ECE 597: Near-Ultrasonic communications for IoT Applications

Progress report. Summer Semester

Javier Gonzalo Cañada Toledo

A20429331

Contents

Abstract **3**
1. Introduction **4**
2. Background **5**
2.1. Ultrasound as information carrier 5
2.2. Existent commercial applications 5
2.3. Previous work in this project 6
2.3. Communication schemes for wireless communications 7
3. Proposed solutions **15**
3.1. Send and receive high-frequency acoustic signals on Android 15
3.2. Message exchange using ultrasonic signals on Android 18
3.3. Applications 23
4. Results and discussion **25**
4.1. Frequency analysis 25
4.2. Android App 25
5. Conclusions **29**
6. Future work **30**
6.1. Project continuation 30
6.2. Related lines of work .... **Error! Bookmark not defined.**
Appendix A: MATLAB codes .... Error! Bookmark not defined.
SoundTest function: .... **Error! Bookmark not defined.**
AudioProc function: .... **Error! Bookmark not defined.**
Appendix B: Android Codes .... Error! Bookmark not defined.
MainActivity java code .... **Error! Bookmark not defined.**
Activity_main layout file .... **Error! Bookmark not defined.**
Android manifest .... **Error! Bookmark not defined.**
Build gradle .... **Error! Bookmark not defined.**
References **32**

Abstract

This report presents an overview of all the research done for the course ECE597 during the Summer semester. This project is a continuation for the work already developed in this same course during the past semester. The main goal was to explore the possibilities of using smartphones and other low power IoT (Internet of Things) devices for using ultrasound communications and implementing our own cross-platform solution.

The project was developed in collaboration with other students, in this semester with Yann Hornych, A20432647. This report only focuses on my personal input to the project, briefly discussing the progress made by my colleague and his goals. I worked in the development of three Android applications, being the last one's infrastructure carried out by Yann, while I added some modules and collaborated fixing some bugs. In addition to this, he developed several other prototypes that can be found in his report.

Using ultrasound or near ultrasound for wireless communications between phones stands as a state-of-the-art alternative, that may present several advantages over conventional wireless technologies. However, there are several obstacles to overcome, being among them the low rates achievable by using near ultrasonic carriers and the high interaction of the acoustic waves with the surroundings.

To test the global feasibility of the project and to explore the potential of commercial smartphones, we developed a simple Android app that sends and receives acoustic waves of a variable frequency. The maximum frequency that we can use for this transmission was found to be 22500 Hz, leaving us a range from 18000 Hz to 22500 Hz for carrying out the communication.

With this experiment we also found out that the maximum available sample rate for managing audio on most smartphones is 192000 Hz. However, this value depends on the device's model hardware, and it was discovered that the application was not suited for old Android phones (5 years or more), that usually handle only sample rates of up to 44800 Hz.

Following this experiment, we implemented a first version for a messaging application that allows the user to send and receive fixed length messages through ultrasound, using binary amplitude modulation for encoding them. It allowed the user to define a message length and then send the character chain, encoded on binary ASCII for the other end to receive it and encoded. We achieved to send some short messages with a success ratio of around 50% for close-by devices. The system can send a 5-bit message in approximately 4 seconds.

Last, we developed a second prototype of this application that fixed some of the accuracy problems found in the previous one, with the downside of not being able to process message sending and receiving at the same time. The main improvements added to this second platform were: an error correction encoding, using 7,4 Hamming coding; an automatic gain control module; dynamic buffers handling that allowed for messages of variable length; and a more robust and reliable message header structure. The precision achieved by this was of a 90%, with a speed of 0.5 characters per second.

1. Introduction

The final goal of this project is to analyze the possibilities in using ultrasonic acoustic signals for sending information in Internet-of-Things (IoT) networks. This kind of communication method presents several advantages over conventional electromagnetic ones, such as providing a secure communication channel between close-by devices or being able to work in scenarios in which electromagnetic signals are not suited, for example, inside an ongoing airplane or for aquatic environments.

However, using acoustic waves as the carrier signals presents several complex challenges that must be addressed for achieving an efficient and robust communication scheme. Within this context, we want to decide if the limited hardware and software available in low power devices and smartphones are suitable for this task.

Whereas the past semester our work was aimed for a general study on the suitability of these devices for this task, this second part is mainly focused on the use of smartphones, and how to carry out this communication. We have developed several Android applications to send and receive ultrasonic waves using different modulation techniques, to analyze the performance of the device when processing this kind of signals and we have started the development of a more complex system that would allow us to send full messages.

Like in the first half of this project, I was working as a part of a team. In this case, the student Yann Hornych was the other member working on this task. This report only focuses on my personal contribution, only briefly mentioning some of the tasks he did in order to understand the full scope of this project. I strongly encourage to read his report in addition to mine, since his work works as an extension to mine, using my prototypes as a template to develop more complex and advanced applications.

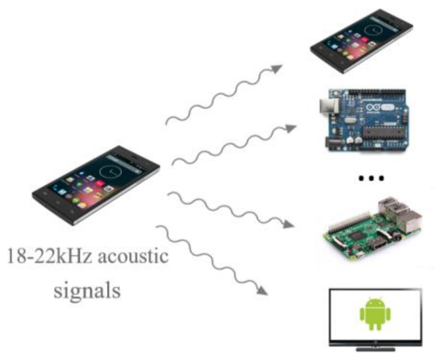


Figure 1: IoT network using near-ultrasonic signals

The report is divided as follows:

- In section 2 we briefly discuss about the motivation behind this project, the results obtained from our previous work and the tools that were used for this second part.
- In section 3 the methodology that was implemented is described, addressing the code that was used, the problems faced and how they were overcome. Additionally, we discuss some possible final applications, this technology could be used for.
- In section 4 we present the results obtained from all of our experiments, analyzing them in order to check their validity and the compliance with the main objectives of the project.
- In section 5 we can find the conclusions obtained from the results obtained and the overall progress of the project.
- In section 6 some future lines of work are proposed, in order to both improve the solutions developed so far and to point out some other ones that could be explored to advance within this field.

2. Background

2.1. Ultrasound as information carrier

As it was previously stated, the use of near-ultrasound acoustic signals in communication schemes for IoT networks could present several advantages over conventional wireless connections as Bluetooth or Wi-Fi [1] [2]. First, the establishment of the communication itself is faster and simpler, broadcasting the signal right through the emitting speaker and being received by the microphones used by the rest of devices, with no need of previous synchronization or other complicated setup procedures.

Second, it stands out as a more constrained and secure communication channel. The great attenuation factor experienced by acoustic signals in their interaction with the surroundings can both be seen as a disadvantage and an advantage, allowing users to restrict the distance within other devices can receive the message they are sending.

And last, the use of ultrasounds stands as the only alternative in some situations. One of these would be underwater communications, a field in within ultrasound signals are widely used nowadays both for communication and sensing applications [3]. Electromagnetic waves can't be efficiently transmitted through liquid water and therefore, ultrasounds appear to be the only option available [4]. Other case scenario would be the communication between devices inside an airplane in mid-flight. While electromagnetic waves could cause interference with the plane instrumentation, non-audible acoustic signals could be a great alternative for this situation.

Nevertheless, to deal with this type of communication carries several problems among them is the high interaction of the acoustic waves with the environment, which causes several phenomena like reverberation, echoes, or overall greater attenuation in the carrier, that are certainly detrimental in the transmission process [5] [6]. These facts lead to the need of a very error prone system, in which the signal processing must take into account very complex exceptions and be able to filter all the noise and adapt the received signals correctly.

2.2. Existent commercial applications

In the previous report we included an in-depth analysis of several existent solutions that work with ultrasonic communications for smartphones. We have included a list below of all the alternatives that were considered and a brief description of their functioning and their advantages or disadvantages.

Google Nearby

Google Nearby is an SDK (Software Development Kit) used in the vast majority of Android devices in the actual market [7]. It uses near-ultrasonic signals to link devices together in the first steps of its communication process, allowing the users to send and receive the actual message after this step by electromagnetic wireless methods as Bluetooth or Wi-Fi. The ultrasonic communication uses DSSS modulation, with a carrier frequency of around 18 kHz. We will talk about this modulation technique on section 2.3.

Its main advantage is that is a robust and fully developed tool, easy to use and with full support from their developer platform. It was created with the aim of allowing other Android programmers to use it and as so, it would be useful in avoiding the programming of too low-level subsystems for this type of project.

However, as it was stated, the main goal of the ultrasonic communication is to send short messages to connect the devices, not supporting the transmission of full messages. The reason behind it can be found in the low speed of near-ultrasound in comparison to electromagnetic carriers. Ultrasonic communication was chosen by Google due to its simplicity and its broadcasting properties, but it was discarded for actual communications due to these speed constraints.

Chirp

Another third-party SDK that was considered for developing this project was Chirp [8]. Chirp offers an acoustic communication scheme that is cross platform and user-friendly. It allows the user to select a "near-ultrasonic" mode, that works with 18kHz acoustic carriers.

In this case, the platform offered most of the functionalities that we needed, offering an efficient and relatively fast ultrasonic communication. We tested both on a Raspberry PI and smartphones, being able to send and receive short messages within an acceptable range.

This solution, however, presented some problems. First of all, some platforms are not yet fully supported, such as iOS or some Android versions, so it is not fully compatible with any device. When running the tests with the apps developed using this platform, we also detected in most cases audible noises produced while the message was broadcasted. In addition to all of this, their developers maintain most of the technical details secret, as well as many basic setup controls blocked (such as changing the carrier frequency apart from 3 fixed values).

Lisnr

Lisnr is another available platform that uses sound to transmit information for smartphones and other media [8]. Their developers promote it as a cross-platform specifically designed for carrying out secure data transactions. As so, the applications developed with it are aimed for being used for secure money transactions or sensitive information.

The main problem with this solution, was again that, like in many of the other ones, offers a final solution, protected by a commercial license. This make it impossible for us to repurpose their technology and develop our own applications. On top of that, it is also aimed for very short and constrained message exchange, which also makes it unsuitable for the objectives we are targeting.

2.3. Previous work in this project

The goal of this section is to offer a summary of the progress done for this project during the previous semester. In general, the work done so far was of more of a theoretical approach. Before jumping into trying to code our own platform from scratch, the efforts during this first half were mainly directed towards trying to adapt some of the existent solutions to our solution. After these were proven useless for our task in hand, we decided to move on into this last approach.

First, we focused on trying to adapt either Google Nearby or Chirp SDKs to our own system. We achieved some positive results with these platforms, being especially remarkable the possibilities opened by the use of Chirp. We achieved to communicate several phones as well as a computer running Linux using the libraries offered by Chirp in a very efficient way. The main problems that we faced were those that were pointed out in the previous chapter. The communication was not completely silent, and, because of the inevitable message encryption, we could not do much out of the basic functionality provided by the developers.

