Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Haptic, Acoustic, and Visual Short Range Communication on Smartphones

Distributed Systems Lab

Marcel Bertsch, Roland Meyer

bertschm@student.ethz.ch, romeyer@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Pascal Bissig, Philipp Brandes
Prof. Dr. Roger Wattenhofer

December 22, 2014

Abstract

Communication between smartphones relies mainly on radio frequencies. In this lab we explore other ways to communicate with built-in hardware using vibrators, accelerometers, microphones, speakers, flashlights, and cameras. We evaluate the feasibility and performance of the following haptic/audible channels: Vibration to accelerometer, vibration to microphone and speaker to microphone. We also examine visible light channels between flashlight LEDs and cameras introducing an algorithm to detect and decode messages sent over the flashlight. Finally, we introduce *BlinkEmote*, a user-friendly application that allows for bi-directional communication between smartphones over such a channel.

Contents

Chapter 1

Introduction

In recent years, smartphones have become very common and also very powerful. One of their main fields of application is communication and the transmission of data. For this purpose modern phones make use of a variety of different technologies, such as WiFi, Bluetooth, or GSM. What most of these “traditional” channels have in common is that they rely on sending and receiving radio-frequency (RF) signals. Other means of communication are not widely deployed and still a hot research topic.

In this lab we explore various alternative ways which allow two off-the-shelf smartphones to communicate over short distances without relying on RF signals. With today’s smartphones featuring a plethora of powerful sensors and actuators combined with the continuously improving computational power the possibilities are vast. We investigate some of them by conducting a series of experiments using different types of Samsung and HTC smartphones running Android.

In Chapter 2 we focus on establishing and examining acoustic and haptic communication channels, using the phone’s speaker, microphone, vibrator, and accelerometer.

In Chapter 3 we use visible light from the phone’s LED flashlight to send short messages to another phone’s camera. We show that the phone can detect a blinking light and decode information from it. We also introduce the app *BlinkEmote* that makes use of our findings to transmit emoticons over the flashlight-camera channel.

We conclude both of these chapters with a short summary of our findings.

Chapter 2

Vibration Communication

In this chapter we take a look at how two off-the-shelf smartphones can form a haptic communication channel. We evaluate communication channels utilizing vibration, accelerometers, microphones, and speakers.

2.1 Related Work

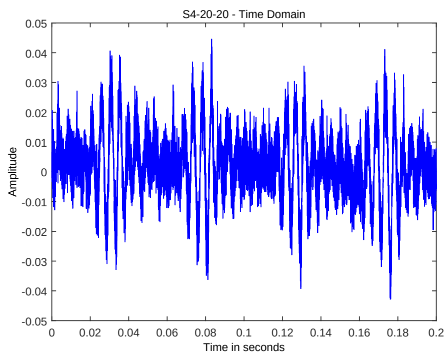
Similar work has been done by A. Studer et. al. [1], where they use such a channel to authenticate future messages sent over a radio-channel, and by I. Hwang et. al [2], where they place two phones on a common surface to transmit messages through the propagation of a vibration pattern. To tie in with these we first focus on exploring the possibilities such a vibration-accelerometer-channel offers, then try to improve it by using other sensors and actuators and compare the different approaches.

2.2 Vibrator to Accelerometer

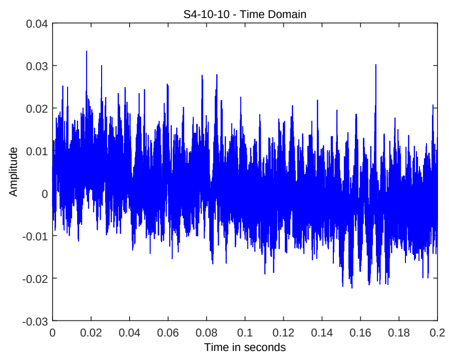
2.2.1 Vibrator

Most of today's smartphones have a built-in vibrator, which is typically used to notify the user of events such as incoming messages when the phone is in silent mode. We evaluated how well the vibrator can be used to encode messages by toggling it according to different patterns. Android's Vibrator API provides us with a convenient feature to do just that. Since the documentation does not specify how small the time intervals of such patterns may be, we built a small app to run a few tests. We placed the Galaxy S4 on a carpet floor and recorded different vibration patterns with a studio microphone, sampling at 192000Hz.

Figure 2.1a shows the amplitude of the sound signal in the time domain as produced by a 20-20 vibration pattern (i.e., the repetition of 20ms of vibrating followed by a break of 20ms). The peaks are clearly visible and occur at regular intervals of 23.8ms (averaged over 3 seconds), which shows us that the accuracy



(a) 20-20 pattern



(b) 10-10 pattern

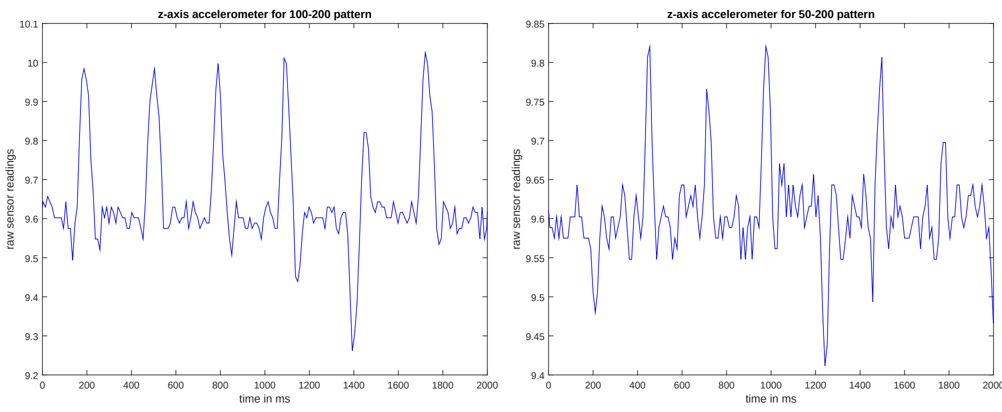
Figure 2.1: Amplitude of vibration patterns produced by the Galaxy S4, recorded by a studio microphone.

is not perfect, but that the precision is quite good. Figure 2.1b shows a 10-10 pattern, where we see that the noise is too large to see any clear peaks or regularities. We repeat the experiment with the Galaxy S2 and reach similar results, i.e., 23.3ms intervals (averaged over 1.5 seconds) with a 20-20 pattern and no clear peaks with a 10-10 pattern. From this we conclude that 20ms is a reasonable lower bound on our control over the vibration duration.

2.2.2 Accelerometer

The accelerometer is a sensor that measures acceleration applied to the phone in all three dimensions. In Android an application can register with the so called *SensorManager* to receive updates whenever new raw sensor readings are available. In order to find out how fine-grained they are, we let the Galaxy S4 vibrate with various different patterns and place it on a wooden table next to the Galaxy S2, which records the sensor data. From the log messages we know that the accelerometer is sampled at 100Hz on average (every 10 milliseconds). Figure 2.2a shows the readings along the Z-axis[1] for a 100-200 pattern, where we can clearly see the individual peaks produced by the vibrator. The time between the peaks is 107.1ms (averaged over 3 seconds), so again we see the lack of accuracy of the vibrator. For comparison, Figure 2.2b shows the sensor readings for a 50-200 pattern. Here we still see the peaks, but they are much more irregular.

[1]After subtracting the effects of gravity all three axes produce similar results, but the Z-axis shows the most distinct shape.



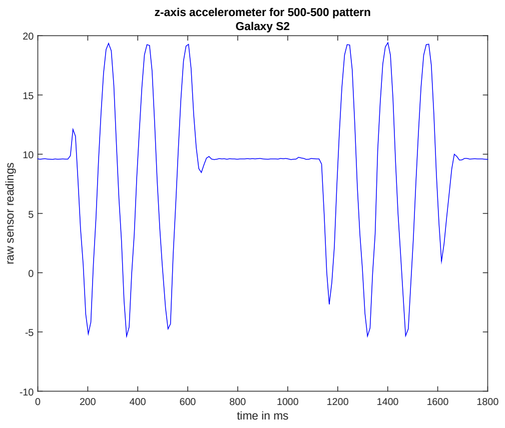
(a) 100-200 pattern (b) 50-200 pattern

Figure 2.2: Amplitude of vibration patterns produced by the Galaxy S4, recorded by Galaxy S2 (showing only accelerometer's Z-axis).

