Camera configuration using the audio ports and a smartphone

Olof Landin
ic07ol3@student.lth.se

Johannes Jakobsson
dt06jj0@student.lth.se

June 28, 2012

Abstract

Audio input/output provides a common interface among a wide range of hardware devices. Axis Communications AB have identified the possibility to implement a communication protocol making it possible to send data between the audio ports of a smartphone and an Axis device. The goal of this thesis is to send image and video from an Axis device to a smartphone. This sets the highest requirement on the transmission rate of the protocol.

Audio-based communication has been achieved by the implementation of a simple protocol, working at four of the levels in the OSI-model: *the physical layer*, *the data-link layer*, *the session layer* and *the application layer*. The session and application layers handle the dialog control and what type of requests that are possible to send, and the response to a specific request. A request is put into a frame in the data-link layer. The frame consists of a header with a data-length field and a checksum followed by the request data. At the physical layer, the frame is then modulated into an array of samples using *Pulse Amplitude Modulation*. All samples are then sent to the sound driver, which plays the signal as audio. At the receiving end, *the Catmull-Rom spline algorithm* has been used to reconstruct the signal during demodulation.

Performance for different configurations of the protocol is measured by the bit rate and error-ratio for every configuration. The impact of characteristics such as the symbol frequency, the number of PAM levels and the use of signal reconstruction have been examined.

The configuration consisting of a 16-level PAM implementation and a baud rate of 2666 Bd, resulted in a throughput of 1180 byte/s. This was the best performance achieved during the course of the project. The byte rate of 1180 byte/s is sufficient to send an image over the channel, but not to send video with a pleasant frame rate. There are still possibilities to improve the protocol, trying different pulses, other modulation techniques such as QAM and sending compressed video instead of a series of JPEG-images.

Acknowledgements

At Axis all the people we have talked to have been very friendly and helpful. We would like to thank all the people at the new business department and in particular our advisor Marcus Johansson for his valuable input as well as his ability to make us feel like a part of Axis Communications.

We would also like to thank our advisor at Lund University, Fredrik Tufvesson, for his feedback on the project. Thank you!

Contents

Chapter 1

Introduction

1.1 Background

Audio input/output provides a common interface on a wide range of hardware devices. Axis Communications AB have identified the possibility to implement a communication protocol, sending data between the audio ports of a smartphone and an Axis device. By using such a protocol, it would be possible to configure a camera by connecting it to a smartphone. Since many Axis devices as well as many smartphones already have audio input/output ports, a communication protocol using these ports could be compatible with many different devices.

Common configuration tasks are: setting the pan, tilt and zoom levels as well as adjusting the focus of a camera. One scenario when this solution would be useful is when a technician is installing a camera. Instead of having to use a custom made device called the *installation module* (see figure 1.1), the technician can use his own smartphone.

When adjusting the focus and framing for the camera using a smartphone, the actual image the camera is recording is required to be sent from the camera to the smartphone. Ideally the images are to be sent as video with a reasonable frame rate. Image and video sets the highest requirement on the desired transmission rate of the protocol.

Figure 1.1: The installation module used when configuring an Axis device

1.2 Problem description

The problem investigated in this thesis is as follows:

Is it possible to implement a, *proof of concept*, software protocol for duplex communication capable of sending digital data between the audio ports of an iPhone and an Axis camera? Below is a list of features wanted for the protocol, ordered by importance:

1. Duplex communication
2. The ability to send an image from the camera to the smartphone.
3. The ability to send video, with a reasonable frame rate, from the camera to the smartphone.

1.3 Limitations

The thesis is aimed as a proof of concept regarding sending information over the audio channel of an iPhone 3G and the Axis camera P3367. For broader use, different camera models and smartphones need to be tested (i.e. different audio chips).

The modulation and demodulation will be done entirely in software, no extra hardware units will be used. The volume of both the camera and smartphone is assumed to be known. A protocol handling variable volume would need additional functionality, since the possible amplitudes in the analog signal is governed by the volume of the device.

1.4 Previous work in the field

As a first step in the project research on what similar projects that have already been done was carried out. Two particularly interesting projects found were the *Project HiJack*[1] and the *Square card reader*[2]. The HiJack project does not only send data over the mobile phone's headset interface, but it also harvests power from it.

HiJack uses a Manchester encoding to modulate the data. Manchester coding substitutes every 1 with 01, and every 0 with 10. Doing this eliminates the saturation of the channel if a long sequence of 1 or 0 would need to be decoded. This approach achieved a transfer rate of approximately 1 kilobyte/s.

The Square card reader, used as a peripheral to read credit cards, also seems to incorporate a Manchester based encoding to send data. No specific bit rate was found for the Square but since it is designed to only send data found on a credit card, high bit rate is not necessary.

1.5 Division of labor

During the work on this thesis both authors have been working together in the same office at Axis Communications in Lund. Naturally there have been discussions, proposals and teamwork during the process. The overall design of the protocol as well as the layout of the report have been done with mutual decisions from both authors.

Most functionality of the protocol have initially been created by one author, but then later on at some point been altered by the other author. But there are some areas where there is no doubt that one of the authors have had more responsibility than the other. A list showing the distribution of responsibilities between the authors is shown below:

Johannes

- Functionality handling requests on the camera.
- Communicating with the camera API to fetch and convert images to a proper format.
- The design of the build environment.

Olof

- All work related with the Core Audio API
- Integration of the protocol written in C with iOS and objective C.
- Simulations of the communication using Matlab.

Chapter 2

Model of a digital communication system

A common approach when designing a protocol for a digital communication system is to use a layer stack architecture. The different layers involved and each of the layers area of responsibility will be described in this chapter.

2.1 OSI-model

In the late 1970s, the Open Systems Interconnection model (OSI) was introduced, a model describing an architecture which allows any two different systems implementing the model to communicate. The OSI-model consists of seven *layers* (see figure 2.1), each layer having a well defined purpose in a network communication process.

There is an interface between each pair of adjacent layers, which makes it possible for data to flow up and down through all the layers in the model. Each interface defines the information and services provided for the layer above it. Systems following the OSI-model can use any combination of protocols, as long as they all implement the interfaces defined by the model [8, page 55].

The network protocol is needed for systems where there are more than two possible recipient hosts, and the transport protocol for systems where there are more than two possible recipient processes within a host.

The presentation layer handles encryption and data compression as well as translation between different data encodings. The network-, transport- and presentation layer protocols won't be needed for the protocol stack implemented in this thesis and won't be further described [8, page 55].

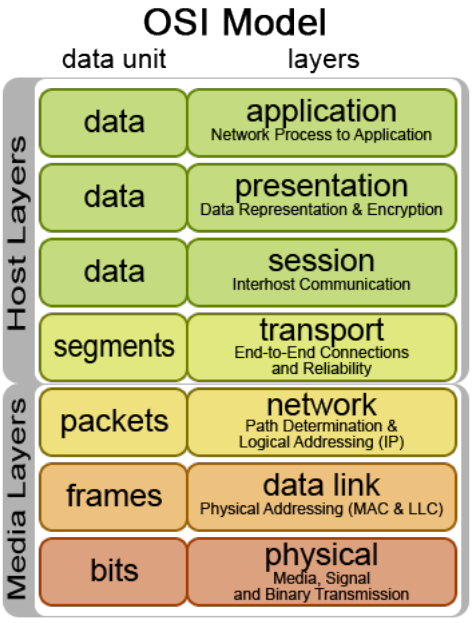

Figure 2.1: The seven layers of the OSI-model

2.2 Application layer

As with all OSI-layers, the application layer provides an interface to the layer above it. Being at the top of the layer stack, the *layer above* in this case means a user or a program using the protocol. The application layer handles the access interface to the network, all possible requests a user is allowed to make are defined by the application layer [8, page 55].

2.3 Session layer

Dialog control and synchronization is the two main responsibilities of the session layer. The session layer allows two systems to enter into a dialog, the communication can be in either *half-duplex* (one way at at time) or *full-duplex* (two ways at a time).

Synchronization points are inserted into the stream of data by the session layer to ensure that all data doesn't need to be resent if an error occurs. For example, an hour long video is sent over a network and a crash happens during the transmission of the 59:th minute of the video. If synchronization points were added after every minute of video, only the frames from the 59:th minute has to be resent [8, page 55].

2.4 Data link layer

The data link layer is responsible for framing, flow control and error control.

2.4.1 Framing

The data link layer pack bits into frames, which can be of either fixed or variable size. The header of a frame often include information such as the source and destination address [8, page 267-340].

2.4.2 Flow control

Flow control is needed in order to control the amount of data that can be transmitted before receiving an acknowledgment. The receiver and transmitter must be in agreement over the amount of data that can be sent before an acknowledgment is received [8, page 267-340].

2.4.3 Error control

When a received bit differs from a sent bit it should be detected and handled by error control. The first objective of error control is to detect an error, for this to be possible redundant bits are added to the data in such way that there is a relationship between the redundant bits and the actual data bits.

Two very common methods for creating the redundant bits are called *checksum* and *cyclic redundancy check (CRC)*. Both methods take a block of data (for instance the data in a frame) as input and produces a vector of bits of a fixed size. For example, if CRC is used, it is included in the frame and the receiver can recalculate the CRC using the received data and compare with the CRC obtained from the frame.