Apart from that, we also tested other tools as the cross-platform software defined radio platform GNURadio [9]. This system offered a very complete and powerful tool for developing this type of applications on Linux devices, but, unfortunately, it is not yet fully adapted to Android devices. For this reason, we decided to move on to other solutions.

Another task that was developed during this first segment was to run several experiments using MATLAB and a smartphone in order to define the limits of this kind of communication. We tried to analyze the frequency response of commercial speaker-microphones pairs to see if they are suited for this type of communications. These tests revealed that most microphones and speakers would handle with not too much attenuation transmitting sounds of up to 23 kHz approximately, before experiencing a noticeable drop in the amplitude of the received signal.

Finally, we started to encode what would be the first prototype of the Android application that is presented with this final part. We developed an app that allowed the user to transmit 22100 Hz square waves, and to receive them on the other end. The basic functioning and the layout used are very similar to the application that is explained in section 3.1., in which I present a new take on this problem, with a more versatile and complete application.

2.3. Communication schemes for wireless communications

Despite the particular properties inherent to acoustic communications, the overall communication process follows a similar pattern to any other wireless transmission scheme. The main difference would be that due to the interaction of the waves with the surroundings and the limitations of the transmitter and receiver, the parameters used, and some of the techniques would have to be changed for them to be suited.

A basic block diagram for a genera communication scheme can be found in Fig. 2. As we can see, there are two differentiated subgroups, the emitting part and the receiving part. Within the emitting part we include all the processing made to the signal before being transmitted into the channel. On the other hand, the receiving part deals with the processing of the data after is captured from the channel on the other end.

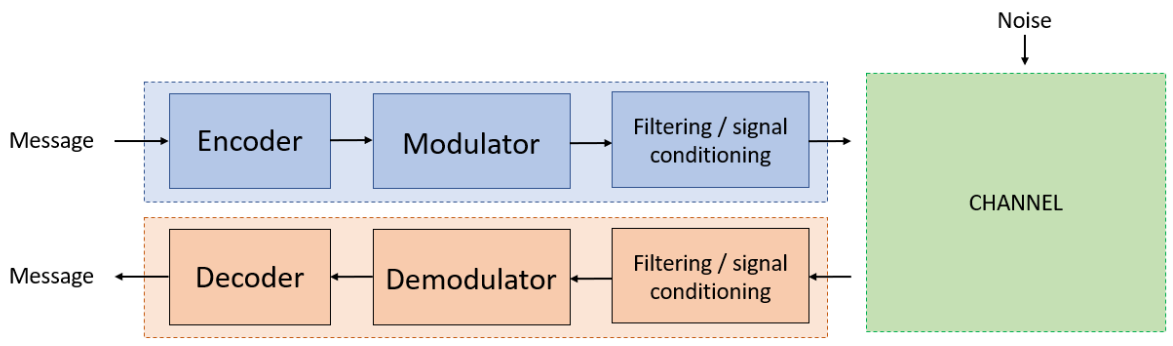

Noise
Message
Encoder
Modulator
Filtering / signal conditioning
CHANNEL
Message
Decoder
Demodulator
Filtering / signal conditioning

Figure 2: Wireless communication block diagram

In the following sections we will discuss in better detail each of those blocks, and how would they be for this particular type of data transmission. Most of the discussion will be focused on what would be the theoretically best solution, not the solutions that were finally implemented. As so, this section serves not only to better understand the problem to fix, but also to set the final goals for this project or similar ones.

2.3.1 Modulation

Modulation refers to the way in which the message is transmitted through the carrier signal. It is a vital block in any communication scheme, since it relates to how the data will be corrupted by the noise present in the channel and how robust the signal would be to errors and information loss. New modulation techniques are proposed every day and finding the optimal technique for a particular application is not a trivial problem. For this reason, we can find countless studies on modulation on its own for acoustic data transmission: [10][11].

In general, and within the context of this project we will only analyze the most basic modulation techniques, that are the following:

Amplitude Shift Keying (ASK)

ASK transforms the value of the symbol into a specific amplitude level, using a fixed frequency carrier. It is the simplest modulation technique, since the modulation would only require multiplying the carrier by a gain, that is set by the symbol. In a binary scheme (B-ASK) for example, we could send the wave when we want to send a ‘1’, and not sending the wave when we want to express a ‘0’. The demodulation would require analyzing the amplitude of the wave at that frequency and comparing it to certain thresholds.

The general expression for this kind of modulation can be seen in equation (1). In this expression, x_{BB} would be the baseband component, in this case a variable that sets the total amplitude of the signal.

$$x_{ASK}(t) = x_{BB}(t) * \sin(w_C t + \varphi_0) \qquad (1)$$

For demodulating the signal on the end there are many techniques that may be used. The first would be to filter the signal to only get the component at the particular frequency we are dealing with, and then simply analyze the peak values or the root-mean square values of the filtered part. Other approach would be to perform a Fast Fourier Transform analysis of the wave and see the gain of that frequency component. And last of all, we could just perform a low-pass filtering of the received data, obtaining the symbol sequence at the symbol rate.

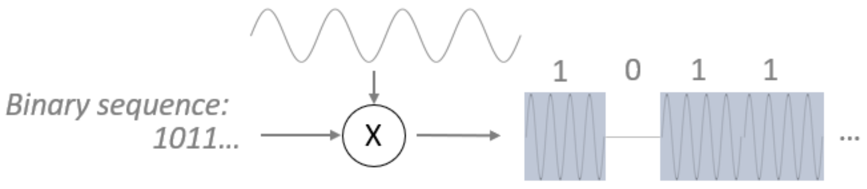


Figure 3: ASK modulation

Frequency Shift Keying (FSK)

The technique known as FSK consists in encoding each symbol by a specific frequency band, while the signal amplitude remains constant. The expression for this type of communication can be found in equation (2). In this expression, the baseband component x_{BB} sets the frequency of the transmitted way.

$$x_{FSK}(t) = \sin(w_C t + \int_{-\infty}^{t} x_{BB}(t)\, dt) \qquad (2)$$

As it seems obvious, the process to encode and decode the signal in this scheme is not as straight forward as it happened with ASK. For encoding we must use an efficient method to switch into the different frequencies with enough speed. On the other end, we have to find two different frequency components for the demodulation. This leads to a more complex logic, having to either filter the wave twice, or comparing it against two different tables.

FSK is, in general, more robust and reliable than amplitude modulation, not being affected as much by the attenuation in the channel or transient noises (we can assume that noise would not affect the frequency of the signal as much as its amplitude). However, there are other effects to keep in mind, in specific, the Doppler effect, when the two communication nodes are moving in relation to each other, has to be addressed, because it could lead to false frequency transitions, and therefore, errors in the message content.

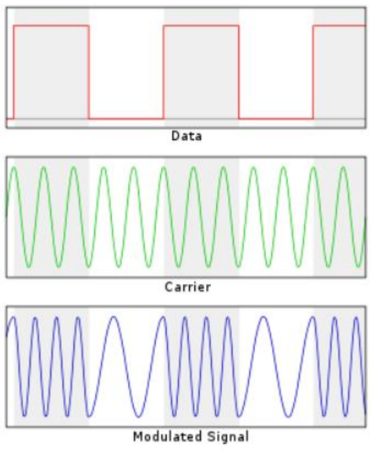


Figure 4: FSK modulation

Phase Shift Keying (PSK)

Another popular modulation method, widely used in electromagnetic communications, is phase modulation. In PSK we define a different phase for the wave representing each symbol, as it can be seen in Fig. 5. The general equation that sets the emitted signal would be equation (3), in which the baseband component sets, in this case, the phase of the wave, while not altering either the frequency or the amplitude of the signal.

$$x_{PSK}(t) = \sin(w_C t + x_{BB}(t)) \qquad (3)$$

PSK needs a perfect synchronization between emitter and receiver, to check if the phase of the wave was changed from the previous value, and most importantly, if the actual state represents a particular symbol or another. As so, it is not suitable for acoustic communications, in which the low transmission rates don't allow us to perfectly synchronize both ends.

The definition of PSK processes is usually depicted on a constellation diagram [12]. This diagram shows the phase and amplitude values for every symbol, being the phase, the angle in relation to the x axis, and the amplitude, the distance to the center of the plot. Notice that ASK can also be represented by this kind of diagrams, with points that has the same angle but different distance to the origin. In fact, it is very typical to find modulation schemes that use both amplitude and phase to define each symbol.

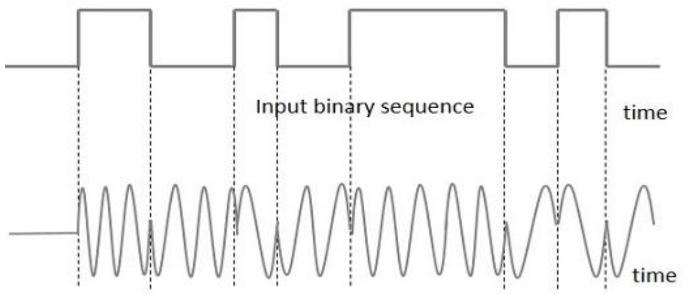


Figure 5: PSK modulation

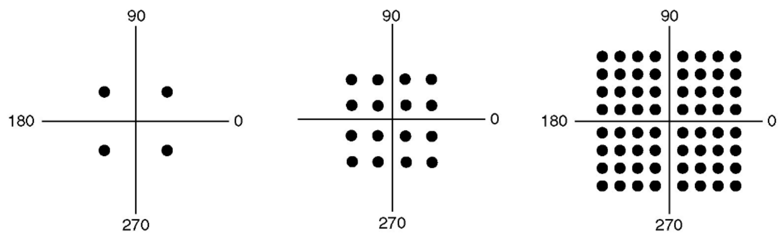


Figure 6: Examples of constellation maps for a 4 symbols scheme (left), 16 symbols (center) and 64 (right)

Pulse Width Modulation (PWM)

Modulating the pulse width of a fixed-frequency periodic pulse signal would also be an alternative for this type of systems. In this solution, that is depicted on Fig. 7, we change the duty cycle of the wave to encode a certain symbol.

The problem with this solution is that we need of a very precise and fast module to decode it, since we have to calculate this pulse width at a notably higher rate than the carrier frequency. In hardware solutions, or by using external timers, this should not be a problem. But with the system we are dealing with here, that consists in the use of high-level libraries on top of the Android operating system, this would be unfeasible.