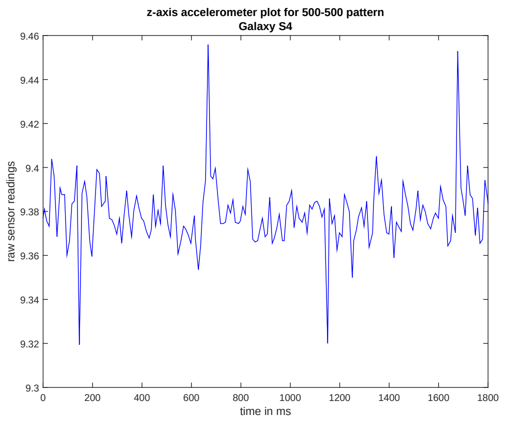
2.2.3 Evaluation

We conduct a series of experiments in which we place the Galaxy S2 and the Galaxy S4 on different surfaces and in different relative positions, where one of them vibrates with a 500-500 pattern and the other one records the acceleration along the Z-axis. From the results we make a number of interesting observations:

- Placing the phones side-by-side on a wooden surface yields the best results, as opposed to stacking them on top of each other or placing them on textile or stone surfaces.
- The accelerometer readings of the Galaxy S4 show a high amount of noise (compare Figure 2.3a to 2.3b). The data is bad even when recording the acceleration from its own vibrator. We conclude that accelerometers among other sensors can be heavily device dependent and thus have to be used with care.
- When we place the phones on a large wooden table, e.g., the kind found in lecture rooms, we are able to reliably record the vibration over a distance of 6m. We assume that the vibration pattern would propagate much further, but we were unable to find a suitable surface to test this assumption.



(a) Galaxy S2



(b) Galaxy S4

Figure 2.3: Comparison of measurement quality for accelerometer of different devices.

2.3 Vibrator to Microphone

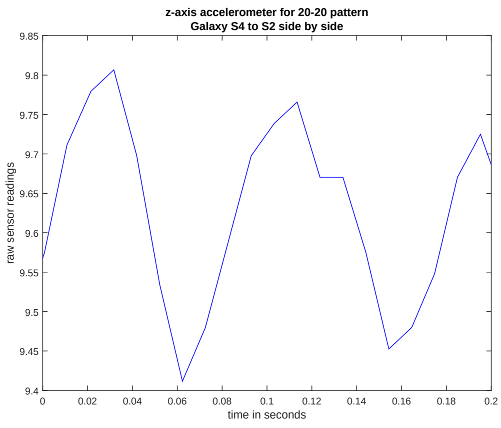
2.3.1 Accelerometer versus Microphone

Experiments with the accelerometer show severe device dependency and low sampling rates. With the microphone, however, we have a very high sampling rate and less device dependency. Figure 2.4 shows the recording of a 20-20 vibration pattern comparing accelerometer and microphone. Apparently, the vibrator is loud enough to be recorded by the microphone over a short distance and thus we can use it to record vibrations at a much higher sampling rate than with the accelerometer.

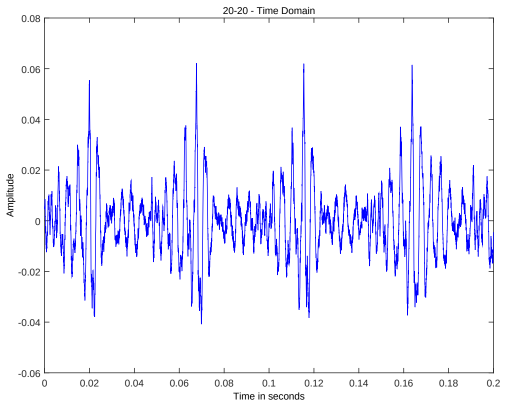
2.3.2 Detecting Vibrator with Microphone

Unlike the accelerometer the microphone is prone to acoustic noise. This forces us to find a way to separate the vibrator's sound from the rest, hence we have to analyze the vibrator's audible footprint. To do this we transform the recorded audio signal from the time into the frequency domain using Fast Fourier Transform. Figure 2.5b shows the vibrator's frequency spectrum clearly indicating three peaks at around 204Hz, 408Hz and 613Hz, whereas these peaks are not present in a silent environment as shown by Figure 2.5a.

Holding the phones back-to-back while vibrating improves the signal propagation drastically. An additional peak at around 816Hz can be observed. Figure 2.6 suggests that the vibrator operates at roughly the same frequencies on all tested devices which enables device independent filtering of the vibrator signal. To check whether this could still work in a noisy environment we compare the



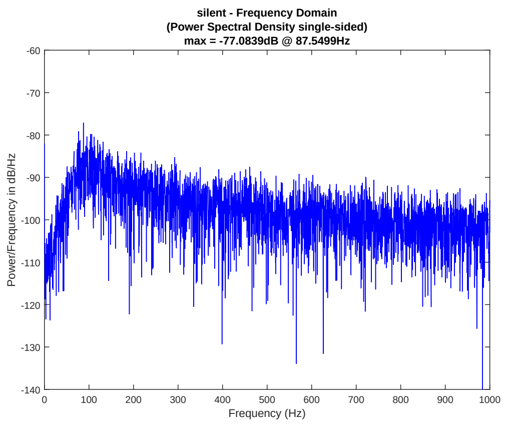
(a) Accelerometer



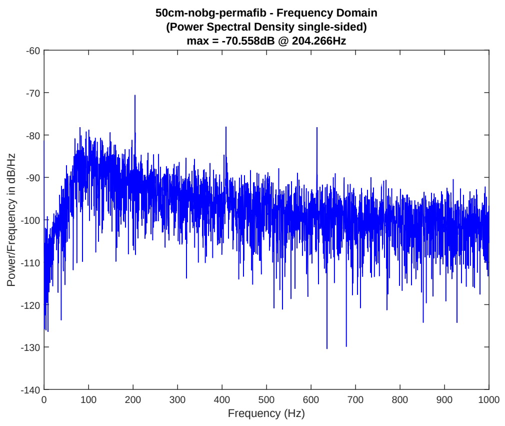
(b) Microphone

Figure 2.4: Recording vibration with accelerometer versus microphone. Galaxy S4 is vibrating, S2 recording, on a wooden table side by side.

vibrator recording to a recording taken in a cafeteria during lunch time which is shown in Figure 2.7. We conclude that at least the 204Hz frequency could be extracted when holding the phones back-to-back in such an environment.



(a) Silence

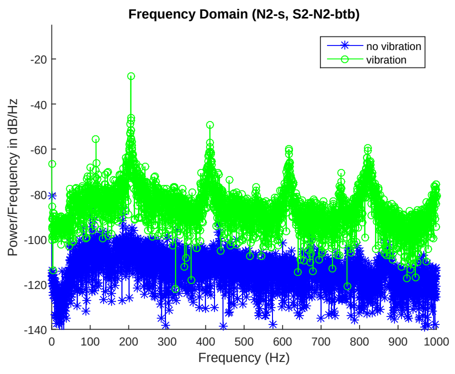


(b) Vibrating 50cm away

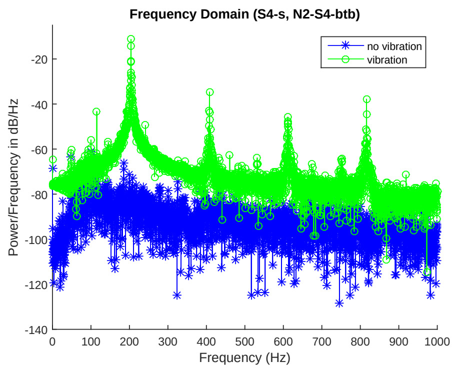
Figure 2.5: Audio recorded by Galaxy S2 in frequency domain, silent versus S4 vibrating 50cm away on concrete floor. Higher frequencies are much lower and thus omitted.

2.3.3 Matched Filter

Although technically FFT could be used to transform small samples of audio data into frequency domain and analyze the signal there, it is not very practical



(a) Galaxy S2 to Note 2



(b) Note 2 to Galaxy S4

Figure 2.6: Frequency domain of recorded vibration holding the phones back-to-back compared to recording without any vibrating phone nearby.