There are two main methods of where to proceed after an error has been detected: *Forward error correction* and *retransmission*. If the retransmission method is used, when an error is detected, the receiver tells the sender that an error occurred. The receiver in turn resends the frame. This process is called *automatic repeat request (ARQ)*. The simplest form of ARQ is called Stop-and-Wait ARQ an works as follows:

Each frame sent gets a sequence number, which is appended to the frame header. A frame with sequence number x is sent to receiver, if everything goes well the receiver responds with an *acknowledgement* (ACK) which has a sequence number of the next frame it expects to receive (in this case $x + 1$). A time-out will be triggered on the sender if no ACK arrives and the frame will be resent. That no ACK arrives could either happen because the receiver never got frame x or the ACK sent from the receiver got lost. If it was the ACK that got lost, the receiver will receive frame x one additional time, but the receiver will have stored the sequence number and knows that it already have gotten that frame. The receiver will (again) send an ACK with the sequence number

$x + 1$. This process will continue until all frames and acknowledgments have arrived correctly.

With forward error correction the receiver tries to correct the errors. This can be achieved if the number of errors is small and the bits are coded (this will add additional redundancy).

Some common coding techniques used in forward error correction are: *hamming coding* and *convolutional coding* [8, page 267-340].

Fletcher's checksum

Fletcher's checksum is a position dependent checksum that approaches the properties of a CRC, but with a lower computational bound. Fletcher's checksum as well as CRC will both detect all single-bit errors and both allow a small fraction of errors to get through undetected; 0.001526% using a CRC and 0.001538% using Fletcher's checksum. A Fletcher checksum of 8 *bits* uses two counters, one for summing up the bytes in the message, and one which takes the cumulative sum of the first counter (see figure 2.2). This solves the problem of the single summation counter where the checksum is insensitive to the order of the bytes [20].

```
sum1 = 0
sum2 = 0

for i in 1 to message_length do
        sum1 = (sum1 + message[i]) modulo 255
        sum2 = (sum1 + sum2) modulo 255
done

check1 = 255 - ((sum1 + sum2) modulo 255)
check2 = 255 - ((sum1 + check1) modulo 255)
```

Figure 2.2: Pseudo code for an implementation of Fletcher's checksum using two check values at 8 bits.

2.5 Physical layer

The physical layer controls the communication between the communicating nodes in a network. A common example of communicating nodes is the communication between a switch and a computer in a *Local Area Network* (LAN).

Two major roles for the physical layer is to convert the binary data from the link layer in to signals that the network can transmit, and to decide the data flow and parameters of the channel between the nodes [8, page 29-42].

2.6 Components involved in the physical layer

The components typically involved in the physical layer are presented in figure 2.3 where the following important components are presented: *message source*, *vector encoder*, *modulator*, *channel*, *demodulator* and *vector decoder* as well as a *message destination*. The basic objective of the physical layer is to send a arbitrary digital *message* over a *channel*, and to receive the same exact message on the receiving end.

The first step in the transmission process is to convert the message into a series of *symbols*, the conversion is carried out in the *vector encoder*.

Consider a system where there are only two possible symbols, 0 and 1. The *message* "0" is sent to the vector encoder which encodes the symbol to the *symbol vector* $s = (5000, 1)$. The vector s is supposed to capture the characteristics of a symbol, in this case 5000 denotes the frequency of the signal and 1 the amplitude. Let S be the set of all possible symbols, $s_i \in S, i = 1..N-1$, where N is the number of different symbols that exists, then every T (*the symbol period*) seconds a symbol from S is sent. The *symbol rate* becomes $1/T$. A message consist of one or more symbols and each symbol has a symbol vector representation.

The next step is to, through *modulation*, convert the symbol vector into a analog waveform that will be transmitted over the channel. For every symbol s there is a corresponding signal waveform. On the receiving end the *demodulator* demodulates the received signal back into a symbol vector r. The vector decoder then tries to estimate the sent message from r. If the receiver output message is not equal to the transmitter input message, an error occurs. The receiver is considered optimal when it minimizes the probability of such errors for a given communications channel and set of symbol waveforms: [6].

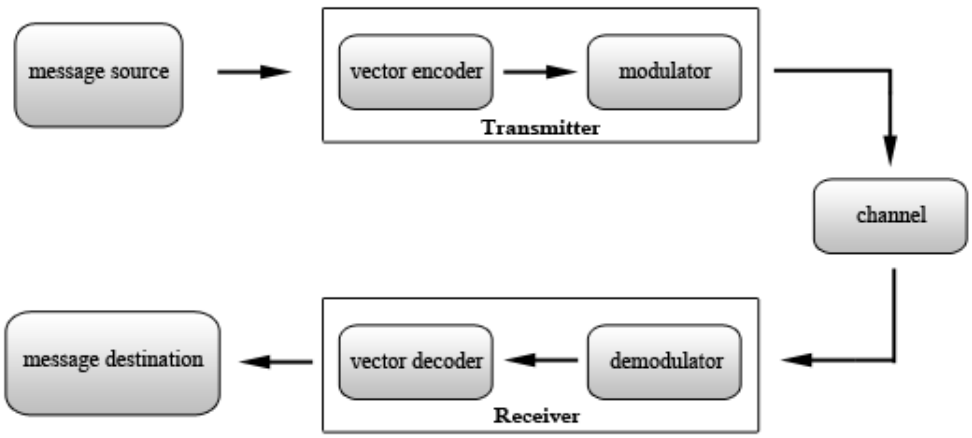

Figure 2.3: Components involved in the physical layer

2.6.1 Synchronization

In order to synchronize the signal, the transmitter can send a predefined synchronization pattern at the beginning of a message. The receiver then computes the cross correlation between the input signal and the known pattern. When a peak in the correlation is detected, the position of the message can be inferred.

An important property of a synchronization pattern is that the pattern's autocorrelation have a distinct peak when the *lag*, l, is equal to zero and low values for $l \neq 0$. This is to ensure that the pattern has a distinct peak only when it is exactly aligned with a matching symbol [5].

2.6.2 MAP and ML Receiver

Let $s_i \in S$ be the sent symbol and $\tilde{s}$ the receivers estimate of the transmitted symbol. P_e denotes the probability of error, the probability of $s \neq \tilde{s}$. The optimal receiver will minimize P_e.

MAP stands for *Maximum a Posteriori* and is a method for deciding which symbol that was sent. If the MAP-receiver receives the symbol vector r then the probability for each of the possible symbols s_i are:

$$P(s = s_i|r) = \frac{P(r|s_i) \cdot P_{s_i}}{P_r} \tag{2.1}$$

Since P_r is independent of the transmitted message, the receiver does not need to take it into account when deciding the sent symbol. The general MAP decision rule can be formulated as follows:

$$\arg\max_i P(r|s_i) \cdot P_{s_i} \tag{2.2}$$

If all symbols are equally likely, the receiver is called the *maximum likelihood receiver* (the ML receiver) and the decision rule can then be shorted to:

$$\arg\max_i P(r|s_i) \tag{2.3}$$

[7, page 233-255]

2.6.3 PCM

PCM, or *Pulse Code Modulation*, is an pulse modulation technique to describe a analog signal as digital. A PCM encoder has three processes:

1. Sampling
2. Quantization
3. Encoding

First the analog signal is sampled at a given sample rate, the sampled values are then quantized to the nearest quantization level [8, page 121-129]. For example, if the samples are stored as 16-bit values, there will be $2^{16} = 65536$ different values to quantizes the sample to. The digital representation of the quantized values are then encoded into a signal where the binary digits are coded as pulses that have the same height and shape. This makes PCM simple since the receiver only needs to be able to locate the presence or absence of a pulse [4].

2.6.4 Pulse Amplitude Modulation

Pulse Amplitude Modulation or *PAM* is a modulation technique that generates as set of possible symbols by altering the amplitude of the signal $g(t)$. With ℓ different amplitudes a signal can be modulated as $s_\ell(t)$.

$$s_\ell(t) = A_\ell g(t) \tag{2.4}$$

$$\ell = 0, 1, ..., M-1$$

The amplitude alternatives, A_ℓ, determines the amount of information that can be stored in a signal $s_\ell(t)$ [7, page 31]. For $M = 256$ (called 256-PAM) the signal can alternate between 256 levels and can thus store one byte ($1byte = 2^8 = 256$) of information in one waveform.

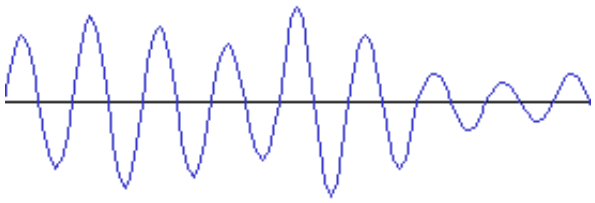

Figure 2.4: A PAM modulated signal with 16 different amplitudes ($\ell = 16$)

2.6.5 Frequency Shift Keying

In the modulation technique, Frequency Shift Keying (FSK), a set of possible symbols are generated by altering the frequency of the carrier wave. The signal alternatives, $s_\ell(t)$ is described as;

$$s_\ell(t) = A\sin(2\pi f_\ell t + \nu) \tag{2.5}$$

$$\ell = 0, 1, ..., M-1$$

f_ℓ denotes the different frequencies the signal can alternate between. ν is the phase of the wave, and A is the wave amplitude. In pure FSK A and ν are constants [7, page 40]. If for example ν could alternate between two values the signal can hold twice the amount of information, but would also require a much more complex receiver.