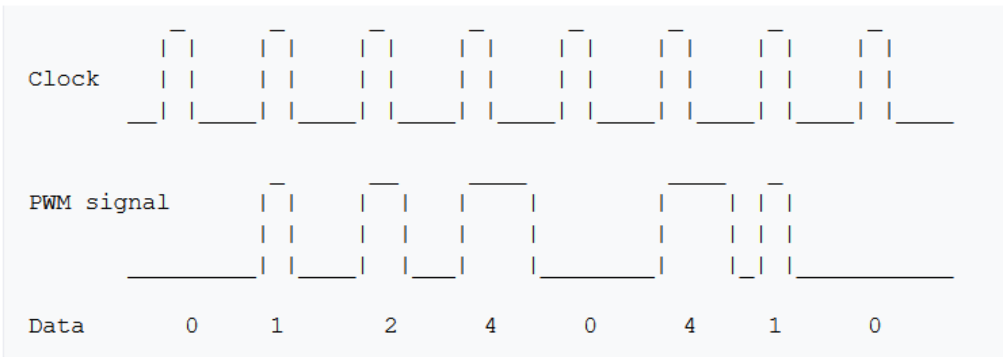


Figure 7: Pulse width modulation for a 5-symbol encoding scheme

Spread spectrum modulation

Other more advanced modulation techniques widely used in the industry nowadays are spread-spectrum methods. Although, explaining them in detail would be out of the scope of this project, they are highly important, duo to their popularity and their positive properties in terms of error robustness and resistance against transmission interception by attackers.

Spread-spectrum techniques modulate the signal in a way that they drastically increment the bandwidth of the data sent. This way, we disperse more the data into the frequency spectrum, making it more difficult to lose a large amount of data in the transmission, since it travels through many different frequency bands.

The most important spread spectrum techniques are: Frequency-hopping spread spectrum (FHSS) [13], direct-sequence spread spectrum (DSSS) [14], time-hopping spread spectrum (THSS) [15] and chirp spread spectrum (CSS) [16]. Among these, we will only discuss in detail the direct-sequence spread spectrum technique, due to its great suitability for acoustic communications [10].

DSSS multiplies the signal that must be transmitted by a pseudo random code, that spreads its data into multiple frequency paths [17]. This noise signal has a higher chip rate, what translates into a broader passband for the transmitted wave. Due to this fact, DSSS is very robust against interference, since any noise component would only affect the data chunk transmitted within that frequency band, not altering the rest of the data.

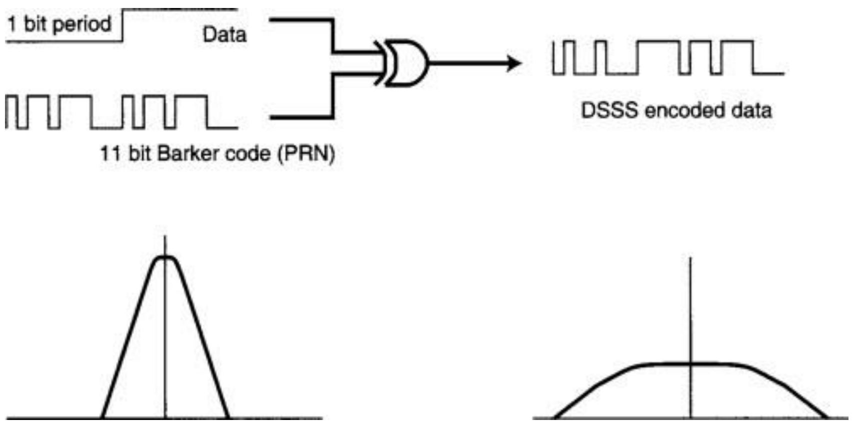


Figure 8: DSSS modulation

Modulation for acoustic communications

From the previously mentioned methods, we can directly discard phase shifting when using acoustic carriers. The impossibility to perfectly synchronize both receiver and transmitter makes it impossible to set a default phase to compare the variations to.

ASK, despite its simplicity, also stands as a poor choice for this kind of systems. Due to the great dependence of the signal amplitude on the interaction of the wave with the surrounding it would also be very tough to have a robust and efficient ASK system with acoustic carriers. To this fact we can also add the problems related to multipath propagation and reverberation, that would make the issue even more complex. Despite this, as it will be seen, ASK has been used for the first prototypes developed in this project, always taking into consideration the limitations of this technique.

Additionally, pulse modulation, although feasible in theory, in practice remains useless for these systems. The reason behind this is the high limitation on the sample rate we have for commercial hardware devices as smartphones. To correctly alter and, most importantly, detect these alterations, in the pulse width of a high frequency signal would require more precision than the one we seem to have (from our experiments we saw that the maximum achievable sample rate would be 192000Hz).

With this, we have two main alternatives for a robust solution. These would be FSK or spread spectrum techniques. Almost all the applications that are nowadays using acoustic communications use DSSS as the main modulation technique, for the reasons that were previously mentioned.

2.3.2 Encoding / decoding

Although we can also refer as encoding for the modulation part of the process, this section deals only with the process to transform the original message, for example, a list of characters, to an efficient set of symbols to be later modulated into the physical signal.

In general, we would transform every character from our chain of input information into the minimum possible set of symbols that we can modulate with our system. For example, if we have a Quadrature Phase Shift Keying modulator, that is, that modulates into 4 different phases with constant amplitude, we move from 256-bit ASCII characters to 256/4 = 64 symbols. This means it would take us 64 cycles to transmit each symbol.

More advanced encoding schemes try to decrease the number of symbols needed in order to increase the speed or the process. This would be the case of optimal bit allocation schemes, specially used in video communications. In these, we try to encode the most probable symbols with the lowest set of bits possible (or modulation symbols), being, on the contrary, the less probable ones coded with longer sequences.

Message encoding plays also a major role in error correction techniques, as it will be explained later. In general, we can encode the message in ways to make it less susceptible to noise or data loss during transmission. This can be achieved by encoding it in a way that it is easier to check if the symbol received was wrong (parity check), by sending redundant copies of the data, or by using prediction coding to know in advance if the next symbol received is likely to be altered.

2.3.3 Synchronization

Synchronization between emitter and receiver ends is vital for the receiver part to identify when there is a message available and its length. One of the most common methods to synchronize the process is to use synchronization signals. These signals are special headers or footers, that, added to the actual data to transmit would indicate the receiver when to start decoding. This process is part of the encoding block, but we have decided to treat it separately because of its importance.

The main features that any synchronization message should comply with is that it may be robust against bit loss, as simple and short as possible to avoid noticeable speed losses, and for it to be clearly distinguishable against the actual message symbols. In summary, it must be a distinctive short message that clearly indicates the beginning of the data exchange.

The use of footers can be avoided when the message length is fixed or when we encode this length value into the header. This last alternative is the most common one, being for example used in standard IP packets encoding for Wireless networks. As so, it is the solution that we will use for our system, as it will be further discussed.

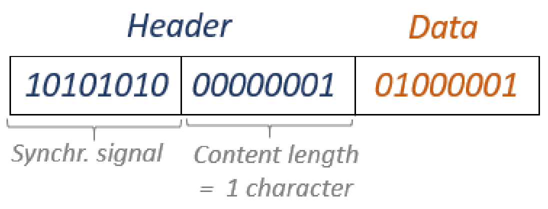


Figure 9: Data packet structure used for our project

2.3.4. Error correction

Another part of the encoding process would be to perform error correction. There are several approaches to detect and solve errors found in the transmitted data. In the context of this project we will briefly discuss the most widely used ones, and especially the ones that are suited for our particular application.

One solution would be to perform Multiple Description Coding [18]. This process consists on sending within the data packets redundant copies of the most vital information, such as headers. It is simple to implement and provides a robust solution that ensures that at least the main parts of the message are not lost. On the other hand, increasing the size of the packets to include redundant copies increases the latency of the system, significantly reducing the transmission rate.

Another solution would be the use of parity check encoding methods, such as Hamming coding [19]. With this technique, we encode the set of symbols into a new set which has maximum Hamming distance. This way, we can check, on the receiving side, if any of the bits of the symbol is wrong by calculating its Syndrome value. The number of bit errors detectable for each symbol depends on the size of the encoding and decoding matrices. As it is obvious, to be able to detect more variety of errors would lead to longer encoded messages, and, thus, slower communication.

And finally, there are also feedback methods, which can be complementary to the previous ones. Their principle is very simple, if the receiving side detects any errors it asks the sending half to send those parts or the whole message again. Although, any advanced methods have been developed in order to optimize this process, this method is by far the slowest from all the ones that were discussed. With the low transmission rate we have when using acoustic carriers, this method remains useless in comparison to others.

2.3.5. Transmitter / receiver

This block refers to the actual technology used for transforming the data into a physical signal. Depending on the carrier we would have different devices to perform this task. In electromagnetic communications, antennas are used, but in our case, for acoustic signals we make use of speakers and microphones.

Speaker and microphones both operate following the same principle. They make use of the piezoelectric effect to transform electric signals into physical vibrations or vice versa. It is important to point out that their denomination only depends on what are they were designed to be used for, although their components are very similar, and both can actually be used as transmitter or receiver.

There are several parameters that characterize a speaker or microphone: their maximum gain, power consumption, efficiency, level of noise… Among all of these, the most crucial one for our application would be the device's frequency response. The frequency response of a speaker or microphone refers to their behavior in terms of the amplitude of the signal produced/received in relation to the frequency of the sound to be transmitted.

In Fig. 10 we can see an example of the frequency response of a commercial microphone. As we can see there is a significant decrease in the gain for the signal received after a certain threshold, that corresponds to the limit of the audible spectrum. This is caused by the fact that these devices are usually optimized for being used in audio applications, for which ultrasonic signals would not be noticed and are so can be eliminated.

This frequency response tendency is common to most devices in the market, and it is a huge obstacle for our task in hand. As it was mentioned, some of the experiments developed during the previous semester were aimed for finding the exact limit after which communication would be impossible with commercial hardware. But this value is difficult to determine since every device has a different behavior, and, usually, smartphones manufacturers do not include thorough information about the technical specifications about these hardware parts. Again, this is caused by the indifference of the vast majority of their users to these details.

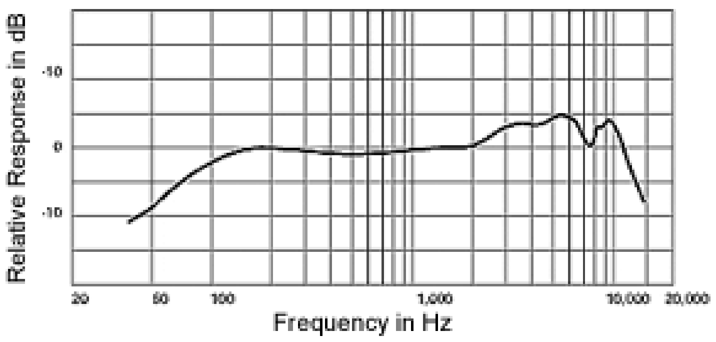


Figure 10: Frequency response of a commercial dynamic microphone model Shure SM58

2.3.6. Signal conditioning

The transmitted signal must be conditioned to optimize its transmission. This step would include the processes made in order to filter the noise out and to, in general, make more efficient the transmission, also including speed or gain adjustments.