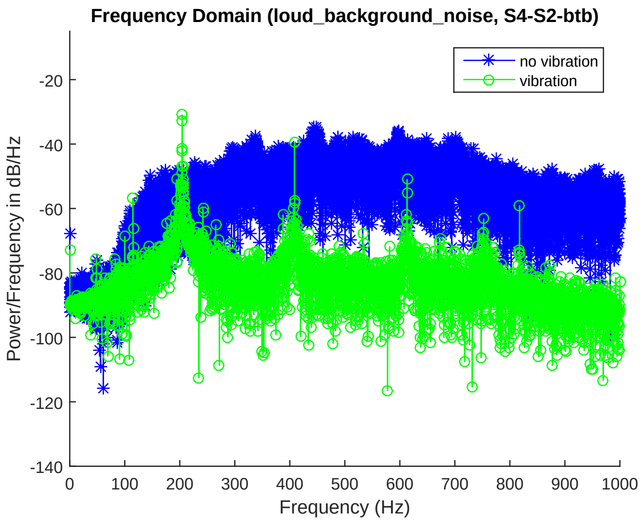


Figure 2.7: Galaxy S4 vibrating and S2 recording back-to-back compared to a noisy background (cafeteria at lunch time).

because of the computational overhead. As an alternative we take a look at the *Matched Filter* technique which we apply directly to the audio signal of a vibration pattern (20ms vibrating, 300ms pause) in the time domain. As a template we manually extract a single 20ms vibration segment and use it on the full signal. As a result we get the correlation of the template within the signal, peaking on the best match (see Figure 2.8). In the presence of background noise it gets more difficult to identify the vibration sound and the result is partially flawed as can be seen in Figure 2.9b. As a noisy environment we again use the cafeteria at lunch time which features a lot of variation in frequency and intensity. We conclude that, in general, Matched Filter can be used to detect the vibration signal in a noisy environment as long as the noise is not too similar to the vibration frequency.

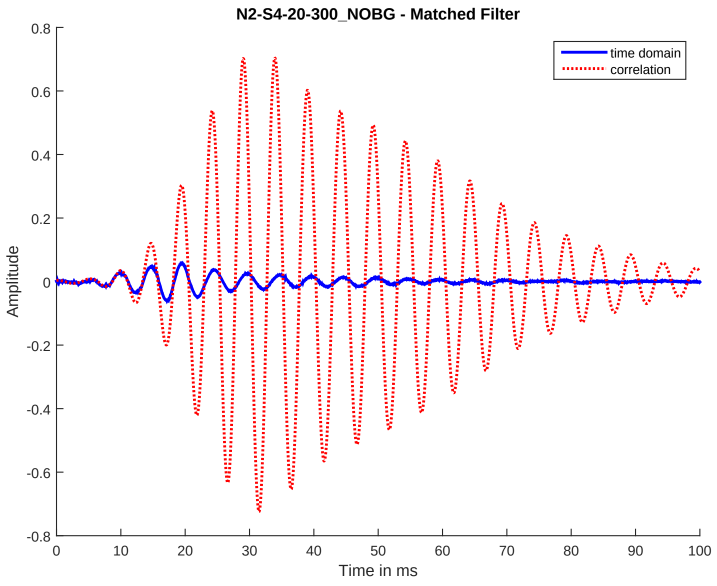
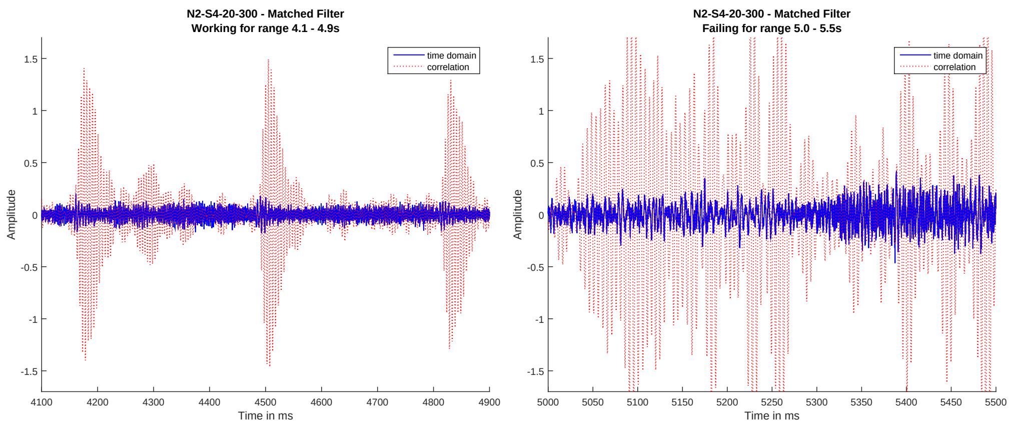


Figure 2.8: Note 2 to Galaxy S4 vibration detection with Matched Filter.

2.4 Speaker to Microphone

One of the limitations of the vibrator is that it is not possible to control the volume or frequency of the sound it produces. Having shown that we can use a microphone to record a vibration pattern and extract information from it we look at the alternative of using the phone's speaker instead to produce the desired frequencies. For this we perform a few tests in which we use one of the phones to produce a sound at 204Hz and another phone records it.



(a) Vibration can be detected (b) Vibration cannot be detected

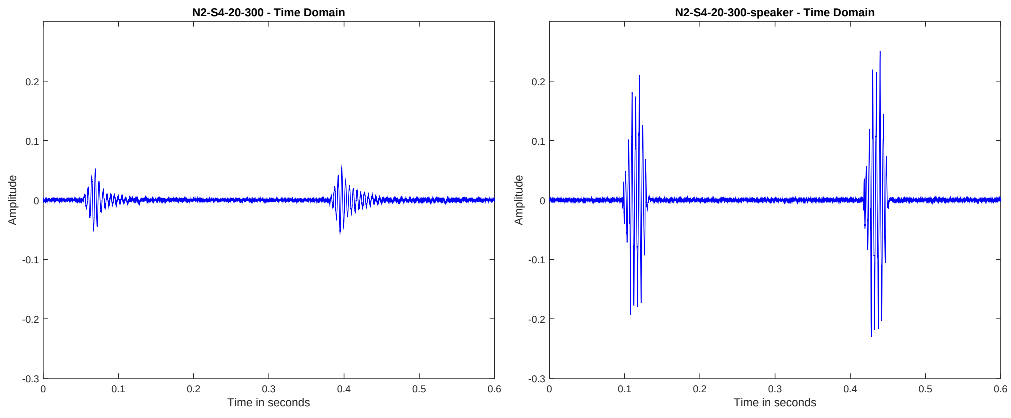
Figure 2.9: Note 2 to Galaxy S4 vibration detection with Matched Filter in a noisy environment. On the left 3 peaks at roughly 300ms intervals are visible. On the right loud background noise produces many false-positives.

2.4.1 Speaker Volume

When applying the Matched Filter technique to recordings from a quiet environment, the speaker approach yields similar results to the ones we encountered when using the vibrator. Examining the signal in the time domain shows however that the speaker's peaks are of significantly higher magnitude as can be seen in Figure 2.10. As a consequence, the speaker has a clear advantage over the vibrator in a noisy environment as shown in Figure 2.11. In this scenario we can simply increase the speaker's volume such that its pattern can still be detected by the Matched Filter, whereas the vibrator's pattern is mostly drowned out by the noise, resulting in fewer matches.

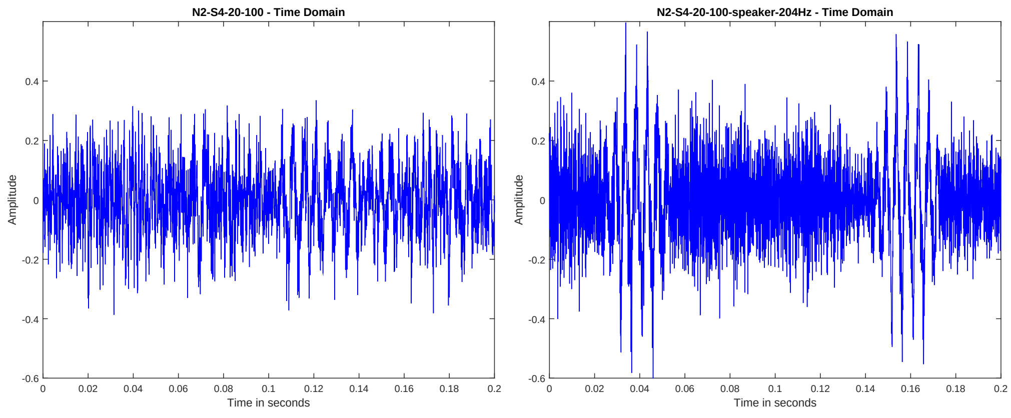
2.4.2 Speaker Frequency

In addition to adapting the volume we can change the frequency of the generated tone. With multiple frequencies we can encode information in an intuitive way. Figure 2.12 shows a pattern of six different frequencies (261, 293, 329, 349, 391 and 440Hz) played at half of the maximum volume level. Those frequencies lie close together in a small area of the audible frequency band whereas the frequency spectrum is much larger and we could possibly use frequency modulation to achieve high data rates. However, since plenty of research has already been done on the subject [3, 4, 5], our aim is not to optimize the data rate of acoustic data channels but to conduct a simple feasibility test on smartphones. Using Matched Filter with templates of different frequencies we can indeed distinguish the tones quite well as shown in Figure 2.13.