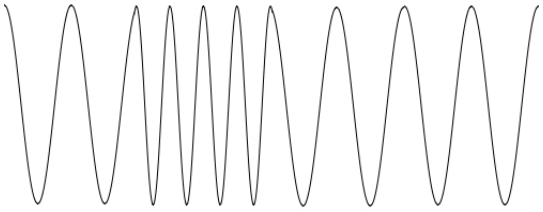

Figure 2.5: Visualization of FSK with two different frequencies modulating the message 0100

2.6.6 Phase Shift Keying

Like the previous modulation techniques, PAM and FSK, Phase Shift Keying (PSK) is modulated with sinusoidal waves [7, page 36]. The symbol alternatives used in PSK are generated by altering the phase of the signal $s_\ell(t)$:

$$s_\ell(t) = A\sin(2\pi ft + \frac{\pi}{\ell}) \tag{2.6}$$

$$\ell = 0, 1, ..., M-1$$

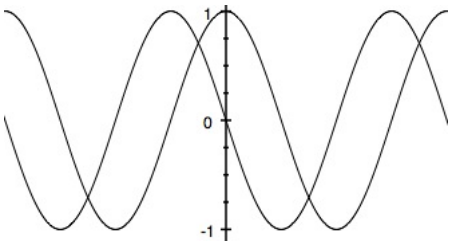

Figure 2.6: Graph of two signals using PSK described in equation (2.6) with different values for ℓ

2.6.7 Quadrature Amplitude Modulation

Quadrature Amplitude Modulation (QAM) is a modulation method in which PAM and PSK are combined. In QAM the information is placed as amplitude values on two orthogonal signal. QAM is defined as:

$$s_\ell(t) = A_l\cos(2\pi ft) - B_l\sin(2\pi ft) \tag{2.7}$$

or equivalently:

$$s_\ell(t) = \sqrt{A_l^2 + B_l^2}\cos(2\pi ft + \nu_l) \tag{2.8}$$

$$\ell = 0, 1, ..., M-1$$

where

$$A_l = \sqrt{A_l^2 + B_l^2}\cos(\nu_l) \tag{2.9}$$

$$B_l = \sqrt{A_l^2 + B_l^2}\sin(\nu_l) \tag{2.10}$$

Equation 2.8 shows how QAM is related to PAM and PSK, $\sqrt{A_l^2 + B_l^2}$ will yield variable amplitudes on the symbols and ν_l variable phase. Each combination of amplitude and phase will map to a symbol in s_ℓ [7, page 45-46].

2.7 Signal-to-noise ratio

A common measurement of channel is the *signal-to-noise ratio* (SNR) [8, page 83]. The definition of SNR is described in equation 2.11

$$SNR = \frac{\text{average signal power}}{\text{average noise power}} \tag{2.11}$$

2.8 Shannon capacity

Claude Shannon introduced a formula called the *Shannon capacity* which gives the theoretical highest bitrate of a channel, given the bandwidth and signal-to-noise ratio of the channel. The Shannon capacity is defined as follows:

$$Capacity = bandwidth \cdot \log_2(1 + SNR) \tag{2.12}$$

The Shannon capacity assumes that all noise the channel is subjected to is *gaussian noise* (noise that has a gaussian distribution) [8, page 87].

2.9 Sampling theorem

According to the Nyquist Theorem a signal can be perfectly reconstructed if the sampling rate is at least twice the highest frequency of the original signal [8, page 121].

Chapter 3

Numerical methods

3.1 Fixed Point Math

Doing calculations in floating point on a computer is often less efficient than fixed point due to the complexity in implementing floating point arithmetic in hardware. The difference in speed is especially notable when floating point hardware is not present on the computer, which is often the case in embedded systems [14].

In fixed point math, decimal numbers are represented using integers, a real number x is represented by an integer X with the word size $N = m + n + 1$ bits.

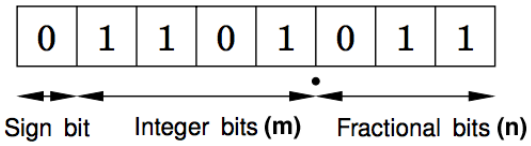

Figure 3.1: A real number with a fixed point representation

The resolution of the number is determined by n, but a larger n means a smaller m which in turn makes the range of possible numbers to describe more restricted. In order to carry out division and multiplication operations with fixed point math, the intermediate result needs to be stored in a structure with double the word size of the operands.

Below is an example of an implementation of a fixed point multiplication (in C):

```
int16_t fixed_mul (int16_t a, int16_t b, int n)
{
        int32_t ans;
        ans = (int32_t)a*b;
        /* Correct for rounding */
        ans += (1<<n-1);
        ans = ans >> n;
        if (ans > INT16_MAX)
                ans = INT16_MAX;
        else if (ans < INT16_MIN)
                ans = INT16_MIN;
        return (int16_t)ans;
}
```

Figure 3.2: Simple example of fixed point multiplication

3.2 Fourier Transform

The Fourier Transform makes it possible to compute the frequency spectrum of a signal by transforming it from the time domain to the frequency domain.

$$X(f) = \mathcal{F}\{x(t)\} = \int_{-\infty}^{\infty} x(t)e^{-2j\pi ft} \tag{3.1}$$

Equation (3.1) shows the definition of the fourier transform $X(f)$, i.e. the frequency content, of the signal $x(t)$. j is a complex number and t and f denotes the time and frequency [7, page 62].

3.2.1 Discrete Fourier Transform

Analogous to the fourier transform for continuous functions there is the *Discrete Fourier Transform* (DFT) for discrete functions. The DFT is defined as:

$$F(u) = \sum_{k=0}^{N-1} f(k)\omega_N^{ku}, u = 0, ..., N-1 \tag{3.2}$$

where

$$\omega_N = e^{-i2\pi/N} \tag{3.3}$$

It is computational expensive to calculate the DFT. The *Fast Fourier Transform* (FFT) is an algorithm to compute DFT more efficiently. To calculate the DFT, with $N = 1024$, using an algorithm following equation 3.2, will require about 10^6 multiplications, while the FFT only requires about 10^4 multiplications [10].

3.3 Linear least squares

The *linear least squares* algorithm (LST) is a method for fitting a curve to a set of data points. A model, which the data is to be fitted to, has to be chosen when LST is used. Examples of models are a line, a parabola or a cubic curve.

For a set of m data points (t_i, y_i) (T and Y are the vectors of all values t_i and y_i) and a model $f(T, X)$, the goal is to find values on X that satisfy the following equation:

$$f(T, X) = Y \tag{3.4}$$

LST is usually used in situations where there is noise in the data an therefore there will be no exact solution to the equation above. Instead LST finds the values of X which minimizes $||Y - f(T, X)||_2$ [11].

3.4 Catmull-Rom spline

Splines come from the mathematical field of numerical analysis and are often used in computer graphics to fit curves to data points.

The *Catmull-Rom spline* is a form of *Cubic Hermite spline* and can be calculated as follows:

$$\begin{aligned} p = a \cdot ((-i + 2) \cdot i - 1) \cdot i \cdot 0.5 + \\ b \cdot (((3 \cdot i - 5) \cdot i) \cdot i + 2) \cdot 0.5 + \\ c \cdot ((-3 \cdot i + 4) \cdot i + 1) \cdot i \cdot 0.5 + \\ d \cdot ((i - 1) \cdot i^2) \cdot 0.5 \end{aligned} \tag{3.5}$$

The formula takes two endpoints, a and b, two points that the curve must go through, b and c and the interval between b and c. By varying i in the range 0..1 the values between b and c can be interpolated [9, page 628-629].

3.5 Cross correlation

Cross correlation is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. A common situation to use cross correlation is in a matched filter receiver. In a matched filter receiver an unknown signal is correlated with a known signal (called a *template*), to detect the presence of the template in the unknown signal [7, page 244-246].

consider two functions, $x(n)$ and $y(n)$, that have the following properties:

$$x(n) = y(n) = 0 \text{ for } n < 0 \text{ and } n \geq N \tag{3.6}$$

The cross correlation between $x(n)$ and $y(n)$ is defined as follows:

$$r_{xy}(l) = \sum_{n=l}^{N-|k|-1} x(n)y(n-l) \tag{3.7}$$

Where N is the length of the sequence to correlate, and $k = 0$ for $l \geq 0$ and $k = l$ for $l < 0$

The special case, where $x(n) = y(n)$, is called an *autocorrelation* of x(n), which is defined as follows:

$$r_{xx}(l) = \sum_{n=l}^{N-|k|-1} x(n)x(n-l) \tag{3.8}$$

An autocorrelation sequence attains its maximum value at zero lag, this is natural since a signal should match perfectly when it is aligned with itself. A cross correlation sequence will retain the same shape even when the signals involved in the cross correlation are scaled. As a consequence of the scaled signals, the cross correlation sequence will also become scaled. Since the shape often is more interesting than the scale, it is common to normalize the correlation sequence to range from -1 to 1. This can be done as follows for an autocorrelation:

$$\rho_{xx}(l) = \frac{r_{xx}(l)}{r_{xx}(0)} \tag{3.9}$$

For a cross correlation sequence, the normalized version is defined as:

$$\rho_{xy}(l) = \frac{r_{xy}(l)}{\sqrt{r_{xx}(0)r_{yy}(0)}} \tag{3.10}$$

[12, page 117-120]

3.6 Inter Symbol Interference

A problem present in almost all communication systems is *intersymbol interference* (ISI), where subsequent symbols interferes with each other. The duration of a pulse is called the symbol time (T_s), but often the shape of the pulse becomes a bit different when it enters the transmission medium. Figure 3.3 shows three pulses to be transmitted and figure 3.4 the pulses affected by the transmission medium. In figure 3.4 the transmission medium has created a tail of energy which makes the pulse last longer than intended. The time it takes for a signal to die down is called the delay spread. If the delay spread is less than or equal to the symbol time, no ISI will result [16].