In some applications, this step is used only to improve the quality of the received signal, but in most cases, it is a vital part of process, such as in the one we are dealing with. Since we are using acoustic noise for transmitting the information, environmental noise is an unavoidable factor to take into consideration. We need to make sure we are only processing the information transmitted by the carrier of an exact frequency range, in this case at around 20 kHz, ignoring the rest of components.

Apart from taking away noise, filtering can also be used for decreasing or avoiding Intersymbol Interference, or ISI. For this we can use pulse shaping filters, such as a Raised Cosine Filter, a Root-Raised Cosine Filter, or Matched filters. To discuss all the mathematical principles and the difference between these implementations would be out of the scope of this project. In general, they are low-pass filters, whose goal is to make the baseband frequency content closer to a rectangular shape, as it can be seen in Fig. 11. By doing this we avoid ISI after modulating this baseband component with the carrier.

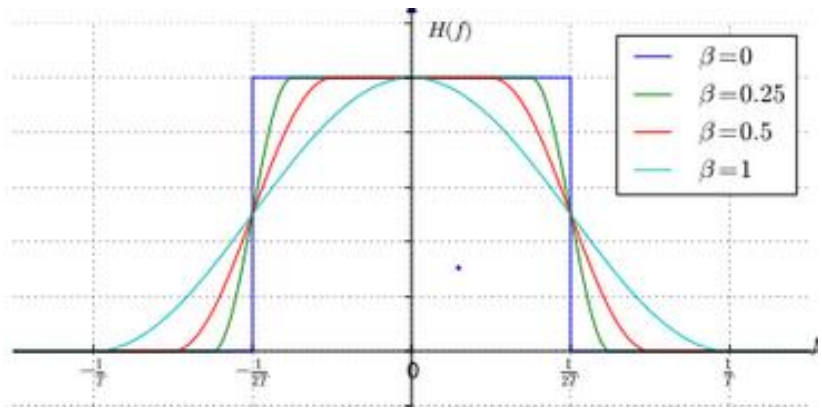


Figure 11: Impulse response variation when applying a Root Raised Cosine filter with different roll-off factors

The use of this kind of filters is crucial in most communication applications, and therefore, is a problem that we have to address in our solution. However, ISI only occurs when using wide spectrum baseband signals, which is optimal in terms of speed, but in our case, we can start by working with a very low symbol rate, so we avoid ISI without the need of any additional filtering.

2.3.7. Gain control

This task would usually be considered part of the signal conditioning. We have decided, however, to dedicate this brief section to discuss about it. Gain control ensures that the amplitude of the received signal has the right conditions to be decoded on the receiving side.

The most typical procedure to control the gain of the received data and to adapt it to the strength of the signal would be to use an Automatic Gain Control (AGC) module. An AGC is continuously calculating the amplitude value of the received signal (by RMS or peak values, for example) and comparing it to a reference. If the mean value of the received signal is too high, we adapt the decoding multiplying it by a lower gain constant. If it is too low, we increase the value of that constant.

The practical implementation of this block can be performed in many ways. One typical solution would be to use a PID control, that continuously tries to adapt the mean value to the desired threshold. The main difficulty found in this process is to be able to process in almost real time the received signal to calculate its mean amplitude. This may certainly be an obstacle with the software and hardware limitations we have with the devices used for this project.

3. Proposed solutions

In the following chapter, we discussed the solutions that were proposed and how they were implemented. We have mainly worked in the development of two Android applications, that were developed from scratch, using the libraries available for Android java programming.

The first one would serve as a first prototype to check the suitability of the phone and the Android OS for this task. It allows us to send and receive sinusoidal waves with near-ultrasound frequency and to display their waveform on screen.

From there we moved on into developing several messaging app prototypes, using different modulation techniques. The first approach was to develop a simple ASK scheme, that I fully developed on my own. After that we worked in developing some enhanced versions of it, in which both Yann Hornych and I collaborated. Finally, my colleague also developed a prototype for a system using frequency modulation. Other modulation techniques were considered, and certainly are recommended to be tested in a near future, but they would have been out of the scope for this project.

This section offers a wide overview of these systems and their functioning, without dealing in much depth with the actual code and the programming involved. We highly recommend consulting the Appendix present in this document, that includes the Java code as well as others Android programming files, to better understand the full software implementation.

3.1. Send and receive high-frequency acoustic signals on Android

This first application provides a simple interface for the user to send and receive, analyzing the results, acoustic signals. The signals produced are sinusoidal waves of variable frequency, that can be changed by the user at any moment. The complete Android code can be found in the Appendix 1, among the layout files used for designing the graphic interface.

This functioning was already targeted during the first half of this project, in which a much simpler app was developed. In addition to this, the other student working on this project, Yann Hornych also developed a more advanced version, in which the graphic display also offers information about the modulation and the encoding applied to the signals. I strongly encourage read his report to know more about this application.

The graphic interface remains almost the same as the one previously shown in my previous report, as it can be seen in Fig. 13. However, as it can be further read in that other document, this application was highly limited. It allowed the user only to use a maximum sample rate of 44100 Hz, to produce a square signal with fixed frequency of 20000 Hz.

The basic class block diagram is shown in Fig. 12. In this section we will discuss in more detail about these blocks. Most of the logic implemented for building this simple system was afterwards adapted for the more complex applications for sending messages, as so we won't get into that many details in the following sections.

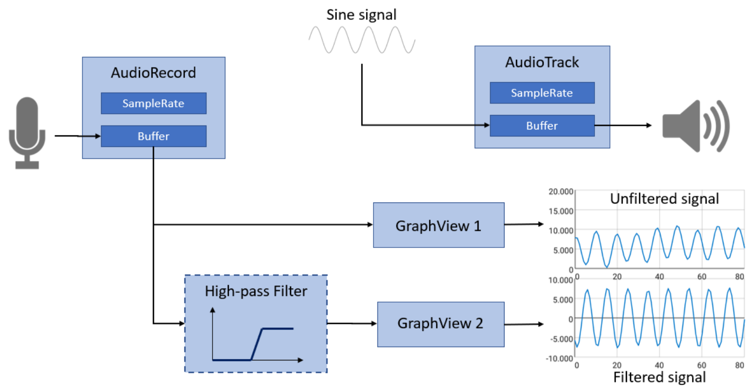


Figure 12: First application block diagram

3.1.1. Basic functioning

The app only uses one screen (one single activity) that is the one seen in Fig. 13. In this interface the user can use two buttons: SEND, that starts or stops generating the sinusoidal wave on the speaker, and REC, that starts and stops the recording through the microphone.

The text fields present to the right of each buttons display the state of the app, informing if it is producing sound, or recording, switching whenever the buttons are pressed. Bellow the buttons we can see two additional text contents, that display the sample rate for recording and playing the sound in the device used. It is important to point out that the maximum available sample rate depends on the particular device hardware, which justifies the high importance of displaying its value on screen.

Using an editable text field, the user can change the frequency of the wave produced, set by default to 20 kHz, the frequency that we were planning to use for our final application. This frequency is updated every time the system goes from not emitting sound to producing the sinusoidal wave. So, if a user changes the value of this field while the signal is being generated, they should not appreciate any change in its frequency, until they stop the speaker and start it again.

When the user presses the REC button to stop the recording, two graphs are automatically shown at the bottom of the screen. The top one shows a fragment of the recorded sound without any alterations. The bottom one displays the waveform obtained after filtering some of the noise of that audio clip.

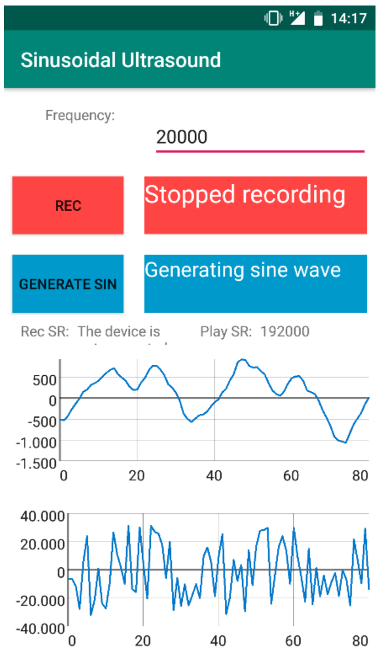


Figure 13: Application layout

3.1.2. Generating the acoustic signal

For handling the sound generation, we used the android library *AudioTrack.* The methods included to the class AudioTrack take a buffer, which contains the full signal that we want to send to the speaker module and play it on the speaker. This wave is played at the sampling rate that we configured the module to work at. Once we have played the whole buffer, it plays it again, until we send a stop signal.

Since what we want to send is in this case a 20kHz sine wave, we used a precomputed table containing a sinewave form for one full period. This table has 1024 elements, as it is obvious, using a bigger table would make the wave to be more precise (closing to a perfect sine). Increasing this array, would however increase the memory usage, so a balance must be found between accuracy and memory space. In this case, using 1024 values work with enough precision for our application and we cannot see any issues in terms of memory space.

For creating the buffer that would be played in the speaker, we go through the sine table, taking values separated by a certain step value, which depends on the frequency of the wave we want to generate. The expression used to calculate this increment is the following:

$$step = length_{table} * ({}^{frequency_{carrier}}/_{sample\ rate}) \qquad (4)$$

We precompute this buffer by selecting values distanced by this step, so when we play the buffer at the sample rate used, the sinusoidal wave would have the desired frequency. A problem faced when trying to solve this problem is the correct choice of the size of the waveform table and the buffer. If these two values are not correctly selected, we may experience undesired noises when the last buffer value played is in different phase to the first value that will be played in the buffer in the next repetition of it.

3.1.3. Recording the signal with the microphone

On the other hand, for managing the sound recording we use the library *AudioRecord.* The initialization for this module is similar to the one used with AudioTrack. In this case, the AudioRecord object reads the values from the microphone and puts them into a buffer that will be later processed. The sample rate used for the recording can be different from the playing frequency, and it depends on the hardware used for the device (audio codec).

It is important to point out that in our design both the recorder and the player are executed in different threads that are continuously executed, starting or stopping the processes by setting two different flags in the functions called when we press the buttons. For future implementations, both processes could be adapted to a single thread, allowing for a more efficient program and avoiding some possible multitasking errors.