(a) Vibrator (b) Speaker

Figure 2.10: Recording vibrator compared to speaker without background noise. The amplitude is higher and more controllable with the speaker.



(a) Vibrator (b) Speaker

Figure 2.11: Recording vibrator compared to speaker with background noise. The vibrator sound gets drowned by the noise while the speaker can be adapted by increasing the volume.

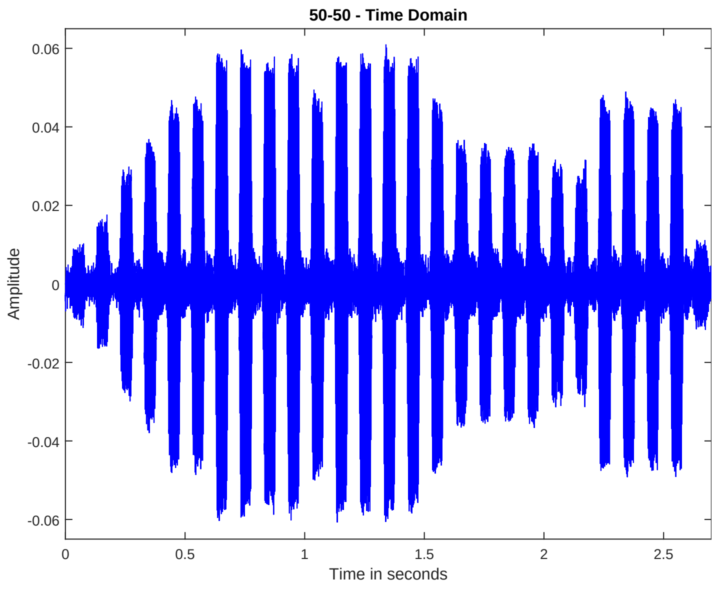


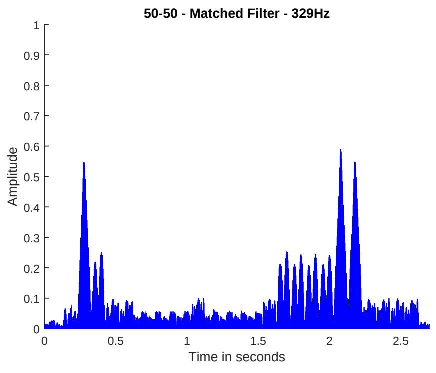
Figure 2.12: Galaxy S4 to S2 sending different frequencies in a 50-50 pattern. The frequency pattern is 261, 293, 329, 349, 391, 391, 440, 440, 440, 440, 391, 440, 440, 440, 440, 391, 349, 349, 349, 349, 329, 329, 391, 391, 391, 391, 261Hz.

2.5 Conclusions

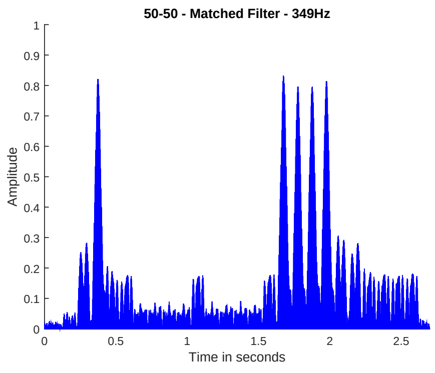
We have shown that it is possible to use off-the-shelf smartphones to establish a communication channel between the vibrator and the accelerometer. Due to the limitations in control and speed of these two devices such a channel would reach only low throughput. Our experiments showed that the vibration signal propagated much further than expected, which makes such a channel unsuitable for security applications in an environment where the propagation medium cannot be controlled. Furthermore, since the performance of the accelerometer as well as the vibrator depends on the phones that are used we believe that such a channel would require too much parameter tuning and calibration overhead to be practical for efficient data transmission. It is, however, suitable to transmit very small amounts of information, as was shown in [1].

Using the microphone instead of the accelerometer, due to a higher sampling rate, increases the speed at which data can be transmitted. It also allows for a longer communication distance. The downside however is that this channel is prone to heavy noise, depending on the environment. Given the fact that the vibrator's volume cannot be increased to cope with such a scenario we see no advantage over other communication methods.

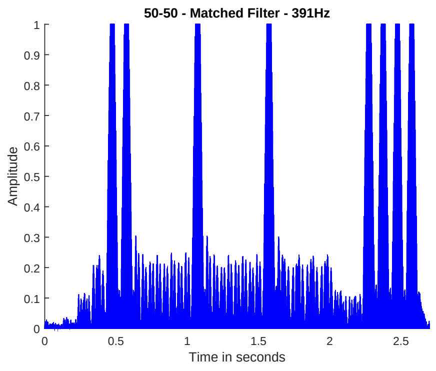
Replacing the phone's vibrator with the speaker turned out to be an improvement due to the ability of changing volume and frequency. This opens up the possibility to encode information not only in the time domain but also in the frequency domain, which we briefly explored. At this point we decided to not pursue this topic any further since a lot of research has already been done on communication over an acoustic channels. Multiple sophisticated implementations already exist on smartphones [4] [5].



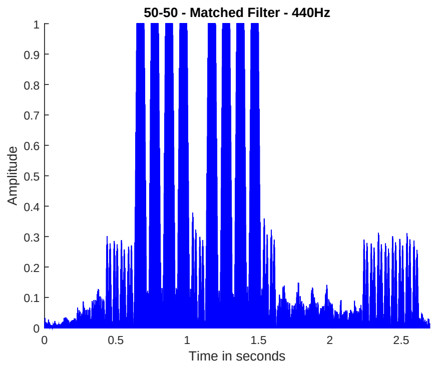
(a) 329Hz



(b) 349Hz



(c) 391Hz



(d) 440Hz

Figure 2.13: Matched Filter with different frequency templates. One can clearly see which frequencies are present at which times in the pattern.

Chapter 3

Visible Light Communication

In this chapter we explore how visible light communication (VLC) can be implemented on smartphones across distances of a few meters and what the possibilities and limitations are.

3.1 Related Work

M. M. Galal et. al. [6] [7] showed how the LED of a smartphone can be used to form a secure one-way visual channel to a photodetector over a very short distance, e.g., to send the information from a magnetic card to an ATM. J. Ekberg et. al. [8] propose a system that uses cameras, screens, and LEDs of two smartphones to provide mutual authentication when they are close to each other. In contrast, our goal is to use off-the-shelf smartphones only, i.e., no additional hardware as is the case with [6] [7], and build a system for sending messages across longer distances, similar to Morse code.

3.2 Algorithm

For our visible light communication channel we developed a data encoding schema and an algorithm consisting of two parts. One part is the encoding and decoding of the message and the other is detecting the blinking part in video frames used to locate the sender. In combination these parts can be used to send and receive short messages (a few bits) or even locate a person in a crowd, for example in a stadium, based on a characteristic blinking pattern.

3.2.1 Encoding and Decoding

We communicate by turning the flashlight LED on and off in time slots of fixed length, which have to be large enough for the camera to detect. Relying on smartphone cameras limits the speed of transmission severely due to their low frame rate. The sender repeats its signal without preamble, but with a detectable pause in-between such that the receiver can at any time connect to the sender and then record until one message is completely sent and decode. As a payload we choose short binary messages of four to eight bits. Since our carrier signal is also binary we modulate `zero` as `01`, `one` as `011` and `pause` as `00` where `0` stands for LED off and `1` for LED on. While this might not be optimal in terms of data transmission rate, it turned out to work well with our non-synchronized timing slots as well as for the blinking detection.