Figure 3.3: What three rectangular pulses look like when they are sent from a transmitter.

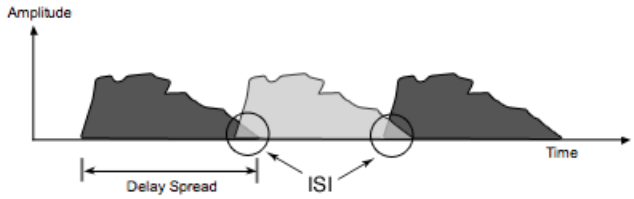

Figure 3.4: This figure shows the pulses from figure 3.3 after they have entered the transmission medium. A receiver will not be able to separate the overlapping signals and will, at the areas marked ISI, sample the sum of the overlapping pulses.

One of the causes of intersymbol interference is what is known as multi-path propagation, which occurs when a signal from a transmitter reaches the receiver via many different paths. This is primarily a problem in wireless systems. Another factor which introduces ISI are systems that have limited bandwidth [17].

Another problem that can introduce error to the receivers decision making process is called *timing jitter* and refers to the fact that the actual sampling point of the receiver will not always be optimal and hit the peak of the pulse [15].

According to Ken Gentile [15], two criteria have to be fulfilled to get rid of ISI:

- The pulse shape should be zero at the sampling point of every other pulse. If the receiver samples at the optimal sampling point, the sampled value will only be influenced by the correct pulse.
- The amplitude of the pulse shape should decay rapidly outside of the main lobe of the pulse. This quality is needed to handle receivers having trouble with timing jitter. If the receiver samples at a non-optimal moment and if the pulse shape doesn't decay rapidly, the risk is that adjacent pulses will have an impact on the sampled value.

3.7 Raised cosine

The raised cosine is a pulse shaping filter used to eliminate ISI as well as to narrow the bandwidth used by the signal. Below is the definition of the filter:

$$rc(t) = \frac{\sin(2\pi t/T_s)}{2\pi t} \frac{\cos(2\pi\alpha t)}{1-(2\alpha t/\pi)^2} \tag{3.11}$$

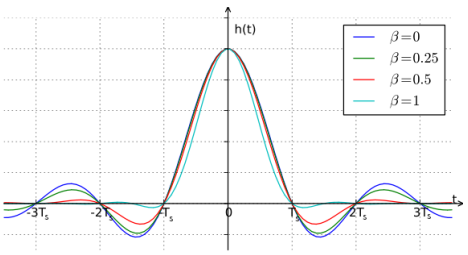

Figure 3.5: The raised cosine pulse shown in the time domain for various values of α

The factor α is called the roll-of factor. With $\alpha = 1$, a pulse uses a one-sided bandwidth of $W = \frac{1}{T_s}$. This is by itself not very impressive, in fact it uses as much bandwidth as a simple rectangular pulse, but it has the added benefit that it is rapidly decaying in the time-domain (the second criteria from section 3.6) [15].

The raised cosine have an adjustable bandwidth, stretching from $W = \frac{1}{2T_s}$ to $W = \frac{1}{T_s}$, depending on the value of α (see figure 3.6).

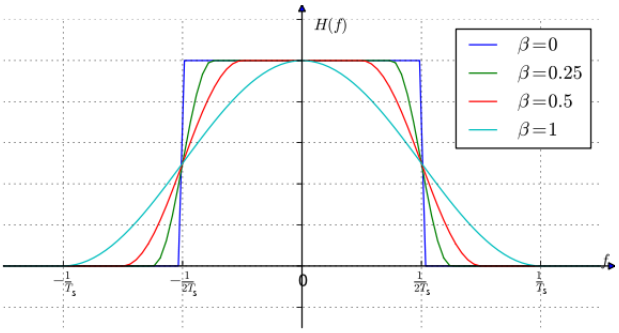

Figure 3.6: The raised cosine pulse shown in the frequency domain for various of α

As mentioned in the section 3.6, a pulse shape should be zero at the sampling point of every other pulse in order to conquer ISI. This is true for the raised cosine pulse and is illustrated in figure 3.7 [15].

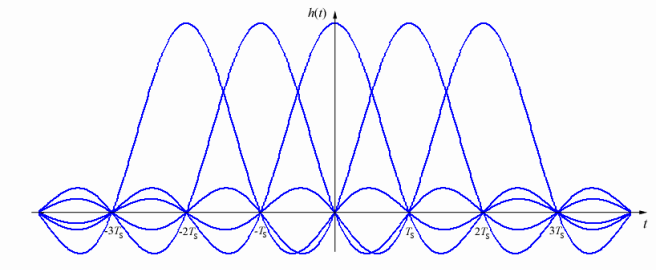

Figure 3.7: Five subsequent raised cosine pulses overlapping on one another. However, because the peak of each pulse corresponds with the zero crossing point of the following pulse, ISI is minimized.

Chapter 4

External API:s

In the design of the protocol some external libraries and API's were used to help speed up the development.

4.1 libfixmath

libfixmath is a platform independent library used to perform fast non-integer calculations by the means of converting them to a fixed point notation [22].

4.2 ALSA

Advanced Linux Sound Architecture (ALSA) is the API used to give audio capabilities to the Linux operating system used on the P3367, Axis camera. The API functions gives the programmer the ability to read and write to the PCM interface [19].

4.3 Core Audio

Core Audio handles all audio requests in Apples operating system for the iPhone. Like the ALSA interface on the camera, Core Audio uses PCM to convert an analog signal to digital [21].

4.4 Kiss FFT

Kiss FFT is a C library implementing the Fast Fourier Transform. The FFT library that can use either fixed or floating point data types [23].

Chapter 5

Implementation

This chapter describes the implementation of the prototype which was developed during the work on the master thesis. The implementation on the camera side (referred to as *Server*) was done in C as an user space application. The API used to communicate with the sound card on the Server was ALSA. On the smartphone (referred to as *Client*) the implementation was made in Objective-C/C, and uses Apples Core Audio framework to read and write to the sound card. An overview of the implementation, in the form of a block diagram, can be found in the appendix (see figure A.1).

5.1 Transmission technologies and protocols

In order to get a working communication system for the prototype, a protocol stack was implemented. The protocol will be described in this section and it consists of four of the layers in the OSI-model: the application layer, the session layer, the data link layer and the physical layer.

5.1.1 Application layer

The application layer contains the high level logic for processing the information that has been received and demodulated on the channel. The data entities handled in the application layer are called *application data packets* (ADP). Each ADP consist of a simple header with one field called the *request type*, after the header comes the *request data*. The two main functions that the application layer handles are:

- **HTTP requests:** An application data packet with the request type *HTTP* is sent by the Client to the Server containing the CGI [1] data that is to be sent to the web server. Upon receiving the CGI, the Server encapsulates the raw CGI data into a HTTP request and then sends it to

[1] Common Gateway Interface. Delegates web server information to executable files on the Server. Used to set different configuration values for the camera.

the local web server located on the camera (Server). The answer from the web server is then sent back to the Client with an ADP with the request type *HTTP_RESPONSE*.

- **Image requests:** This request type instructs the Server to save a JPG image from the video stream and send it to the Client. A picture can be split up in to several smaller ADPs to ensure that the whole picture doesn't need to be resent if one ADP containing a part of the picture gets a bit error. On the Client side the image data gets saved to a buffer until a ADP with request type *PIC_DONE* gets sent from the Server, telling the Client that all image ADPs have been sent and the buffer containing the image can be viewd.

5.1.2 Session layer

Dialog control is implemented in a half duplex manner. In the initial state of the communication system, both the Client and the Server is listening on the channel. When a user triggers an event on the Client, the Client will stop listening and then send out the request triggered by the event (figure 5.1). After sending the request, the Client will listen for a response from the Server. The Server will simultaneously receive the request and perform whatever action the Client asked for and return a response.

No synchronization is implemented in the session protocol.

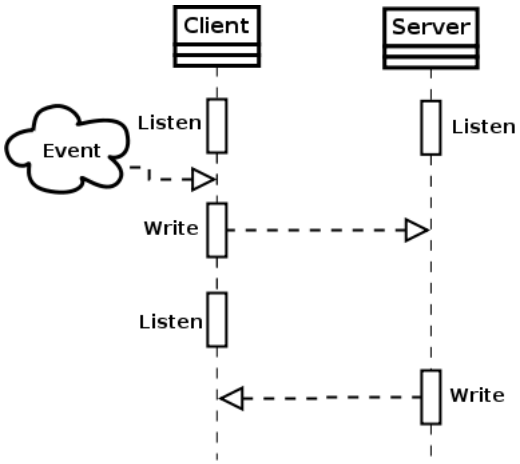

Figure 5.1: Diagram over the communications between the Client and the Server.