3.1.4. Initialization of the system

When we start the application, it is important to make sure that all the parameters are well defined, and that the system is able to send and receive sounds. Furthermore, as we previously mentioned, the maximum sample rate that can be used for receiving or generating sound depends on the technical specifications of the device, so we have to determine these values before starting the communication.

For this matter, what we do is we run a loop in the code that goes through different sample rates for both the player and the recorder, ordered from high to low. We try to use these sample rate values for the transmission, and if we get an exception we move on to the next, lower value. Whenever we detect a suitable sample rate, we break the loop and we store that value, that we use to initialize the recorder or the player.

Within this initialization code I have also included a statement that checks if the maximum available sample rate is too low for having a successful communication. This way, we can avoid users to try the app on not suited smartphones.

3.1.5. Processing the signal

The values received on the recorder buffer are filtered by a 6-pole filter Butterworth high-pass filter, with a cutoff frequency of 19 kHz. This way we avoid most of the background noise that may be present in the moment of the data transmission. We did not consider it necessary to filter out higher frequency noise components, and this is because we are working close to the maximum frequency that can be detected by the microphone.

As so, even if we are getting high frequency noise on top of the signal, we are not actually sensing them with our microphone. Furthermore, if these noise signals are in fact within the microphone’s range, it would be extremely complicated to tell them apart from the carrier signal, ue to their closeness in terms of frequency.

3.1.6. Showing the graphs for the signal received

Once we received the signal, we displayed them in two graphs. The first one shows the raw data, received directly from the microphone buffer. The bottom one shows the signal once it is filtered by the high-pass filter and all the background noise is attenuated. These two graphs can be seen in more detail in Fig. 14. The x axis is measured in number of samples, taking the recorder sample rate.

The graphs were built using the open source android library contained in the GraphView package [20]. To create them we only had to send both data arrays as the parameters to display on two GraphView objects. In the applications that were later developed by my colleague Yann Hornych, he improved these graphs by making them scrollable and adding the feature to zoom in and out in them.

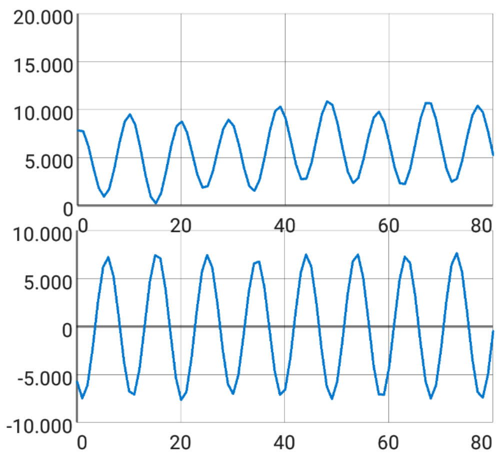


Figure 14: Graphs detail from the first applicationm

3.2. Message exchange using ultrasonic signals on Android

Once we corroborated that transmitting this kind of signals through smartphones was feasible and efficient, we started developing full ASK encoding applications. The first attempt, that I carried out on my own, allows the user to send and receive short messages of a fixed length in a highly limited way. After that, Yann Hornych designed an improved version, that was mainly coded by him, which presents a more robust and complete system. I helped him during this task, working in specific in some of its subsystems, such as error correction encoding.

Additionally, the other student has also developed the first prototype for a system that uses binary Frequency Shift Keying. For more information about this other solution, that presents greatly promising results check his report. As it was previously mentioned, we expect FSK to be more efficient and error prone than ASK for our final application, having tested only ASK so far due to its simplicity.

In this section, I will talk about those two first tries to develop an ASK scheme, talking in depth about the first one I designed in its entirety, and focusing only on my input made to the second one. It is important to point out, that the second, more advanced, application used the same basic functioning as the one I designed, so the main core would be very similar.

3.2.1. First prototype

This first version, which layout can be seen in Fig. 15, comprises a basic messaging app that allows the user to send fixed-length messages. This is an important constraint, since both the sender and receiver must agree prior hand on the size so they can encode and decode the messages properly.

In this case we have only one button at the top of the screen, to send the message. The recording is continuously been done in the background, except from when we are sending a message. This was done to simplify the problem, since the thread managed could not take all the incoming signal processing at the same time as the generating process. This limitation is not too much of a drawback, since it is highly unlikely that we receive a message in the exact moment we are trying to send one.

The modulation technique used is binary ASK, with 1's coded as a sine signal of the maximum amplitude possible and 0's encoded as the absence of the signal. This is not the most efficient way to carry this task out, as it will be discussed later. Nevertheless, it was fixed for future implementations as we will see in the next section. For the demodulation part, we have decided to use the Root Mean Square (RMS) value of the filtered received signal. If the RMS of the signal during the symbol cycle is greater than a threshold we determine that we received a '1', being a '0', if it is bellow that threshold.

The encoding is done by simply using the ASCII table binary values of every character we want to send. To this chain of binary symbols, we attach a header that is a series of 8 ones. Again, this is not the optimal solution, since we can easily get noise that could imitate this behavior, leading to false positives. However, this first approximation is precise enough for this first prototype, and other header configurations were tested in later solutions.

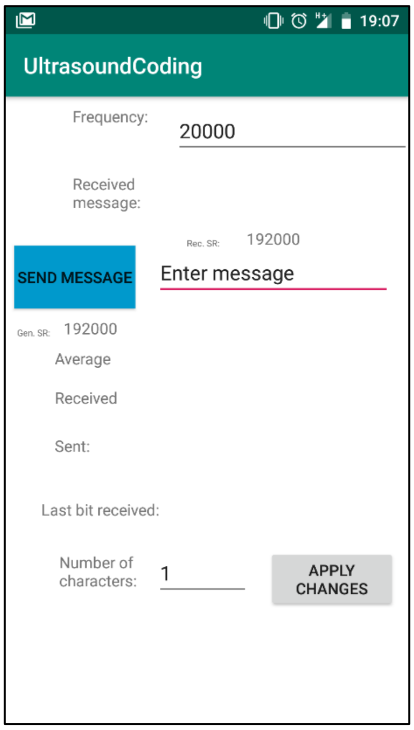
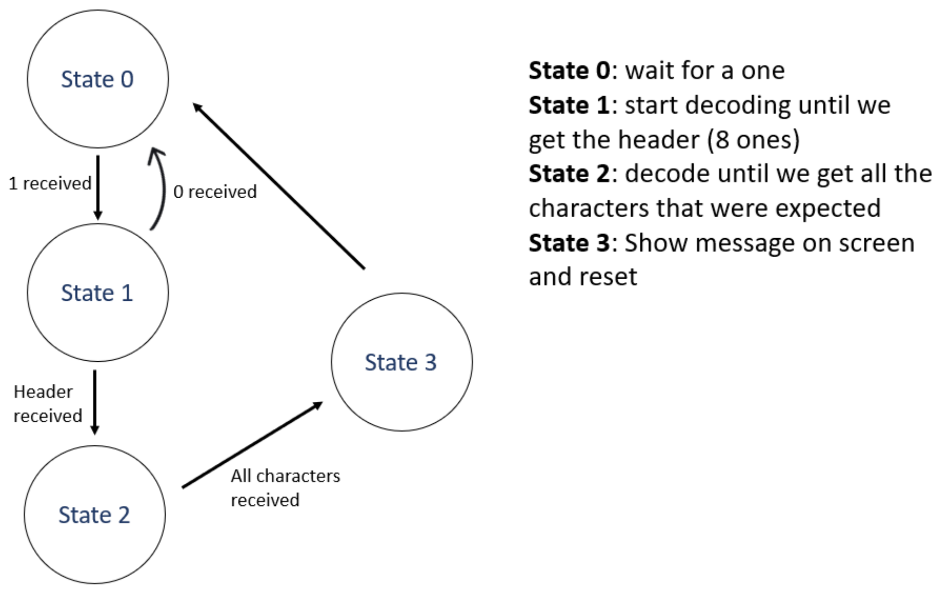


Figure 15: Main screen for the first messaging app developed

To carry out both the encoding and the decoding we use two state machines, that are represented in Fig. 16 and Fig. 17. For the encoding part we start by putting 8 bits into the playing buffer, that will be the header for the message. After the header was sent, we start encoding the characters introduced, until we reach the total length that was introduced as a parameter. We are using two different states one for sending the first character and other for the rest, the reason behind this is that every time we try to send a new character we have to convert it first into the binary representation and restart a temporary buffer that stores the bits needed. After all characters are sent, we kill all the processes related to generating the sound and we reset all the variables for starting to encode a new message.

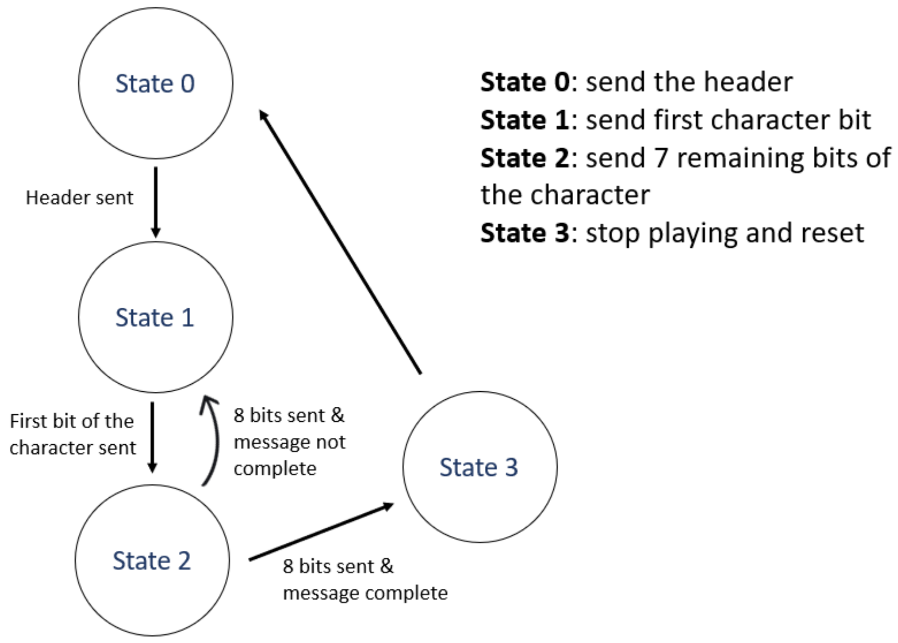
For decoding, on the other hand, we have the following states. In the initial state we would be reading the buffer and demodulating its content until we find a '1'. When we find a first '1' we move to state 1 and see if we get the whole header, formed by 8 ones on a row. If we get any '0' before obtaining the header we restart the process. Once we received the full header, we start decoding the rest of the message until we have all the characters we expected. The final state would only be used for resetting the system and going back to the initial state.