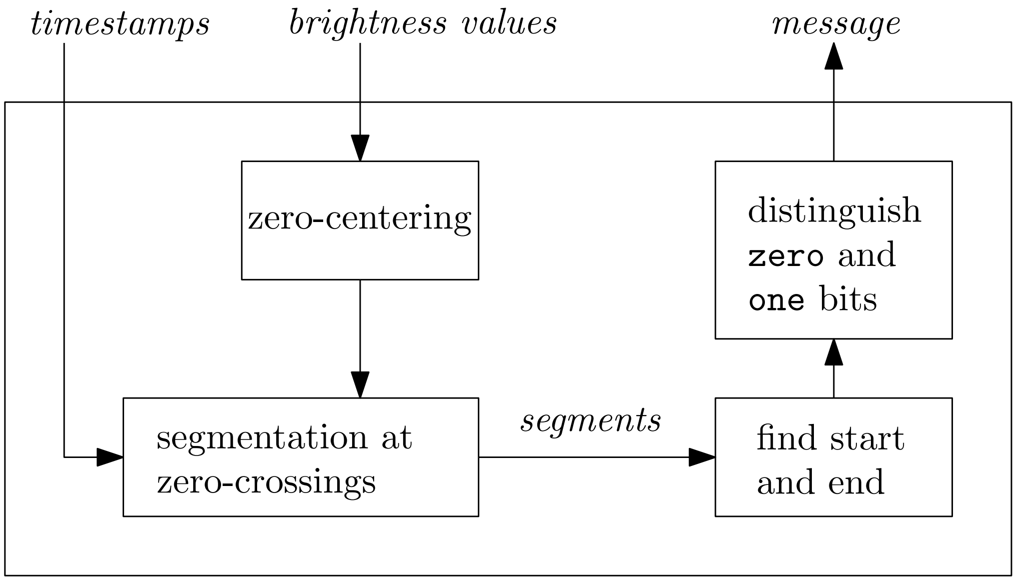
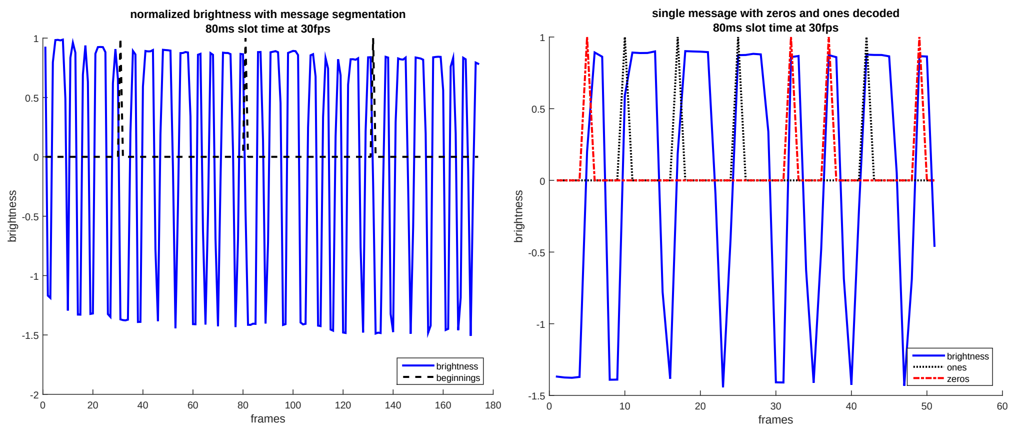


Figure 3.1: Decoding a message from brightness values and their corresponding timestamps.

The decoding part of the algorithm takes a vector representing the brightness over time for the image part of the video where the blinking was detected. As *brightness* we denote the sum of the grayscale pixel values. For each video frame we get one brightness value together with the frame's timestamp. The timestamp is required because the frame rate may vary and unfortunately we cannot assume the frames to be equidistant in time. As a first step, we subtract the mean from each brightness value, thereby zero-centering the data. Then the vector gets partitioned along the zero-crossings into segments which are either above or below zero and with a duration assigned which is the difference between the first data point in the next segment and the first in the current segment. Thanks to the low speed of the camera the slot times are high enough to distinguish between single and double slots even with jitter and without clock synchronization. Negative segments of two slots mark the transition from one message to the next (see Figure 3.2a). Positive segments in between correspond to the payload bits according to their duration where a single slot means `zero` and a double slot means `one`. Figure 3.2b shows an example decoding for the message `01110010` with the positive segments mapped to the corresponding bits. Figure 3.1 summarizes the decoding algorithm.

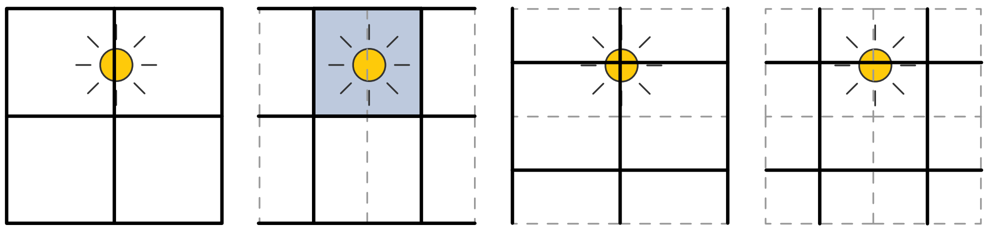


(a) Segmentation (b) Decoding

Figure 3.2: Decoding algorithm for 80ms slot time at 30 frames per second. Extracting a single message and decode to `01110010`.

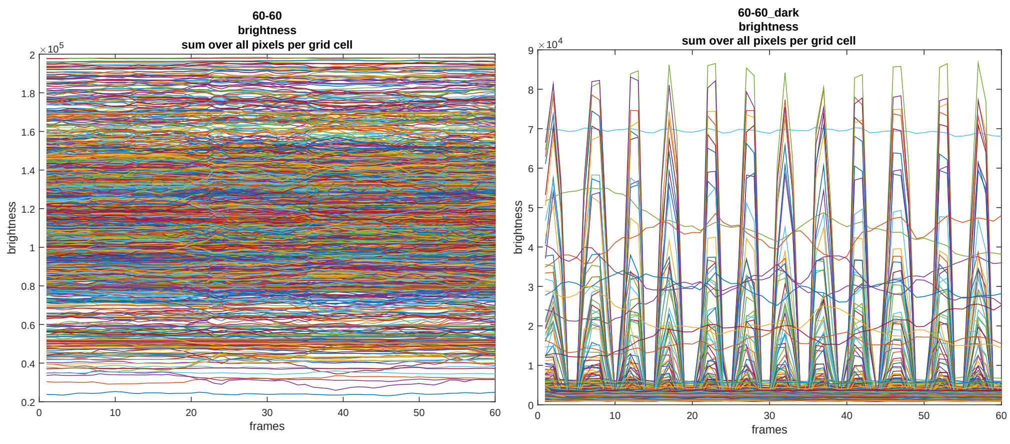
3.2.2 Blinking Detection

To feed the decoding part with the correct data we have to locate the part of the image where the blinking resides. Therefore, we lay a grid over every video frame and compute the brightness value from the grayscale pixels in each grid cell. Choosing a suitable cell size is crucial. If the cells are too large the blinking might be drowned by the noise in the cell, if the cells are too small the blinking is likely to jump from one cell to another due to camera shake. The optimal size depends on the distance between the sender and receiver and thus is a parameter for this algorithm provided by the user. Experimentally we figured out that a grid cell size of around 30 pixels (at a resolution of 640×480) works well for distances up to 10 meters. Note that in general it is better for detection to use small cells, however the smaller the cells the more computational effort is required from having to analyze brightness over time for each cell. Things get even worse because we need to shift the grid to avoid the blinking to hit a border between two or more cells. We shift the grid by half of the cell size horizontally, vertically and in both directions resulting in a total of four grids which results in a large number of cells (see Figure 3.3). Unfortunately this restricts the video resolution and with it the maximum distance at which we can detect blinking due to the limited computational power on smartphones. In theory however, the algorithm could be used for arbitrary resolutions and small grid cell sizes enabling blinking detection on greater distances with a stable camera setup.



(a) Original grid (b) Horizontal shift (c) Vertical shift (d) Combined shifts

Figure 3.3: To increase the probability that the blinking lies completely within one of the square cells, as is the case in b), the grid is shifted by half the grid size in both directions, resulting in 4 grids.