5.1.3 Data link layer

Framing

When data is flowing down through the OSI-model, the data link layer will encapsulate binary data into a frame with headers that holds information about the data that is being transmitted. If the direction is up in the OSI-model, the headers of the frame will be removed and the data contained in the frame will be sent upwards.

A frame in this project consists of four different sections;

1. A field stating the number of bytes contained in the data field.
2. Checksum of the data
3. Frame number
4. Data

A frame has a dynamic length and thus the length field is needed to calculate which bytes that contain information.

The second header part, the checksum, is used for error control, so the receiver of the frame can check that the frame was received correctly. The final header part containing the frame number is used in the flow control to ensure that no frame was lost in the transmission. After the header comes the actual data.

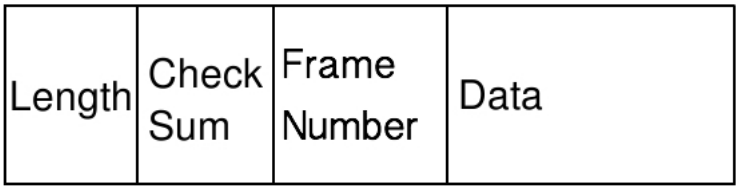

Figure 5.2: Structure of a frame.

Flow and error control

The flow and error control is the two most essential parts of the data link layer, governing how the synchronization between the two devices should be conducted, and how potential errors should be handled.

In a duplex system the flow control needs to be able to tell which device should be listening or receiving, and how much information that can be sent in a frame.

The specific flow control implemented in this thesis incorporates frame numbering and a checksum for error control. The implemented checksum is

based on Fletcher's checksum, described in section (2.4.3). When the transmitted checksum differs from the calculated one, or when a frame number is not the expected one, a NACK is sent with the corresponding frame number that was expected to the sender of the faulty frame. The sender then resends the requested frame.

In the default state of the protocol the Server is waiting for information from the Client. When a frame is sent from the Client to the Server, the Client is blocked from sending another frame until a frame with a valid checksum is received. In the case of a frame not being caught by one of the units, the Client implements a time out that gets triggered if a response is not received in a fixed time.

There is no error correcting codes in place in the protocol, which help keep the overhead on the frame size to a minimum, but also means that a single bit error will force the frame to be resent. Due time constraints there hasn't been sufficient research in the possibility to implement error correction in the protocol, and the possible gains of such functionality.

5.1.4 Physical layer

Modulation

To transmit a bitstream over an analog channel, some kind of modulation is needed to transform the bitstream into a transmittable signal. In the implementation described in this report, modulation and demodulation is done entirely in software. During modulation a frame is first converted into symbol vectors in the vector encoder and then to an array of samples, which is sent to the sound driver and then transmitted to the channel as a analog signal.

Demodulation works the other way around; the sound driver samples the analog signal and sends the sample array to the demodulation function, which in turn sends a symbol vector to the vector decoder to decide which symbol that was sent. Two different types of modulation methods were implemented and tested; PAM and FSK.

Synchronization

To distinguish a message in the analog signal, a known symbol is needed to indicate the start of a message. To achieve this, a special synchronization symbol is appended to every frame coming from the data link layer, which is to be detected before every frame when demodulating a signal.

To locate the synchronization symbol, the program calculates the cross correlation between the input samples and the known synchronization symbol. When a peak is found and the correlation value is above a given threshold, the beginning of the message, s, is found.

A one period long sine wave at 2000 Hz was used as the synchronization symbol for the implementation. The symbol was arbitrarily chosen at the early stages of the implementation and it is perhaps not the most optimal synchro-

nization symbol. The synchronization symbol used have generated a peak in the correlation that has been adequate for determining the beginning of the message.

As used in this implementation, s denotes the index in the sample array where the fist symbol of the message starts. The signal is then split into segments, each segment consisting of l samples (l = symbol sample length). The first segmentation method, follow the formula that the n:th symbol consists of l consecutive samples, starting at position, $p = s + l \cdot (n - 1)$. This works well for short messages, but will not point to the correct samples when n is big. The sampling isn't accurate enough for the above described segmentation method to work for larger values on n. The sampling slowly drifts away and after a while the true position of a segment will be $p-1$ or $p+1$, depending on the direction of the drift.

To compensate for the drift, some kind of resynchronization is needed. As a first approach, an additional synchronization symbol was added for every hundredth byte sent. This adds some unnecessary overhead, both computationally and that more data has to be sent over the channel.

Instead, functionality was created to on-the-fly resynchronize the signal; during the segmentation step three potential symbols are compared to find the one most likely to be in sync. The three different symbols are located at $p = s + l \cdot (n - 1) + i$ where $i \in [-1, 0, 1]$ (see figure 5.3).

The information symbol used in the implementations of PAM and FSK consists of sine waves of one ore more whole periods, which means that the first and last samples of a sampled symbol should have values close to zero. This fact is used by a comparison method, which chooses the symbol that has its first and last value closest to zero. Few operations are needed to compare the three possible starting positions of a segment and the method has worked well to carry out a correct segmentation.

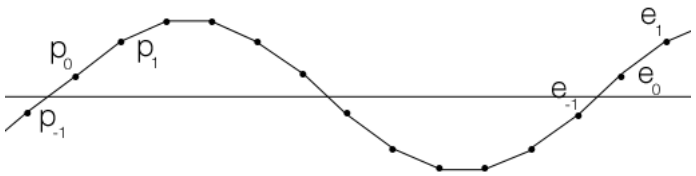

Figure 5.3: p_i, $i \in [-1, 0, 1]$ denotes the possible starting points of a symbol and e_i the end points. In this case $i = -1$ gives the best fit.

PAM

PAM was implemented by locating the max value for each symbol received from the segmentation. In order to better reconstruct the sent signal, the curve fitting algorithm, Catmull-Rom spline, was used. The Catmull-Rom algorithm is relatively cheap computational wise and effectively decreased the mean error of the determined amplitude (see section 6.1).

To find the maximum amplitude of a symbol, the sample with the greatest amplitude is found. Then Catmull-Rom is used to interpolate values between the maximum sample and the two samples immediately before and after. The maximum of the interpolated values give the final amplitude value (figure 5.4).

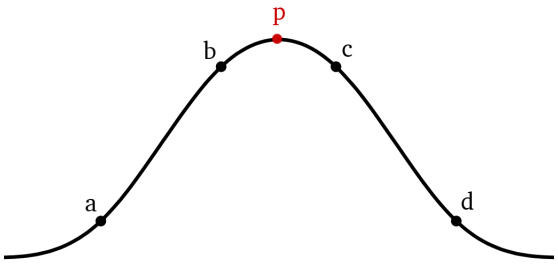

Figure 5.4: Point p with the highest amplitude on the curve fitted to the data points *a, b, c* and *d*

PAM was implemented for 2, 4 and 16 different amplitude levels. It was more straight forward to implement support for configurations where the number of bits per symbol were a power of two. Since a byte consists of 8 bits, it is trivial to divide it into parts of size 2 bit or 4 bits. Diving a byte into parts of for example 3 bits, would of course be possible, but it is not as trivial and has not yet been implemented.

Given the received amplitude r, the sent amplitude s is estimated according to the ML-receiver rule (eq 2.2), $P(r|s_i)$ is approximated by using the inverse of the *Euclidean distance* between r and s_i. The distances are not normalized to actual probabilities, instead the smallest distance accounts for the highest probability. The decision functionality is implemented in a C module called the *vector decoder*.

FSK

An implementation of the FFT algorithm was used to extract the frequency information from the signal. The FFT algorithm used was the *KISS FFT* which has the goal of being easy to implement and understand, rather than being the most effective. To improve the computational efficiency of the FFT algorithm, a fixed point version was used. Using fixed point arithmetic instead of floating point speeds up the computation, but results in round off errors.

The FSK-implementation was used in an early stage of the project and although it did work, it was very slow. Using the FFT-algorithm to extract the frequency yields a lot of computational overhead, since the algorithm computes the correlation with a lot more frequencies than the ones used as information bearing symbols. The FSK-implementation was useful in the beginning of the project because it was easy to implement and could be used to test the functionality of other parts of the project. The PAM implementation eventually performed much better than the FSK (higher bit rate, lower error ratio), so much so that all development later on in the project was focused on PAM.

5.1.5 Calibration

The variables (s_i) used by the vector decoder are dependent on the output volume of the device sending a signal. If no data has been exchanged between the Client and Server, a calibration step is carried out. A sequence of bytes, known before hand by both parts, is sent from the Client to the Server. Using the LST-algorithm with a line as model all values, s_i, can be calculated. This is accomplished by using the values from the known bytes as T and the measured amplitudes as Y (see section 3.3).

5.1.6 Channel

The audio socket on the iPhone has four poles, where one pole is dedicated for ground, one for input (mono) and two for output (stereo). The Server has one input socket and one output socket, each with three poles (two audio-channels and one ground). All sockets where compatible with $3,5\ mm$ TRS plugs. As channel for the protocol a new cable was made that connected the two sockets of the Server with the socket on the Client (figure 5.5).