This state machines are evaluated periodically using a timer that runs at the targeted symbol rate. The period for this clock is 75 ms, which leads to a symbol rate of 13.3 Hz. Looking at these values it seems obvious that ISI will not be a problem, since 13.3 Hz is notably lower than the carrier frequency of 20000 Hz. For this reason, we did not consider implementing any additional filtering.



State 0: wait for a one
State 1: start decoding until we get the header (8 ones)
State 2: decode until we get all the characters that were expected
State 3: Show message on screen and reset

Figure 16: State diagram for the decoding process

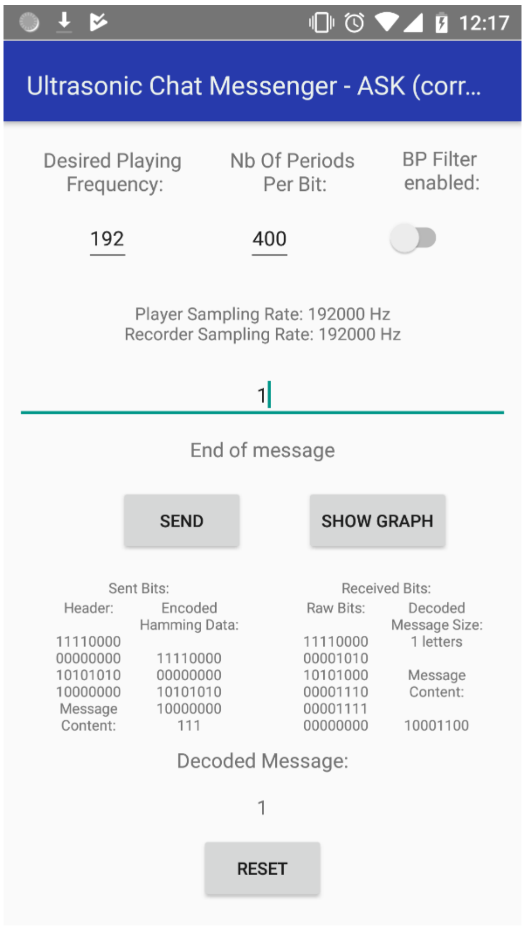

State 0
Header sent
State 1
First bit of the character sent
8 bits sent & message not complete
State 2
8 bits sent & message complete
State 3

State 0: send the header
State 1: send first character bit
State 2: send 7 remaining bits of the character
State 3: stop playing and reset

Figure 17: State diagram for the encoding process

3.2.2. Second prototype

After achieving such successful results from our first prototype, several interesting opportunities opened in front of us. Yann Hornych used the code developed by me on that first prototype to develop a yet more advanced messaging application for Android, in which I collaborated by fixing several bugs and adding some modules.


12:17

Ultrasonic Chat Messenger - ASK (corr...

Desired Playing Frequency:	Nb Of Periods Per Bit:	BP Filter enabled:
192	400	

Player Sampling Rate: 192000 Hz
Recorder Sampling Rate: 192000 Hz

1

End of message

SEND SHOW GRAPH

```
        Sent Bits:                         Received Bits:
Header:      Encoded               Raw Bits:     Decoded
             Hamming Data:                       Message Size:
11110000                           11110000      1 letters
00000000     11110000              00001010
10101010     00000000              10101000      Message
10000000     10101010              00001110      Content:
Message      10000000              00001111
Content:     111                   00000000      10001100
```

Decoded Message:

1

RESET

Figure 18: Main screen for the second messaging app

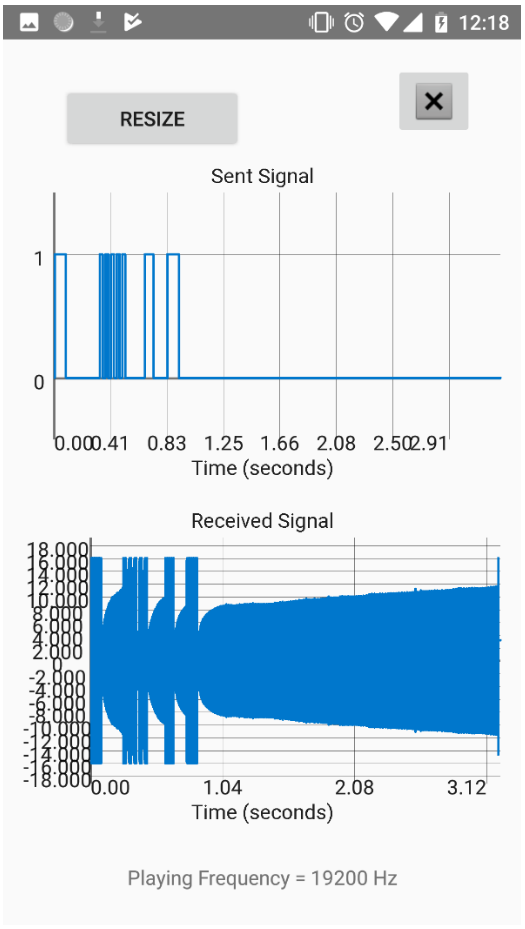


Figure 19: Graphs display screen for the second messaging app

A more detailed description of all the changes made by him in this second take can be found in his report. In general, the enhancements added in comparison to the previous application would be the following:

- Cleaner and more user-friendly layout, with an additional screen to show the graphs in better resolution. These graphs can be zoomed in and out and are scrollable.
- The graphs depicted include the symbol sequence sent as well as the received signal, so it is easier for the user to see the difference between what they sent and what they received.
- 0's are now modulated as a sine signal of smaller amplitude than 1's. This way we distinguish between receiving a zero and not receiving any message at all.
- Instead of encoding one bit at a time, the whole sequence of characters is pre-encoded and modulated and then thrown into the buffer. This way we avoid issues related to continuously stopping and starting the sound generation (audible clicking).
- More robust header, using the sequence 10101010, in this case preceded by an extra set of symbols, because it was proved through experiments that the amplitude of the first part of the played buffer suffered from great attenuation.
- Added a footer composed of 0's, whose goal is to correct the attenuation in the last part of the transmitted buffer, found in the same way as with the first symbols for the header.
- The length of the messages is not fixed, in this case we send within the header the number of characters that will be sent in the message body.
- Automatic gain control implemented. The system takes the previously received symbols to look for the RMS values for ones and zeros.
- Error correction using Hamming coding. The encoded symbols are additionally encoded with (7,4) Hamming encoding to be able to detect bit errors on the other end.

Nevertheless, there is an additional limitation with this application. This is that, since the computations done are way more advanced and complex than in the previous one, it was not possible to use the sending and receiving part in the same app. The base system sends a message and right afterwards it listens to it and decode it, not being able to detect messages the rest of the time. To this app it was added one that could only receive when we press a button. This way could corroborate that the communication also works from one device to the other.

From all these points, we will discuss in further detail the error encoding scheme. This is due to the fact that this was the part in which I mainly worked for this second prototype. For more information about any of the other features, I recommend again reading Yann's report, or looking at the code included at the end of this document. Despite these improvements, the main core is still working using the first prototype that was formerly developed.

Regarding the error correction process, I implemented a Hamming (7,4) linear encoding to the messages. This encoding transforms adds three additional check bits to every 4-bit long block from the original encoded message. This way, we assure that the minimum Hamming distance between any two blocks is equal to 3 bits, this way we can detect single-bit errors in our received code and correct them.

To carry out the Hamming encoding, we use the Hamming matrix G, shown in Fig. 20. To obtain the 7-bit Hamming code we simply multiply the 4-bit segment by this matrix in our code. The encoded blocks are then put into the playing buffer, as it was done before. In this sense, the new encoding part is similar to the previous one, just adding an additional encoding step in between.

On the decoding side, however, the process adds a new correction step. For every set of 7 bits, Hamming encoded, that are received in the recording buffer, we calculate what it is called syndrome array. This syndrome value is a 3-bit sequence that is computed by multiplying the block by the matrix H. This Syndrome value is used to check the parity of the message. Using the table 1, we can detect if there were any single-bit errors in the code. If the syndrome is different from 0 we check the Syndrome table, and then invert the bit that was wrong from the original message.

For our application, we are not encoding the header, so that way the decoding for the synchronization signal is faster and after that we can just start decoding and analyzing the error for the rest, once we have detected the message. The rest of the message is then encoded, dividing each byte by two groups of 4 and then applying the Hamming coding to them. On the receiving side we calculate the syndrome for every 7 bits sequence, we compare it to the table to see were the error is, and then we correct the wrong values.

$$\mathbf{G} := \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{H} := \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 20: Error correction matrices. Hamming matrix G, Syndrome matrix H, and inverse Hamming matrix R

After correcting all the 7-bit blocks, we decode them by multiplying them by the matrix R, or inverted Hamming matrix, obtaining the original 4-bit blocks that we can then decode, by groups of 8, into ASCII characters.

Syndrome	Error bit position
000	No error
001	0
010	1
011	2
100	3
101	4
110	5
111	6

Table 1: Syndrome error correlation

3.3. Applications

The applications for this type of communication are countless. On the introduction we already briefly mentioned some of them. The objective of this section is to offer a wider discussion of these applications and how the technology would be implemented for those. We would not only focus on the applications that already exist, but we will also try to provide new proposals for what this technology could be used for.

Home automation

To use ultrasound or near ultrasound in communicating IoT home automation devices was the main application thought when we started this project. IoT networks using Wi-Fi and Bluetooth are very popular and well developed for home automation applications. Nevertheless, due to the advantages that we find in the use of ultrasound, that were stated in the first chapters of this document, we could use ultrasound as an alternative or an additional connection method.

The possibility to easily broadcast a message from one device to all of the ones that are within range, without the need of a previous synchronization process, is the main reason behind the great potential this technology has. The user's smartphone can send just one message, and immediately, all the IoT devices close to it will receive and decode the data.

Furthermore, if we add this communication scheme to other connection methods the functionality increases exponentially. Imagine a house whose lights are controlled by IoT controllers. When the user comes home, its smartphone can detect they are arriving home via GPS, or by connecting to the household Wi-Fi, and then start broadcasting the message "turn on the lights" using ultrasound. This way the smartphone does not even have to be activated by the user or be broadcasting the message all the time, which would consume more power.

User identification

This broadcasting feature that was previously mentioned also makes this technology perfectly suitable for secure user authentication. The user's smartphone could broadcast a secure code through their self-phone speaker, instead of needing to use identification cards or other more inconvenient security procedures.

The short range of ultrasonic communications ensure that only when the user is close enough to the identification device, this will grant him the permission, or store his information for some purpose. An example of use for this would be to grant access for a user to enter through a door. Instead of the user having to use a key or scan an ID, the door lock could just detect the ultrasonic signal emitted by their phone and grant them the access if they are authorized.