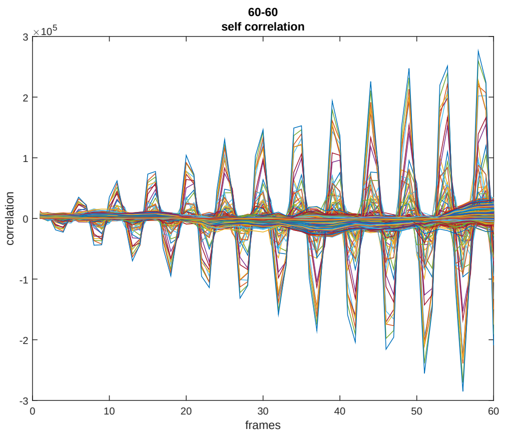
(a) Bright environment (b) Dark environment

Figure 3.4: Brightness values for each grid cell. This illustrates how much easier it is to detect and decode in a dark environment. Note that the values are considerably smaller in darkness.

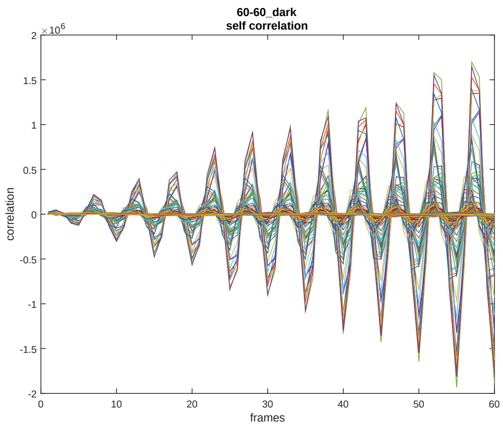
Figure 3.4 shows the brightness values for cells of size 30 × 30 pixels for a simple 60-60 blinking pattern in both a bright and a dark room. It clearly illustrates how much simpler detection is in a dark environment. After extracting the brightness values for each cell we want to find those with a high periodicity. To achieve this we use an autocorrelation approach which low-pass filters the zero-centered brightness values with the normalized values used as a template.[1] Equation 3.1 shows the formula we use to quantify the periodicity of the brightness in a grid cell. n ranges from 1 to $\#frames$ and $K = min(\#frames - 1, n - 1)$. Figure 3.5 illustrates the outcome of the filtering.

$$y(n) = \sum_{k=0}^{K} normalized\_x(k+1) \cdot zerocentered\_x(n-k) - \sum_{k=1}^{K} y(n-k) \quad (3.1)$$

Periodic cells peak higher than non-periodic ones so we can coarsely separate blinking from non-blinking cells by thresholding at a percentage of the maximum using the highest peak of a cell as a score. We discard all cells that do not reach 40% of the highest score over all cells. The threshold is determined empirically and is justified by the fact that we assume the video to be stable and thus the cells which do not contain any blinking only vary minimally which leads to little correlation whereas the blinking and its reflections correlate much stronger. So the threshold of 40% basically separates signal from noise leaving us with a handful of cells either showing the blinking directly or a reflection of it. Figure 3.6 shows the 30 × 30 pixel cells that passed the filtering.



(a) Bright environment



(b) Dark environment

Figure 3.5: Filtering of the brightness values for each cell. The higher the value the more periodic is the brightness in the cell.

[1]This corresponds to the Matlab function `filter` [9] whose implementation is discussed in [10]

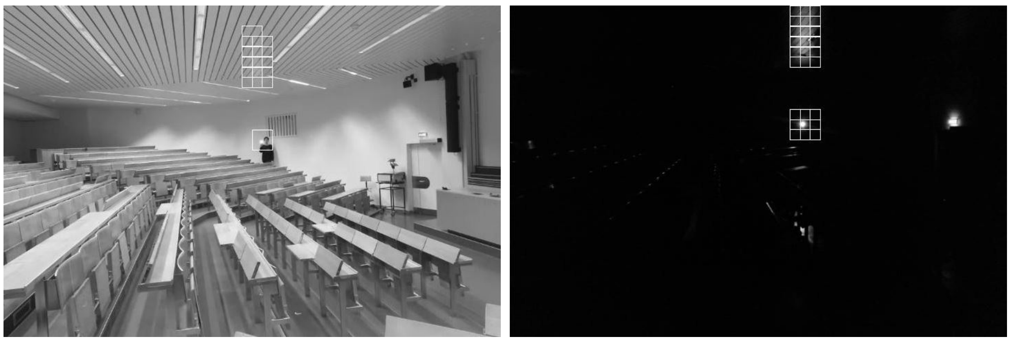


Figure 3.6: Blinking detection including reflections on the ceiling.

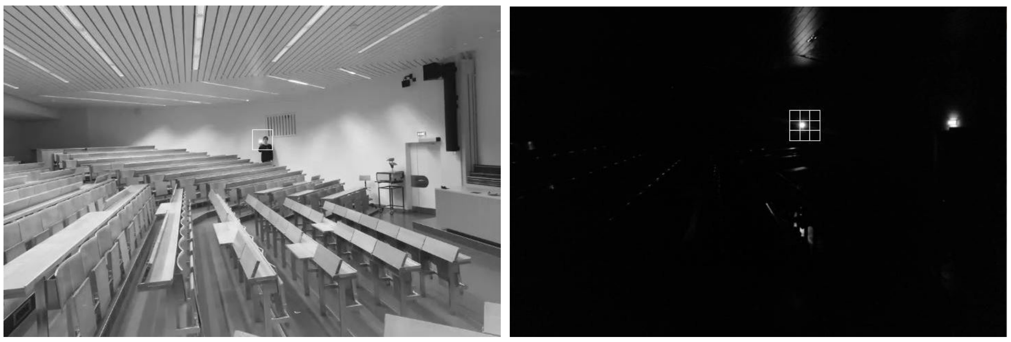


Figure 3.7: Blinking detection with the reflections on the ceiling removed.

For decoding, a reflecting cell may be sufficient but we also want to locate the sender so further filtering has to be done. When comparing reflecting with blinking cells we notice that reflections most often occur on flat specular surfaces such as ceilings or floors. Reflections on flat surfaces are spacious and often completely fill the cell in contrast to the cell with the blinking flashlight which shows a bright spot surrounded by darkness. In other words, the contrast of a blinking cell is high whereas for reflections it is often low. While this reasoning is not completely foolproof and might be invalid under special circumstances, it turns out to be a good heuristic. As a contrast score of a cell we take the standard deviation over the pixel values of a single frame where the flashlight is on. To guarantee that we have such a frame we compute the standard deviation for sufficiently many consequent frames and take the maximum. Figure 3.7 shows the result of the filtering step thresholding at 90%. For decoding we take the cell which scores the highest. A summary of the detection algorithm can be found in Figure 3.8.

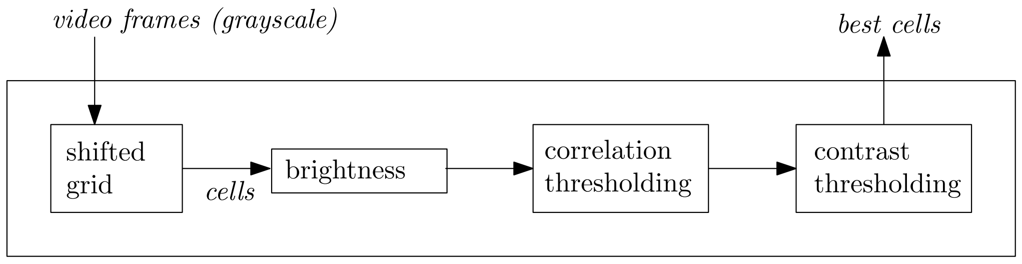


Figure 3.8: Detecting blinking patterns in a video.

3.3 System Components

3.3.1 Flashlight

A potential lower-bound on the rate at which we can transmit data over a visible channel is the rate at which the phone's flashlight can be toggled. Android provides us with two different ways of scheduling periodic tasks; *Handlers* (using the *postDelayed* method) and *Timers*. We implement both and measure their performance by logging timestamps during execution. Figure 3.9 shows the results for an 80-80 pattern. We can clearly see that the postDelayed approach lacks both in precision and in accuracy, while it also shows large periodic peaks, which we assume occur due to interference from the garbage collector. The Timer on the other hand appears to be much more reliable. The reason for this is that it produces less overhead of creating or recycling Java objects and that it does not run on the UI-thread, which is typically busy with updating the screen and handling user input. As a result we rely on using Timers for the rest of this project.