The iOS-framework gives no possibility for a developer to explicitly choose whether the input audio device is the external or internal microphone. According to Karl Nilsson [13], an iPhone starts listening to the external input if a larger impedance than 800Ω can be measured on the input channel. No information about this was found in Apples own documentation, but after adding a resistor in series with the iPhone's input pole, the iPhone did switch to the external microphone if the cable was connected. Without a resistor, the iPhone only recorded input from the internal microphone.

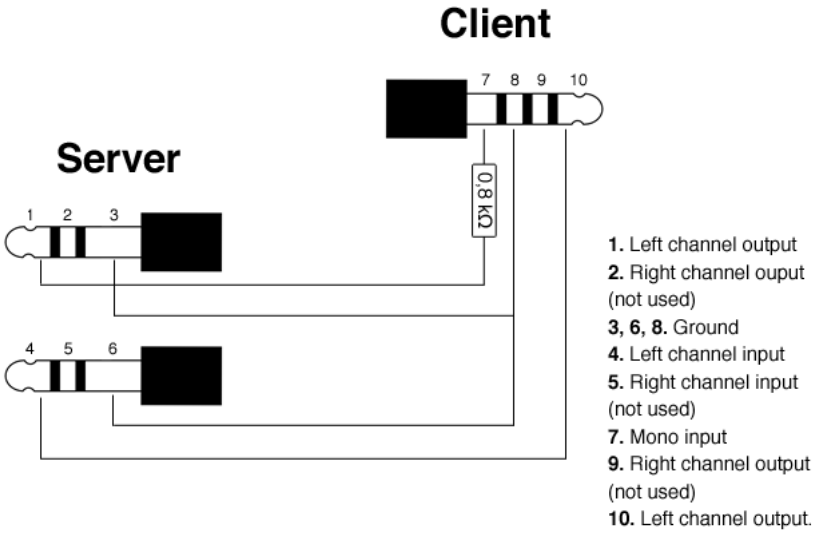

Figure 5.5: Schematics over the cable used as transmission medium.

Channel parameters

The frequency resolution of the channel is limited by the hardware being used. The highest frequency detectable on the Server side is 22 kHz which is close to the *Nyquist theorem* of half the sample rate (48 kHz). The audio hardware components on the Server acts as a low pass filter which filter out all frequencies above 22 kHz. In Apples technical specifications of the iPhone 3G it is written that the iPhone has a frequency response from 20 Hz to 20 kHz.

On both the Client and the Server, each sample is quantized to a 16-bit value, but on the iPhone all samples above ≈ 14000 are floored to 14000.

The SNR was measured for both the Server and the Client (the calculations are described in appendix B). Not surprisingly the Server had a better SNR, with 65 dB while the Client had a SNR of 44 dB.

By using the SNR measurements and the available bandwidth the theoretical highest possible date rate was calculated using the formula for Shannon capacity (see appendix B). For the calculations of the Shannon capacity the bandwidth $20\ 000 - 20 = 19\ 980$ Hz was used, which is the bandwidth constraint set by the Client. The resulting capacity was 53 kilobyte/s for the Server and 36 kilobyte/s for the Client. The values of the resulting capacities should be considered as an indication of what bit rate that might be possible on the channel and not a result proven with statistical certainty.

5.2 JPEG compression

The image file format used when receiving a picture from the image sensor on the camera is the lossy compression format JPEG. JPEG compresses the image data by removing the high frequencies in the color data of the picture. The resulting compressed image is then further compressed with a lossless Huffman encoding.

How much the high frequencies are compressed by the JPEG compressor is dependent on the parameter called the *compression value*. The compression value parameter ranges from 0 to 100, where 0 gives an uncompressed image.

The compression value used in this project is 70 and a resolution of $240x160$ pixels. The image size with these presets varies depending on the content of the image and can range from 2 500 to 3 700 bytes.

5.3 Obstacles

During the course of the project it has been a mystery as to why the amplitude on the iPhone gets floored to 14000. If it is possible to fix the problem and to use all 16-bits to describe the amplitude, more amplitudes could be used for PAM and the bit rate would be improved. Sometimes the amplitude gets cut off at an even lower threshold (see figure 5.6). It has not been determined what triggers this bug, which aggravates the bug fixing process. It might have

to do with the custom made cable that have been constructed without having access to any official documents from Apple describing how the audio input port detects input.

It has from time to time been frustrating to work with the bug described above. For example while testing new functionality and not being able to tell if the transmission fails because of the newly added code, or the amplitude-cutting bug, without saving the transmitted signal and looking at the waveform.

When the amplitude gets cut as shown in figure 5.6 the camera needs to be rebooted to temporarily fix the problem. It is still not known why the problem arises, or why it is fixed by rebooting the camera.

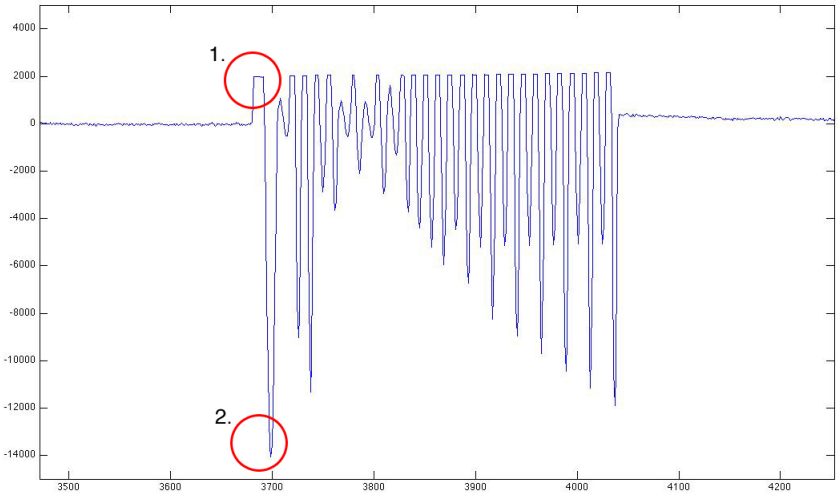

Figure 5.6: The signal when the amplitude cutting bug is present. Only positive amplitudes are affected, as can been seen at areas marked 1 and 2 in the graph above (the amplitude at 1 is supposed to be of the same magnitude as at 2).

Chapter 6

Analysis

In this chapter there are several references to figures which are located in appendix A and are presented there in landscape format.

6.1 Curve fitting with Catmull-Rom

In the first implementation of the PAM-algorithm no curve reconstruction was used, and the maximum sample of a symbol was chosen as r (the received amplitude of a symbol). This generated a systematic error (see figure A.2) which was eliminated by introducing the Catmull-Rom line fitting algorithm.

Without Catmull-Rom (on the Server) r varied periodically between $20\ 820 - 30\ 908$ and with Catmull-Rom r seems to have a normal distribution (see figure A.4 and A.3) with a mean of 30 881 and standard deviation of 9.13.

6.1.1 Performance

The Catmull-Rom spline described in equation (3.5) has been subject to some change during the implementation. Three different version were tested:

1. Floating point
2. Fixed point
3. Fixed point with lookup table

The three different versions was implemented and tested on the iPhone 3G (Client). To measure the CPU time used for the specific algorithm *Time Profiler* was used. Time Profiler is an application built into the development tool Xcode. Xcode is used to write, compile and debug iPhone applications and Timer Profiler is a tool which can be used to get statistics over how much time is spent in what functions of an application during run time.

The test, measured using the Time Profiler, was carried out by starting up the application, calibrating the Server and Client, sending a picture request to the Server and receiving a picture back to the Client. Below is the recorded CPU usage by the different versions of the Catmull-Rom function:

Implementation	Floating point multiplications	Fixed point multiplications	Type conversions	CPU time
Floating point	18	0	0	24%
Fixed point	0	18	13	26%
Fixed point with lookup table	0	4	6	1.6%

As seen in the table the lookup table version using fixed point was the quickest, what is more surprising is that the fixed point version was slower than the floating point version in the test. The reason for the poor performance of the fixed point version seems to be that all of the 13 values first have to be converted to libfixmath's own data type to perform the calculations. Since there is only 18 multiplications to perform, it is not worth the overhead to do the 13 type conversions from float to fixed point.

In the lookup table version the number of calculations was decreased by pre calculating the product of all factors except for a, b, c and d from equation 3.5 for $i \in [0, 0.01...0.99, 1]$.

```
int catmullrom_table (int16_t a1, int16_t b1, int16_t c1, int16_t d1, int i)
{
rational a = RATIONAL_FROM_INT(a1);
rational b = RATIONAL_FROM_INT(b1);
rational c = RATIONAL_FROM_INT(c1);
rational d = RATIONAL_FROM_INT(d1);

rational p = RATIONAL_FROM_INT(0);

p += MUL(a, catmull_lookup_table_a[i]);
p += MUL(b, catmull_lookup_table_b[i]);
p += MUL(c, catmull_lookup_table_c[i]);
p += MUL(d, catmull_lookup_table_d[i]);

return RATIONAL_TO_INT(p);
}
```

Figure 6.1: Implementation of Catmull-Rom with a fixed point lookup table in C.

6.2 Comparison of modulation configurations

An important part of a transmission protocol is the data throughput that can be sustained. To test the practical throughput of the designed protocol a test was set up.