Nevertheless, this setup would have a problem. And that is that for this system to be easier and faster than other identification methods, the phone should be constantly broadcasting the message, using a lot of power. Another solution would be for the device to first detect that there is an authorization request, and then send the information.

Secure data exchange

As it was previously mentioned, most of the already developed applications for using ultrasonic communications with smartphones have targeted secure data transactions [21], such as digital currency exchange or bank account information managing. Like with secure identification, the fact that ultrasound can only be received from a short range, makes it perfect for this type of protected communications.

Wi-Fi networks present several problems in this sense, since we are not protected against fake networks, attackers within the network, or even the security of the router hardware itself. With ultrasounds we could directly communicate two devices without the need of an external network, in a way that would hardly be intercepted unless the attacker is substantially close to the message source.

We could also add end-to-end encryption to this scheme, agreeing both ends on a set of keys prior hand. With this, we will have a very robust and secure channel for sending sensitive information between two devices.

Indoor navigation

The use of ultrasounds for close distance navigation systems is not a new idea. It has already been explored by Qi et al. [2], as well as many other authors [23]. All of them point out the potential in using ultrasound to Wi-Fi for navigating in a more precise and error-prone manner.

Its basic principle is inherited for Wi-Fi indoor navigation systems, being the main difference the use of acoustic signals instead of the electromagnetic waves. The device obtains its location by triangulating the strength of several signal emitting nodes, in this case speakers, that send short messages.

Using ultrasounds would present better precision than just using Wi-Fi networks, up to 0.5 meters against 1.5 meters achievable with Wi-Fi [22]. This is again caused by the great performance of ultrasounds in short distances. Since the attenuation of the sound in the air is so considerable, it is easier to measure distances by the amplitude of the signal received than with Wi-Fi, in which attenuation is more subtle.

Messaging on airplane mode

An additional application for this kind of platform would be to use it as a messaging application that can work in situations where wireless communications are not possible or allowed. An example for this would be to send messages inside a plane that is in mid-flight. Since the use of electromagnetic waves could interfere with the airplane systems, using ultrasound could be a suitable alternative.

This application is not of that much importance as the other ones provided. The situation in which this application would be useful is very specific and unusual. However, it is still a perfectly valid functionality, and further market research should be done in order to discard it as non-viable for its constraints.

4. Results and discussion

This chapter includes all the results obtained from the experiments carried out during this second part of the project. Due to the mainly researching content of this work, we will not present many quantitative results, instead we will mostly analyze the systems developed, pointing out their performance and overall behavior, comparing them to the objectives that we presented at the beginning.

Each section refers to one of the three different applications that were discussed in the previous chapter. Notice that since each one of them mainly expands the functionality of its predecessors, most of the conclusions obtained from their results may be very similar.

4.1. Sending and receiving ultrasounds in smartphones

This experiment was aimed to obtain very simple and straightforward results. The main concern was to know whether the smartphone could perform this type of communication, and to explore the possibilities of the hardware, in terms of speed, sample rate and computational power. Overall, it worked correctly, and we could verify that signal processing at near-ultrasonic rates was feasible with most Android smartphones.

First, we saw that the maximum sample rate usable for playing and recording with the Android OS and the hardware components of most Android devices was 192000 Hz. Since we are aiming to transmit signals within the near ultrasound range: 18000 – 22000 Hz, this sample rate is high enough for us to have an acceptable precision. Using 192 kHz we would send a cycle of a 20 kHz sine every 192/20 = 9.6 samples. This value is higher than the limit set by Nyquist theorem of a minimum of 2 samples.

Regarding the frequencies that are usable for the communication, we observed that after 22500 Hz the device stopped emitting any noise, so we could set the maximum limit around that value. After 19000 Hz we started experiencing some attenuation in the signal received, but with decreases of under the 10%. This means that any value of frequency over the audible range [24] and under that 22500 Hz upper limit should be fit to be used.

The computational power of these devices also proved to be high enough to handle all of the operations to record and process the audio at this high frequency. The filter implemented has 6 poles, but we could test higher numbers to increase the signal to noise ratio received. Nevertheless, since our final goal is to implement a quasi-real-time processing platform, the use of filters with a greater number of taps may be too much for the resources available. It was later corroborated that the use of more complex filters made the system to crash when trying to read the buffer every 75 milliseconds.

Finally, these experiments also served us to see the high dependency on the device's hardware for this kind applications. Some of the older smartphones that were tested could not run this application due to their sample rate constraints. With this, we want to point out the short-term validity for these results, since technology improves exponentially, and we may expect all of the values for either maximum sample rates and frequency ranges to be dramatically improved during the next years.

4.2. Messaging app, first version

The importance of the results obtained from testing this prototype are undeniable. It was our first attempt at building a fully functional app that allowed two devices to communicate to each other. Overall, the app worked with several flaws, being useless from a practical point of view, but extremely instrumental for our research, allowing us to move on into more complex systems.

Regarding its general behavior, we corroborated that the system works without crashing and in a predictable way. It can be used for sending short messages with enough accuracy between two devices that are close to each other. The app has many limitations in terms of precision and speed but, overall works in the expected way.

Performance of the ultrasound communication

Using the app to send the 5-character message “hello” for 10 times in a row, separating the devices four inches and facing the microphone/speaker of each one to the other, lead to 5/10 successful transmissions, being 4 out of the remaining 5 failing in only one bit, that was either lost or mistaken. When we tried to increase the distance between the phones, the system stopped working, most likely because the signal to noise ratio decreased substantially.

With these results we can, once more, see the limitations of this first prototype. It is only suitable for very close-by communications, and it still presents many errors. Nevertheless, we checked that the encoding, modulation and filtering of the data were carried out without a problem. Most of the issues observed were mainly caused by the lack in robustness of the system to adapt to the conditions of the environment.

On top of that, we must take into consideration that all the experiments were done in the ECE 597 laboratory, with many other students and member of the faculty working near us, talking and therefore, making noise. To this human-generated background noise we can add other phenomena, such as the AC working on the background, the wind, sounds from outdoors that come through the windows, echoes… Some of these issues, were targeted to be solved with the next version, that, as we will see, did in fact have better performance.

Finally, it is important to address the quality of the ultrasonic communication. With this, we refer to the absence of audible noise when sending the messages and the volume of it. With this first application, we experience several clicking noises on the speaker caused by the alternation between zeroes and ones in the message. However, these clicks were not loud enough, nor long enough, for the communication to be perturbing to the people close to the device.

Speed

About speed, the application took over 4 seconds to send the message. This result was expected due to the low symbol rate used (75 ms). Later changes for the second version were targeted to improve these results, although we have a great limitation in terms of computation time. When we tried to increase this symbol rate for this app, the system kept crashing. 75 milliseconds was found to be the minimum time for the device to be able to compute the filtering and all the other computations that take part in the process.

User interface

The layout developed is user-friendly and intuitive. Most of the text fields included are there for troubleshooting purposes only and, thus, would be taken off from the final application in case it was released to the market. Nevertheless, the configuration options, for setting up the wave frequency, and to depict the sample rates could be moved to another screen (an extra activity), for the user to not get distracted with that much information.

Additionally, the color palette uses the default settings from Android Studio. This is a cosmetic feature that can be easily changed for future releases too. Since our main goal was to correctly implement the communication scheme, aesthetics was a minor concern for the development of this project.

4.3. Messaging app, second version

This second application presented, in general, an improved performance and a more extensive functionality that the first prototype. Nevertheless, since it was built upon that first version, we can still see that many of the flaws and features found in that previous one are still present in this new application.

We have analyzed these results in relation to three categories: performance, speed, and user experience, the same way as they were organized for the previous section. However, there is a major problem that cannot be located in any of those three categories that is very important to discuss for this second prototype. This is the impossibility to send and receive messages with the same app.

The need of an additional application to just be able to receive the messages from other devices presents a major drawback for this new implementation. All the attempts to try to make the system read the received audio periodically failed. The computing limitations of the Android phone imposes an unavoidable constraint, that should be solved in order to progress with this application. We will discuss about this future improvement on chapter 6.

Performance of the ultrasound communication

One aspect that improves the behavior of this second version is the new header structure, in which we use a longer and more robust synchronization signal and we also encode the number of characters that will be in the message body. Synchronization was achieved with positive results in most of the tests that were carried out. The fact that we added some extra bits at the beginning and the end of the message greatly improves the quality of the reception of the message body. While with the previous attempt half the tries to send data failed, in this case we get successful results for 9/10 messages sent.

Unlike in the first prototype, the message length is in this case variable. The sender introduces in the text field a message formed by the number of characters they want, and the receiver dynamically adapts to this depending on the number stated within the message header. This feature was implemented successfully, and we did not appreciate any errors during our tests regarding it. The receiving part always gets the right number of characters if the synchronization signal was correctly received.

The error correction does not seem to improve the results, since in 9 out of 10 times the message received is correct without any need of correction (syndrome equals to 0 for all its blocks). That 10% of failed tries is caused, as it was mentioned, by not receiving the correct synchronization signal.

We can assume that these improvements experienced in terms of accuracy are mainly caused by the implementation of the automatic gain control, that makes the system's detection of ones and zeroes more robust. The filtering and the rest of the processing done to the signal is the same as in the first application.

Regarding the audible noise produced, this second application still presented the same problem experienced with the previous one. The data transmission generates several clicking noises with the message, but these are not disturbing in any way for the beings next to the devices. Notwithstanding, this is a problem that should be addressed, and that is most likely caused by the fact of the switching between zeroes (no sound) to ones (sine wave is produced).

Speed

Regarding speed, we find that we did not improve the results obtained from the other versions. In this case, the communication takes up to 10 seconds for just one 5 letters message. This may be caused by the new more complex data management process, that has to handle variable length messages and dynamic buffers.

Nevertheless, we are also making some additional tasks only needed for debugging such as printing the original and encoded messages, as well as the graphs for the sent and received signals. These processes involve heavy computations and, thus, take a lot of time to complete. We can expect that the main application could work slightly faster if we get rid of these temporal functionalities.

User interface

The layout for this new application stands as one of the biggest improvements compare to the previous one. In general, all the elements from the main screen are equally placed and following symmetrical patterns that are pleasing to the eyes. Additionally, the external screen to visualize the waves results more convenient, since this way we can analyze them with good precision, and do not distract the users in the main screens from the rest of information.

The addition of a field to set up the number of cycles of the sine wave to encode each symbol makes it for a more versatile system in terms of testing. We do not have to change the code for testing different values for this parameter, making it easier to troubleshoot and carry out our experiments.