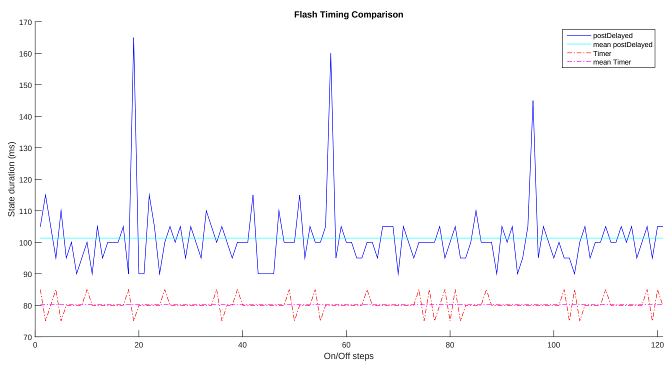


Figure 3.9: Comparison between Handler and Timer for an 80-80 flashing pattern. The Y-axis indicates the time that passed between two steps in the pattern, where 80ms is the ideal duration.

To see how fast we can actually toggle the flashlight we use the high-speed camera of an iPhone 6 (240fps) to record blinking patterns of different interval length. From these recordings we conclude that we can turn the flashlight on and off at intervals of 10ms, while 5ms seems to be too short to still be accurate enough.

3.3.2 Camera

According to their specification all the smartphone cameras we looked at support frame rates of up to 30fps. To verify this number we have the camera record the top of a record player running at 45 revolutions per minute on which we place a white marker. By counting the number of frames it takes the marker to cycles around the center three times and dividing it by 4s $(3 * (60s/45))$ we are able to infer the actual frame rate and can verify that it is according to specificiation, even in low-lighting conditions.

Unfortunately Android does not provide a way to handle a video stream from the camera at 30fps in a frame-by-frame manner.[2] Furthermore, there is no support to read a video file frame-by-frame,[3] which makes it impossible for us to first record to a video file, then process it. This leaves us with two alternatives in which we can record frames and process them afterwards:

PreviewCallback
When displaying the camera preview in an application we have the option

[2]Reading a video stream frame-by-frame may be possible with Android L's new *Camera2 API*, though we are not sure whether it would operate at 30fps. Since Android L is still fairly new and not widely deployed at the time of this writing we decided to stick with the old *Camera API*.

[3]Reading video files would require us to compile the *FFMPEG* library for Android.

to register a callback method that gets called every time a new frame is displayed. The first parameter to the callback contains the frame encoded in YUV-format as a byte array. For our purpose this is a convenient format since we only need the brightness information (and not the colors) which is stored in the first $width \times height$ bytes of the YUV-frame. From experiments we know that this approach typically reaches frame rates of about 20fps, depending on lighting conditions.

OpenCV's CvCameraViewListener2

The *OpenCV library for Android* [11] allows us to implement its *CvCameraViewListener2* interface, which features a method that is called for every new frame from the camera. We can use it to store the frames in memory in the form of a (grayscale) pixel matrix. The performance with this approach seems to be similar to the previous one, reaching around 20fps.

To summarize, we cannot benefit from the camera's 30fps. Instead we have to fall back to using lower frame rates, which dictates a new lower-bound on the achievable throughput of our channel.

3.4 Prototype

To show the correctness of our algorithm we build a prototype application as a proof of concept and evaluate it.

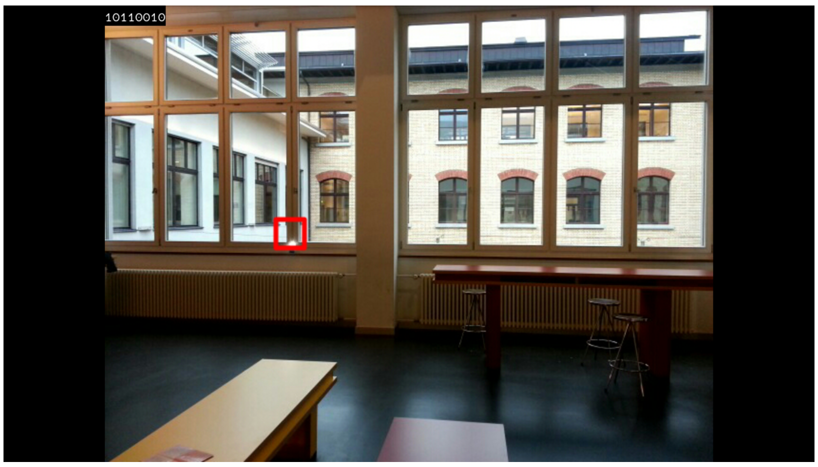


Figure 3.10: Screenshot of the prototype application in action showing the result screen. The decoded message is shown in the top left corner.

3.4.1 Implementation

Our prototype implements both the detection and decoding part of the algorithm on Android. We use OpenCV to access the video frames and also use the included libraries with native support to do fast matrix and vector operations. The user can send binary messages of up to eight bits in a unidirectional channel. The receiving part records frames for a fixed amount of time long enough to capture a complete message of maximum length (8 bits) and starts processing. After processing, a single frame is displayed with the best cell marked and the decoded message shown. The frames are kept in memory as grayscale matrices in 480p resolution. The processing includes all the steps described in Section 3.2. Most of the computation time is due to the correlation filtering step, which is the most costly part and has to be applied to every grid cell. We improve the performance by parallelizing this step, which speeds it up by roughly one third. Using larger cells would improve performance even further but reduces the precision, leading to an undesirable tradeoff. Figure 3.10 shows the processing and result screen.

3.4.2 Evaluation

To evaluate the prototype we fixed two phones (the Galaxy Note 2 for blinking and the Galaxy S4 for recording) on tripods and took them out into the field. We used three different blinking patterns (`00000000`, `11111111`, and `10110010`), each of which we repeated 5 times, and tested how well the prototype could detect and decode them. We did this both during the day (cloudy sky) and during the night. Since we could not observe any significant differences between the three patterns we group them together into 15 repetitions per distance and lighting. The results are shown in Table 3.1.

Table 3.1: Success rate of detecting, i.e., locating the flashlight, and decoding blinking patterns, produced by the Galaxy Note 2 and recorded by the Galaxy S4, at different distances and in different lighting conditions. Three different patterns, each repeated five times, were used for each experiment.

	5m		**15m**		**30m**	
	Detect	Decode	Detect	Decode	Detect	Decode
Night	100%	100%	100%	93%	100%	100%
Day	100%	93%	60%	50%	53%	40%

As we can see the prototype performs very well in a dark environment, both in detecting and decoding the message. In a bright environment, however, the rate of success quickly drops with increasing distance, making it practically unusable beyond 15m. The reason for this behavior is that the grid size is too large and as such there is too little variation in brightness within the blinking cell for it to be detectable. With a darker background this variation is much higher, making detection easier. An additional effect that improves the detection rate at night is the fact that the camera's auto-focus does not work properly, which causes the blinking light to appear larger. As a result it better matches the grid size and detection becomes easier.

Some additional tests (with the same setup) showed that during the night the prototype still works at a distance of 130m with a success rate of 100% for detection and 60% for decoding.

The average processing time, in addition to the 8 seconds of recording, was 18.5s, which is, unfortunately, much too slow to be practical.

3.5 End-user Application

The prototype works well as a proof of concept, but is not very user friendly. The dependency on OpenCV, which requires the user to install an additional third-party application, the slow processing speed and the lack of a duplex channel render the prototype useless in practice. We come up with the new application `BlinkEmote`, addressing all of these issues and introducing an entertaining emoticon chat for the flashlight-to-camera channel.

3.5.1 Implementation

As a first step we drop OpenCV completely for two reasons. On the one hand because we do not want our application to rely on a third-party manager and on the other hand we encountered problems with OpenCV on certain devices causing the prototype to run stable only on Samsung devices. Instead we get the frames directly from Android using the camera preview and converting from YUV-format to grayscale. To shorten the processing time we drop the detection part and focus on decoding only. Detection is now left to the user who has to align the camera with help of crosslines at the center of the screen such that it points directly towards the blinking flashlight. The user may also adapt the size of the crosslines with an intuitive pinch-and-zoom gesture. We use this feature for user-assisted cell size optimization. All the user has to do is to choose an appropriate grid size and keep the blinking flashlight within the cell for a few seconds.