The test measures the time from that a user sends a request from the Client until the requested data have been successfully received. The practical throughput is calculated as:

$$\frac{\text{size of requested data (B)}}{\text{roundtrip time (s)}} \tag{6.1}$$

By designing the test in this way the results will not just reflect the data rate on the channel, but all the steps from modulation, demodulation, receiving the data from the Server and round trip communication.

The time was calculated from when the Client send a request (3 bytes) for the data. When the Server receives the request, it modulates and sends 10 kilobyte of data back to the Client. Appended to the data as well as the initial request is a header of 5 bytes. When the 10 kilobyte message has been received and demodulated by the Client, the timer stops.

To put the measured time in a theoretical perspective, the *Theoretical Throughput* is also listed in the tables. The theoretical throughput was measured purely on the number of bytes per symbol and the baud rate, and does not take in to account any modulation or other time in the system.

$$\frac{byte}{symbol} \cdot \text{Baud rate} \tag{6.2}$$

Another measurement recorded during the tests was the *byte error ratio* which was calculated as:

$$\frac{\text{number of byte errors}}{\text{size of requested data (B)}} \tag{6.3}$$

The data that was sent as a sequence of uniformly distributed random data which was predefined and known by both the Server and Client. Since the data was known by both parts, it was possible to record the number of byte errors for each transmission.

There are a number of different parameters that can be configured to test the modulation;

1. **Amplitude levels:** The number of different amplitudes used by PAM. There is support for 2-PAM, 4-PAM and 16-PAM.
2. **Catmull type:** The different implementations of the Catmull-Rom type, either floating point, fixed point or fixed point with lookup table.
3. **Baud rate:** The frequency of the sine wave carrying the information.

6.2.1 Catmull-Rom type:

To see if the different implementations of the Catmull-Rom algorithm would have any impact on the throughput of the transmision, tests were carried out using the different versions.

Unit type	Byte error ratio	Amplitude levels	Catmull type	Baud rate (Bd)	Throughput (byte/s)	Theoretical Throughput (byte/s)
Client	1	16	None	2666	-	1333
Client	0	16	Float	2666	1173	1333
Client	0	16	Fixed Table	2666	1180	1333
Server	0	16	None	4800	2061	2400
Server	0	16	Float	4800	1636	2400
Server	0	16	Fixed Table	4800	2057	2400

In the table above it is evident that the version using the look up table was much faster than the floating point implementation for the Server.

For the Client it was shown that floating point implementation performed about as good as the fixed point with lookup table. The reason for this result probably is that the floating point arithmetic is better in the Client than the Server.

Another interesting detail from the table is that the Server still worked without any signal reconstruction, whereas the Clien was not able to even detect a single byte.

6.2.2 Amplitude level and symbol frequency

Tests were conducted for 2-PAM, 4-PAM and 16-PAM with variable symbol frequency. For each tested configuration, the test was carried out 3 times and the mean value of the throughput and byte error ratio was recorded.

Unit type	Byte error ratio	Amplitude levels	Catmull type	Baud rate (Bd)	Throughput (byte/s)	Theoretical Throughput (byte/s)
Client	0	16	Fixed Table	2000	891	1000
Client	0	16	Fixed Table	2400	1060	1200
Client	0	16	Fixed Table	2666	1180	1333
Client	0.0013	16	Fixed Table	3000	-	1500
Client	0	4	Fixed Table	4800	1062	1200
Client	0.0011	4	Fixed Table	5333	-	1333
Client	0	2	Fixed Table	6000	545	750
Client	0.0029	2	Fixed Table	6857	-	857
Server	0	16	Fixed Table	4000	1739	2000
Server	0	16	Fixed Table	4800	2057	2400
Server	0.0001	16	Fixed Table	5333	-	2666

The Server performed much better than the Client, this is mainly due to the fact that the amplitude gets cut on the Client and therefore has a considerable smaller range of possible amplitudes (as described in section 5.3).

For the Server the best setup was 16 amplitude levels and a symbol frequency of 4800 Hz, whereas the Client performs best at 16 amplitude levels and 2666 Hz in symbol frequency. Since it is the Client that is going to receive the largest amount of data, it is important to optimize with that unit in mind.

6.2.3 Frame size

The setup using 4 amplitude levels and a symbol frequency of 5333 Hz had a byte error ratio of 0.0011. A test was carried out to examine if a higher throughput would be achieved if the data was to be transmitted in several frames, which would lower the error ratio per frame. In the table below the throughput of sending 10 kilobyte of data with different frame sizes are shown.

Unit type	Frame size (bytes)	Amplitude levels	Baud rate (Bd)	Throughput (byte/s)	Theoretical Throughput (byte/s)
Client	1000	4	5333	414	1333
Client	500	4	5333	467	1333
Client	250	4	5333	246	1333

As dividing the data into several frames resulted in such poor throughput, it was not used in the final implementation. In the current configuration data bigger than 10 kilobyte will be divided into several frames. Images sent from the Server are usually < 10 kilobyte (see section 6.3) and thus data will almost never be divided into more than one frame.

6.3 JPEG quality

Receiving a picture from the Server takes about 3 to 3.5 seconds with a compression number of 70 and a resolution of $240x160$ pixels (see figure 6.2 and section 5.2). The received picture is good enough for telling where the camera is pointed, but does not contain enough detail to use for example to set the focus. If video was to be sent as series of JPEG images, the frame rate would be about $\frac{1}{3}$ frame/s, which is way to slow to even be considered as video.

Figure 6.2: Example picture taken with a compression of 70 and $240x160$ pixels resolution

Since the protocol is too slow to send images with high quality, it is interesting to consider other possible solutions which have smaller requirements on the bit rate. One example of this is to take a picture on Server and then only send the image size to the user. Guided only by this value the user can then regulate the focus, and since there is very little data to send, get a response with a new file size, almost instantly. The image size that the user receives is an indication of how sharp the picture is. This is because of the fact that the image size increases when there is much high frequency content in the JPEG picture. Because of the correlation between the file size of JPEG-image and the sharpness of the image, a user can use the file size to to get an indication on if the image gets sharper, without actually having to look at the image. When the focus has been set using this technique, as a final step an image with higher quality could be sent to the user to verify that the focus is correctly configured.

This can of course be done automatically if the camera has a digital focus setting, unless the camera has a lens that has to be set manually. In that case the user needs some indication of where the focus is to be able to properly set it.

Chapter 7

Conclusions

7.1 Results

A proof of concept protocol capable of sending digital data between the audio ports of an iPhone 3G and an Axis P3367 has been successfully implemented. The modulation method used is 16-PAM and the configuration yielding the best throughput is presented in the table below:

Unit type	Byte error ratio	Amplitude levels	Catmull type	Baud rate (Bd)	Throughput (byte/s)
Client	0	16	Fixed Table	2666	1180

1180 bytes/s is enough to send a low quality JPEG picture with the resolution $240x160$ with a transmission time of about 3 to 3.5 seconds. The bit rate is too low to send video.

Using Shannons formula for channel capacity, the theoretical highest possible bit rate for sending data to the iPhone was calculated too 36 000 byte/s. Since it is a big difference between the achieved bit rate and the theoretical max, there is a lot of room for improvements (some suggestions are described in section 7.2)

In the problem description in section 1.2, three main goals for the project are presented. Two of the three goals, *Duplex communication* and *Sending images*, were met. The third goal, *low frame rate video*, was not met, but could probably be accomplished if project was further developed.

7.2 Discussion

The most important goal of the project was to create a proof of concept, to show that it was possible to send data between an Axis device and a smartphone using the audio ports, and also to roughly show what bit rate that is to be

expected. A fully working, but unoptimized prototype has been desired rather than a non-working prototype with some heavily optimized parts. Of course a fully working, optimized prototype is the most desired, but the lack of time has made it necessary to forgo certain parts.

In this chapter problems with the current implementation as well as possible improvements will be discussed.

7.2.1 Modulation

The modulation technique used, PAM, gave sufficient results with regards to implementation complexity and performance, but could be improved further.

As described in section 5.3 the amplitude of the Client gets cut at a lower value than on the Server. This sets a big constraint on the current implementation of PAM. It has not been possible to match the same baud rate as the Server, even when the Catmull-Rom algorithm is used measure more exact values of the amplitude.

According to the sampling theorem it should be possible to fully reconstruct a signal as long as the sample rate is twice the highest frequency in the original signal. The sample rate used in the implementation was $48kHz$, but the highest baud rate which generated error free communication was only $6000Hz$ (see section 6.2.2). This suggests that the signal reconstruction using Catmull-Rom is far from perfect. In an attempt to save processor power the Catmull-Rom only operates on small segments off the signal, where the amplitude of a signal is measured. It has not been tested to reconstruct the entire signal.

In section 1.4 a similar project to this is briefly presented. The project used a simple form of Manchester encoding and still managed to get a bit rate of about the same as the one achieved in this project. The reason that a Manchester based encoding was not implemented during this project was because the possible bit rate in using a more complex modulation technique would surpass bit rate of the Manchester encoding.

7.2.2 Flow and error control

The flow control logic incorporates frame numbering and a checksum making it easy to find errors in the transmission. There is however no correction if an error is presented, only resending of the effected frame. If error correction codes were to be implemented the baud rate could be increased for baud rates in the current implementation where the byte error rate is reasonably low.

