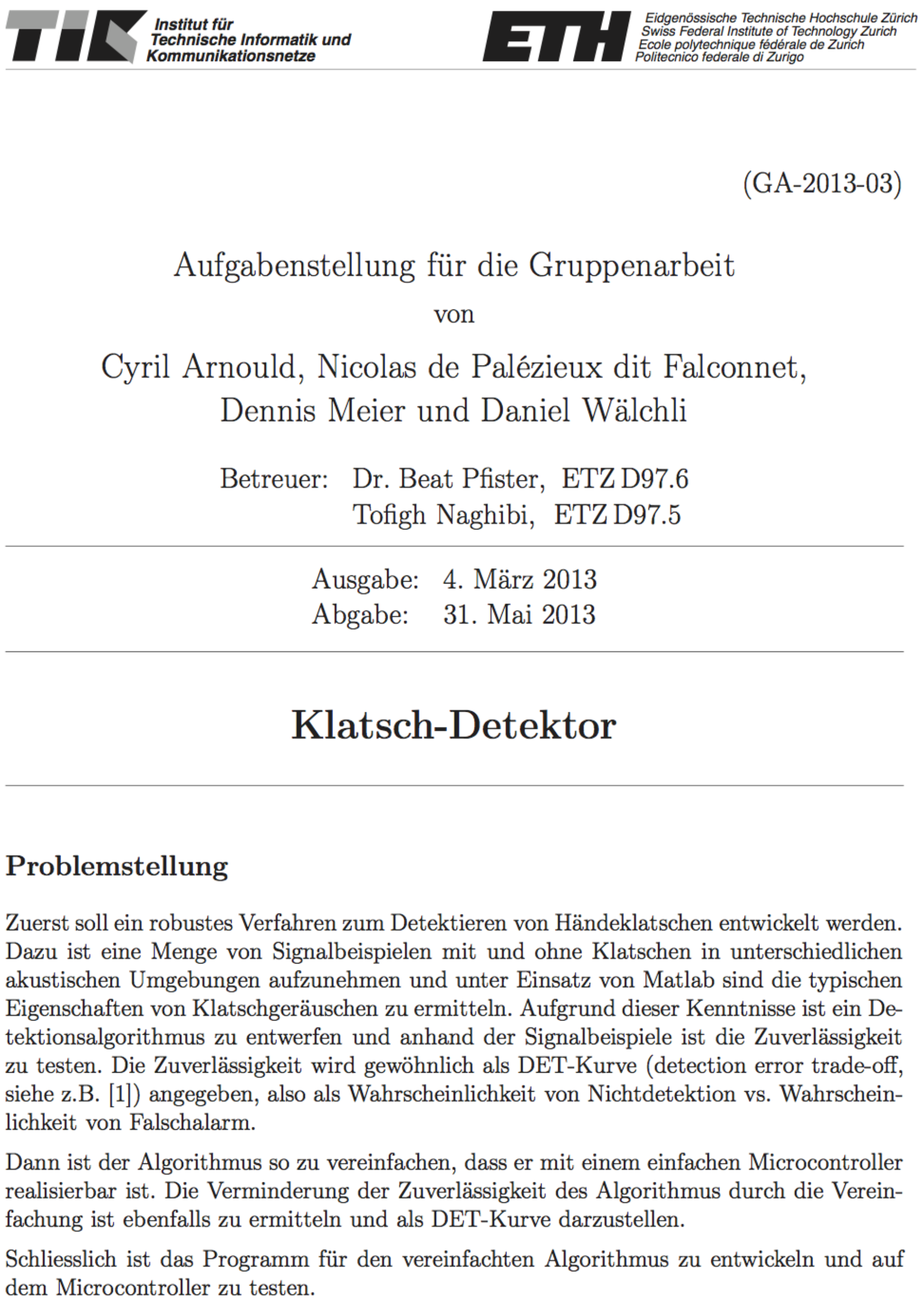
Sources

1. A. Martin and G. Doddington, et a.; “The DET curve in assessment of detection task performance”. In Proceedings of Eurospeech, pages 1895-1898, 1997; http://www.itl.nist.gov/iad/mig//publications/storage_paper/det.pdf
2. AMIV Bastli, “AVR-Demoboard resource page”; http://www.bastli.ethz.ch/index.php?page=bausaetze%2FAVRdemoboard
3. AMIV Bastli, “usbasp 2.0 programmer resource page”; http://www.bastli.ethz.ch/index.php?page=bausaetze%2FuJTAG
4. Atmel, “ATmega32 Datasheet”; http://www.atmel.com/Images/doc2503.pdf
5. Revolution Education Project, “Simple Sound Detection Circuit”; http://www.picaxe.com/docs/picaxe_sound.pdf
6. Toddy Cangica, electronics-lab.com, “Microcontroller Clapper Switch” http://www.electronics-lab.com/projects/mcu/020/index.html
7. Embedded-Lab Website, “Making a simple clap switch”; http://embedded-lab.com/blog/?p=6439
8. stevenSwift, Arduino forum, “re: clap detect”; http://forum.arduino.cc/index.php?topic=70740.msg525716#msg525716
9. Electret, “capsule microphone & low noise adjustable gain microphone amp datasheets”; http://www.adafruit.com/datasheets/CMA-4544PF-W.pdf http://www.adafruit.com/datasheets/MAX4465-MAX4469.pdf
10. B. Pfister and T. Kaufmann; “Sprachverarbeitung: Grundlagen und Methoden der Sprachsynthese und Spracherkennung”, pages 59-69.

Appendix A: Schematics AMIV Demoboard



Appendix B: Aufgabenstellung des Instituts



Institut für Technische Informatik und Kommunikationsnetze

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

(GA-2013-03)

Aufgabenstellung für die Gruppenarbeit
von
Cyril Arnould, Nicolas de Palézieux dit Falconnet,
Dennis Meier und Daniel Wälchli

Betreuer: Dr. Beat Pfister, ETZ D97.6
Tofigh Naghibi, ETZ D97.5

Ausgabe: 4. März 2013
Abgabe: 31. Mai 2013

Klatsch-Detektor

Problemstellung

Zuerst soll ein robustes Verfahren zum Detektieren von Händeklatschen entwickelt werden. Dazu ist eine Menge von Signalbeispielen mit und ohne Klatschen in unterschiedlichen akustischen Umgebungen aufzunehmen und unter Einsatz von Matlab sind die typischen Eigenschaften von Klatschgeräuschen zu ermitteln. Aufgrund dieser Kenntnisse ist ein Detektionsalgorithmus zu entwerfen und anhand der Signalbeispiele ist die Zuverlässigkeit zu testen. Die Zuverlässigkeit wird gewöhnlich als DET-Kurve (detection error trade-off, siehe z.B. [1]) angegeben, also als Wahrscheinlichkeit von Nichtdetektion vs. Wahrscheinlichkeit von Falschalarm.

Dann ist der Algorithmus so zu vereinfachen, dass er mit einem einfachen Microcontroller realisierbar ist. Die Verminderung der Zuverlässigkeit des Algorithmus durch die Vereinfachung ist ebenfalls zu ermitteln und als DET-Kurve darzustellen.

Schliesslich ist das Programm für den vereinfachten Algorithmus zu entwickeln und auf dem Microcontroller zu testen.