On the other hand, since the encoding scheme for this case is more convoluted than the one previously used, it would be convenient to dispose of a new screen to visualize details about the encoding part. In this additional screen, that could work in a similar way to the one created for depicting the graphs, we could see the value of the message in the different encoding steps: binary encoding, Hamming encoding, received sequence, syndrome-corrected received sequence, and then final decoding.

5. Conclusions

As it can be seen from the results obtained, the project projects many promising results, all pointing towards ultrasonic communication to be a game changer in the near future. Along all the applications that we discussed in chapter 3.3 we could point out countless more, specially if we consider hybrid systems that use ultrasound in addition to Wi-Fi, Bluetooth, or other technologies.

With this project, we achieved to prove that it is possible to use commercial hardware devices such as smartphones to manage ultrasonic communications without a problem. From our first application, we showed that the sample rate that can be used is high enough for this communication. Furthermore, with our messaging apps, we could successfully send messages between two devices using this channel. T

Nevertheless, the project itself did not reach all the goals that were expected. We can see the countless limitations that are found in all the applications developed. Although a great progress was achieved in the process, a final, ready-to-use application is yet to be developed.

In terms of accuracy, we need more robust processing methods. As we saw, errors are pretty frequent in the codes received, partly because of the distance constraints, partly because of the failure in filtering all the present background noise. We can expect that, by testing more advanced modulation and encoding techniques, as it will be discussed in the next chapter, much better results in terms of accuracy could be obtained.

Regarding speed, this is the point in which we can see the biggest flaws for the platform developed. The impossibility of using a higher than 16 Hz symbol rate, leads to a very slow process for the message to be first sent and then received. The use of near-ultrasonic carriers already imposes itself a great limitation in terms of speed (the maximum symbol rate would be around 22.4 kHz), but we can see that we are still far away from that limit.

Likewise, the incompatibility of the system to many older smartphone models, as well as new ones with limited computational power, stands as a major issue. This factor is the only one not caused by our work, but it is still an important point to take into consideration. As it was stated, we expect this kind of applications easier to implement and more versatile in a near future, looking at how fast smartphone technology is advancing.

Overall, there is still a lot of research to do on the matter. But this project perfectly sums up the problems we are trying to face and the obstacles that are found for fixing them. It stands as a starting point for other researchers to launch into this field of application and work on a state-of-the-art approach for wireless communications.

6. Future work

In this last chapter, we will discuss about the points that could be improved from the work done, as well as about future lines of work related to this project, that we propose to be explored. As it was mentioned in the previous section, the applications developed are still far from achieving the final objectives that we proposed in the beginning of the project.

On top of that, this report, and the whole project in general, has been mainly focused on testing the technology and develop a software platform to support future applications. To explore these applications is certainly a must for future lines of work, although they were briefly discussed on chapter 3.3.

6.1. Points to improve

First, let's discuss the points that should be improved from the work done so far. Despite global optimizations and enhancements in term of precision, speed and other parameters, we want to focus on the parts that did not work as expected or that were left unfinished.

On this matter, one of the biggest flaws, observed specially on the latest versions would be the impossibility to send and receive messages at the same time. We could do this on our first application, however, this one presented very limited accuracy and overall performance. For this matter, a further research on the matter must be carried out, in order to find a balance between performance and the possibility to record the audio all the time.

Another problem experienced was the slow speed achieved. Although we were expecting great limitations in this matter, since we used 20000 Hz carriers, the rates achieved were bellow 20 Hz in most cases. This is a matter that has to be addressed along the previous one, since the speed directly depends on how much time we need to process every sample.

Optimizing the computations done, specially in terms of filtering and encoding, would presumably improve the results obtained in these two aspects. If that measure do not work we would then try to reduce the complexity of the computations, giving away some precision, in order to improve speed and performance.

Finally, it is important to point out that none of the solutions tested provided totally noise-free operation. This is a problem that we had already seen in other commercial platforms, in specific Chirp, so we are not sure if there is a fix. Notwithstanding, a further and thorough research on this must be carried out to doubtlessly discard this feature.

6.2. Additional functionality

The goal of this section is to propose new features that could be added to the current implementations without the need of redesigning the whole system or moving on to another collateral field. For these purposes we have the next and final section.

A feature that would be easy and useful to implement would be to include information about the sender within the message header. This way we could identify the authors for each message and process the information consequently, an essential feature if we want to handle communications between more than two devices.

In addition to this, we could add some kind of scheme for linking devices, such as Wi-Fi or Bluetooth connections do. This way we can manage the information more efficiently in the app, allowing the user to have different contacts or usual devices to connect to. This functionality is already used for the Google Nearby SDK, in which the communication works following a subscription protocol, in which we first connect the devices and then start the information exchange [7].

Last, we could try to imitate the existent messaging apps adding to the interface a log of all the previous messages, including times and dates. This way we would improve the user-friendliness of the app, adapting it to more familiar screen layouts, such as Facebook Messenger, or Whatsapp.

6.3. Future lines of research

Regarding other related lines of research that should be explored, we have to first point out the exploration in detail of those applications that were proposed in chapter 3.3, as well as many others that could be thought of. As we can see, some of these applications would be sufficed by just minor changes to the already-developed application, for example in communicating phones on airplane mode. Others, however, would require for an extensive research and designing work, such as the navigation applications.

The study of new modulation techniques also stands as a vital part of future researches. Apart from DSSS, massively used for this kind of applications, we would propose the exploration of other more complex techniques as Manchester encoding. Furthermore, we would explore the possibility to develop our own modulation technique, that is optimized for the problem we try to solve.

Last of all, we believe it will be interesting to explore the compatibility of these developed apps and, in general, to the solution proposed, to other platforms. As it was mentioned, at the beginning of the project, one of our goals was to make the implementation cross-platform. This objective was left out for now as unrealistic with our time resources, but further work should be done to see how we could adapt this Java programs to other operating systems and languages.

References

[1]: P. Getreuer, C. Gnergy, R. F. Lyon, and R. A. Saurous, “Ultrasonic communication using consumer hardware”, IEEE Transactions on Multimedia, vol. 20, pp. 1277-1290, 2018.

[2]: J. Qi, and G.-P. Liu, “A robust high-accuracy ultrasound indoor positioning system based on a wireless sensor network,” Sensors, vol. 17, no. 11, p. 2554, 2017.

[3]: Wei Su, Yurong Lin, Wenhui Liu, Xialin Jiang and Xiaoyi Hu, "Study of single-carrier coherent high-speed underwater acoustic communication," *2013 OCEANS - San Diego*, San Diego, CA, 2013, pp. 1-4.

[4]: M. Chitre, S. Shahabudeen, L. Freitag, and M. Stojanovic, “Recent advances in underwater acoustic communications amp; networking,” in OCEANS 2008, Vols. 2008-Supplement, 2008, pp. 1-10.

[5]: C. Lopes and P. Aguiar, “Aerial acoustic communications,” in Proc. IEEE Workshop Appl. Signal Process. Audio Acoust., 2001, pp. 219–222.

[6]: J. Saniie, “ultrasonic signal processing: system identification and parameter estimation of reverberant and inhomogeneous targets,” Ph. D Thesis, purdue University, 1981.

[7]: Google Inc., “Nearby,” Google Developers. 2019. Online. Retrieved from: https://developers.google.com/nearby/

[8]: Chirp, “Chirp applications,” 2019. Online. Retrieved from: https://chirp.io/

[9]: GNU Radio, “GNU Radio, The Free And Open Software Radio Ecosystem,” 2019. Online. Retrieved from: https://www.gnuradio.org/

[10]: Weijie Shen, Haixin Sun, Yonghai Zhang and En Cheng, "A comparison of modulation in underwater acoustic DFT-SOFDM system," *2009 2nd International Conference on Power Electronics and Intelligent Transportation System (PEITS)*, Shenzhen, 2009, pp. 184-187.

[11]: H. Yu, W. Kim and K. Chang, "A study of multicarrier modulation schemes for underwater acoustic communications," *2014 International Conference on Information and Communication Technology Convergence (ICTC)*, Busan, 2014, pp. 747-748.

[12]: B. Schweber, “Eye and Constellation Diagrams, Pt 2”, September 15, 2017. Online, Retrieved from: https://www.analogictips.com/eye-and-constellation-diagrams-pt-2/

[13]: S. Zoican, "Frequency hopping spread spectrum technique for wireless communication systems," *1998 IEEE 5th International Symposium on Spread Spectrum Techniques and Applications - Proceedings. Spread Technology to Africa (Cat. No.98TH8333)*, Sun City, South Africa, 1998, pp. 338-341 vol.1.

[14]: R. Hanlon and C. Gardner, "Error Performance of Direct Sequence Spread Spectrum Systems on Non-Selective Fading Channels," in *IEEE Transactions on Communications*, vol. 27, no. 11, pp. 1696-1700, November 1979.

[15]: M. Z. Win and R. A. Scholtz, "Ultra-wide bandwidth time-hopping spread-spectrum impulse radio for wireless multiple-access communications," in *IEEE Transactions on Communications*, vol. 48, no. 4, pp. 679-689, April 2000.

[16]: Xiaowei Wang, Minrui Fei and Xin Li, "Performance of chirp spread spectrum in wireless communication systems," *2008 11th IEEE Singapore International Conference on Communication Systems*, Guangzhou, 2008, pp. 466-469.

[17]: Telecom ABC, “DSSS - Direct Sequence Spread Spectrum”. No date. Online. Retrieved from: http://www.telecomabc.com/d/dsss.html

[18]: V. K. Goyal, J. Kovacevic, R. Arean and M. Vetterli, "Multiple description transform coding of images," *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*, Chicago, IL, USA, 1998, pp. 674-678 vol.1.

[19]: R. W. Hamming, "Error detecting and error correcting codes," in *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160, April 1950.

[20]: Graphview, “Android Graphview”, 2019. Online. Retrieved from: www.android-graphview.org/download-getting-started/

[21]: B. Zhang *et al*., "PriWhisper: Enabling Keyless Secure Acoustic Communication for Smartphones," in *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 33-45, Feb. 2014.

[22]: S. Sosa-Sesma and A. Perez-Navarro, "Fusion system based on WiFi and ultrasounds for in-home positioning systems: The UTOPIA experiment," *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Alcala de Henares, 2016, pp. 1-8.

[23]: I. Marin-Garcia, P. Chavez-Burbano, A. Muñoz-Arcentles, V. Calero-Bravo and R. Perez-Jimenez, "Indoor location technique based on visible light communications and ultrasound emitters," *2015 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, 2015, pp. 297-298.

[24]: R. Pujol, “Human auditory range,” Cochlea.org, May 6th, 2018. Online. Retrieved from: http://www.cochlea.org/en/hear/human-auditory-range