To assist the user in keeping the image stable, we use Android's built-in video stabilization and further enhance it with gyroscope data to compensate shake by moving the crosslines in the opposite direction. The gyroscope support can be a little annoying due to improper calibration so we put it in as an optional feature, which can be toggled by tapping the screen. To enable duplex communication we merge the sending and receiving part into a single Activity and perform decoding

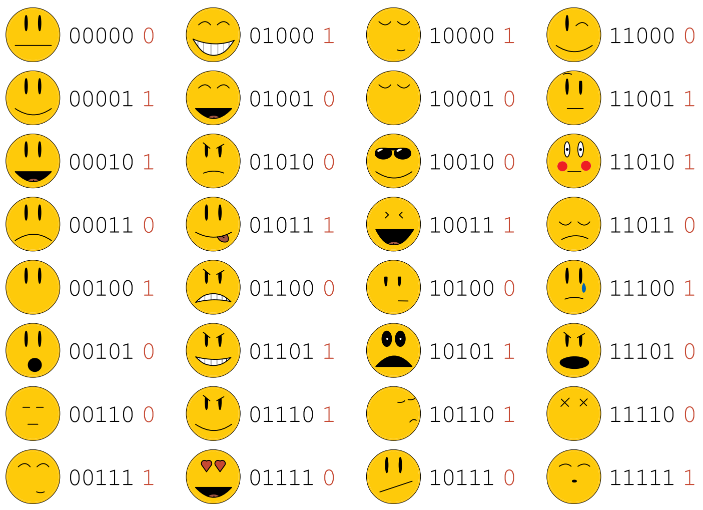


Figure 3.11: BlinkEmote features 32 emoticons that can be sent and received. Each of them is encoded as a 5-bit-pattern plus an even parity bit.

periodically in a background process. Instead of the raw bits the user is presented with a broad repertoire of emoticons, each of which is encoded as a 5-bit pattern plus an even parity bit to avoid false-positives (see Figure 3.11). Figure 3.12 shows screenshots of the emoticon selection menu and the application in action.

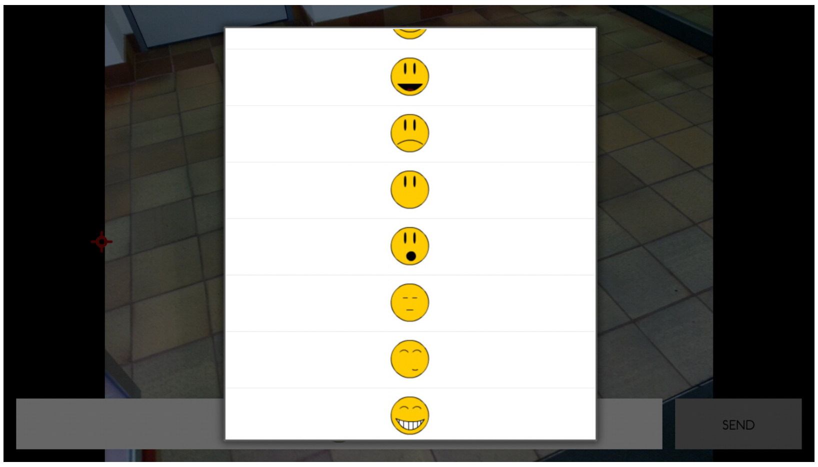
3.5.2 Evaluation

The application works well even in duplex mode and over some distance. The maximum distance depends on the lighting conditions and on how well the user can aim and stabilize the phone. Depending on the message we reach a data rate of 21 to 30 bits per minute with a latency of about 2 seconds.

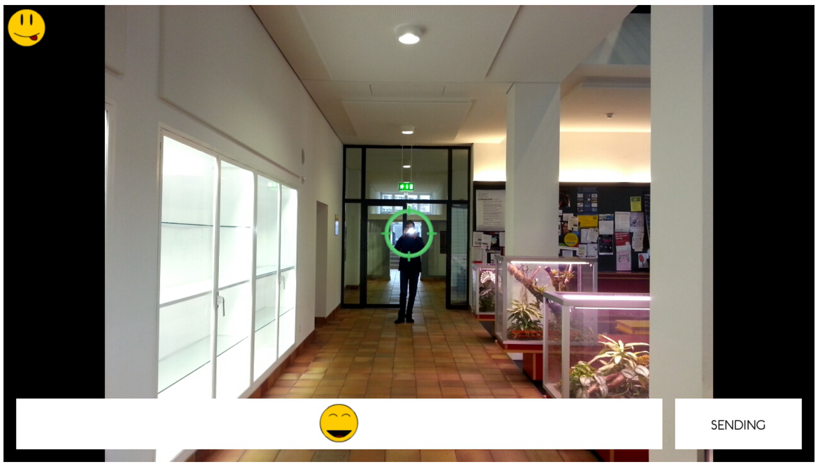
The additional anti-shake mechanics based on gyroscope are questionable and did only improve the handling on the HTC One whereas on all the Samsung phones the drift was too strong for it to be useful. While the direct usage of the application might not be evident the feedback was throughout positive and the application was considered to be fun. One major problem however is the bright flashing which gets annoying and could potentially damage the eye or might even trigger epileptic seizures. As a first countermeasure we added a warning message to inform the user upon first use of the application.

3.6 Conclusions

We showed that VLC can be implemented on modern smartphones in a way that is fast enough for end-users to send short messages over distances of a few



(a) Emoticon selection



(b) Sending and receiving emoticon

Figure 3.12: Screenshots of the end-user application `BlinkEmote` in action. The user is sending an emoticon (selected at the bottom) while simultaneously receiving one from his colleague (displayed in the top left corner).

meters, at a rate of about half a bit per second. We also showed an algorithm to locate a blinking light source in a video, but the performance of the typical smartphone is not high enough to do so in reasonably short time. The main limitations are currently the low frame rates of the camera as well as the lack of computation power to perform complex operations.

3.6.1 Future Work

Smartphone technology is rapidly improving, performance and cameras are getting better. The iPhone 6 for example features a camera with 240 frames per second which could drastically improve our data rate. The possibilities of Android's new Camera2 API are yet to be explored. Data rate could be further improved by using a more efficient encoding schema and optimizing the slot time. Looking deeper into the gyroscope stabilization may help to extend the distance at which the application could operate. Security is another aspect to be investigated, especially if flashlight communication is used outside the entertainment domain. Finally, one could investigate the use of infrared light instead of visible light to improve user acceptance, as we see more and more smartphones equipped with IR-hardware.

Bibliography

[1] Studer, A., Passaro, T., Bauer, L.: Don't Bump, Shake on It: The exploitation of a popular accelerometer-based smart phone exchange and its secure replacement. Technical Report CMU-CYLAB-11-011, CyLab, Carnegie Mellon University (Feb 2011)

[2] Hwang, I., Cho, J., Oh, S.: Privacy-aware communication for smartphones using vibration. In: Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on. (Aug 2012) 447–452

[3] : Wikipedia - Dial-up Internet access. http://en.wikipedia.org/wiki/Dial-up_Internet_access

[4] Frigg, R., Corbellini, G., Mangold, S., Gross, T.: Acoustic data transmission to collaborating smartphones - an experimental study. In: Wireless On-demand Network Systems and Services (WONS), 2014 11th Annual Conference on. (Apr 2014) 17–24

[5] Frigg, R., Gross, T.R., Mangold, S.: Multi-channel acoustic data transmission to ad-hoc mobile phone arrays. In: ACM SIGGRAPH 2013 Mobile. SIGGRAPH '13, ACM (2013) 20:1–20:1

[6] Galal, M., El Aziz, A., Fayed, H., Aly, M.: Employing smartphones xenon flashlight for mobile payment. In: Multi-Conference on Systems, Signals Devices (SSD), 2014 11th International. (Feb 2014) 1–5

[7] Galal, M., Fayed, H., El Aziz, A., Aly, M.: Smartphones for payments and withdrawals utilizing embedded led flashlight for high speed data transmission. In: Computational Intelligence, Communication Systems and Networks (CICSyN), 2013 Fifth International Conference on. (June 2013) 63–66

[8] Saxena, N., Ekberg, J.E., Kostiainen, K., Asokan, N.: Secure device pairing based on a visual channel. In: Security and Privacy, 2006 IEEE Symposium on. (May 2006) 6 pp.–313

[9] : Matlab Filter. http://ch.mathworks.com/help/matlab/ref/filter.html

[10] : Matlab Filter Implementation. https://ccrma.stanford.edu/~jos/fp/Matlab_Filter_Implementation.html

[11] : OpenCV for Android. http://opencv.org/platforms/android.html