Splitting up long frames to reduce the error rate was tried, but did not produce results that could compete with longer frames and a modulation configuration with lower error byte ratio.

7.3 Future work

A lot of measures could be taken in order to improve the implementation, below is a list of the actions that would be taken if the project was to be further developed.

- Investigate further how reconstruction with Catmull-Rom algorithm compare with other algorithms used for signal reconstruction, such as cross correlation with oversampled symbols or least squares approximation, with regards to speed and precision.
- The symbol used in the current implementation is a one period long sine wave. The pulse was chosen mainly because it was easy to implement. It would be very interesting to compare the use of a sine wave as a pulse with, for example, a raised cosine. The biggest advantage with the raised cosine is that it eliminates ISI which could open up for more advanced modulations.
- One way of improving the modulation with regards to the number of bytes per symbol could be to implement QAM. Using QAM would introduce abrupt changes in the signal where to symbols with different phase follow each other. If a raised cosine filter would be implemented in combination with QAM, this effect would be countered since the raised cosine filter will filter out high frequency components.
- Performance tuning with regards to the number of bits per sample. In the current implementation of PAM there is only support for 2,4 and 16 amplitude levels. For example a 8-PAM or 32-PAM implementation might outperform the current configurations.
- Research regarding implementation of error correction codes in the data link layer and the possible gains, or lack there of, of such an approach.
- Making the initial calibration state also determine the modulation configuration. In the current state the modulation configuration has to be manually set on the Client and Server in order to work.
- It would be possible send video over the channel with a lower bit rate if the video was better compressed than sending a sequence of JPEGs. On the Sever there is support for streaming video with the format H.264. H.264 would be well suited for this project, because it only sends information about the pixels that has changed since the last frame, leading to a lower bit rate compared to sending a series of JPEG pictures. Implementing a H.264 encoder for the Client would lower the constraint on the bit rate needed to send video.
- Extensive testing of all the components in the system, including testing the protocol on different devices.

Appendix A

This appendix contains figures and tables that was too big to fit together with the related text in the report.

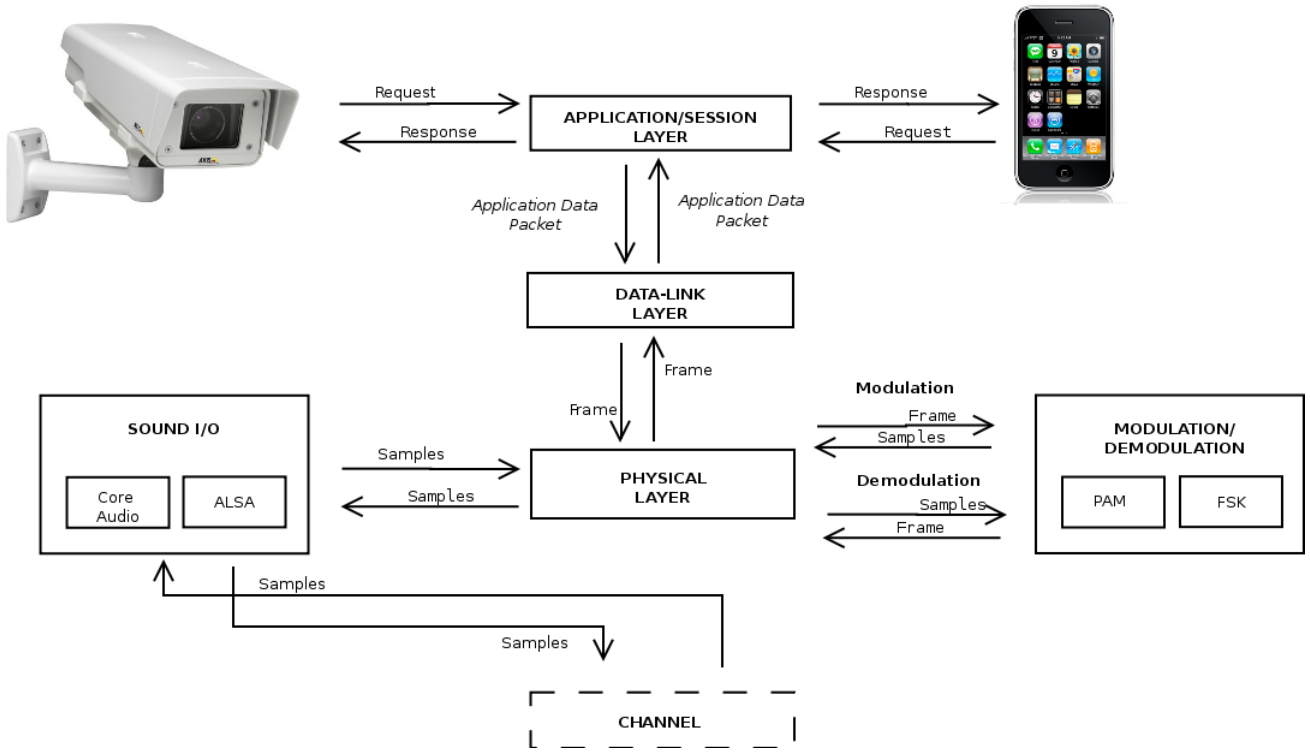

Figure A.1: A diagram showing the overall structure of the system. Each block shows which other blocks it communicates with as well as the type of data sent between the blocks.

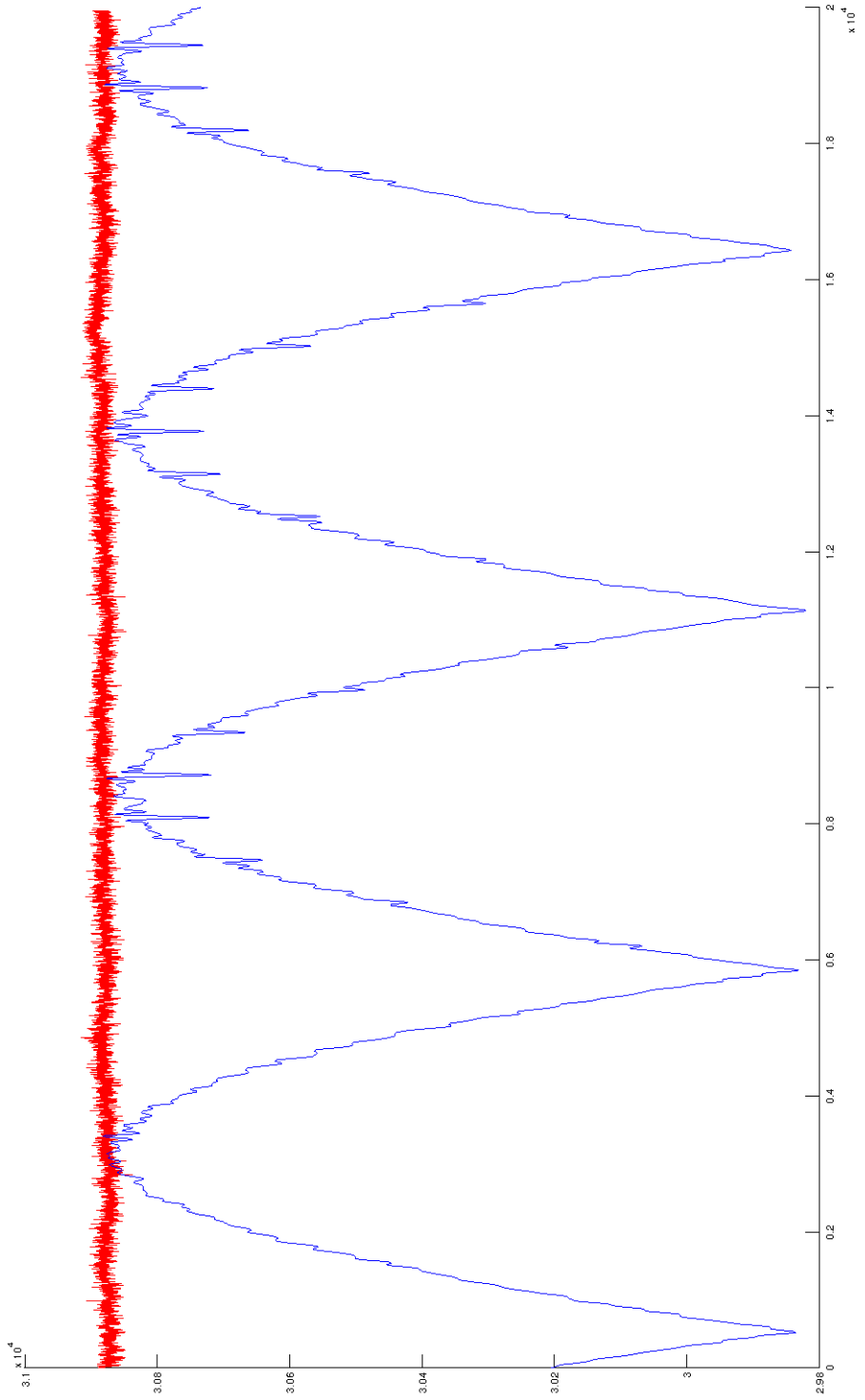

Figure A.2: A symbol with amplitude s was sent over and over again and every received amplitude r was stored. The plot shows all r:s collected using Catmull (top red graph) and without (bottom blue graph).

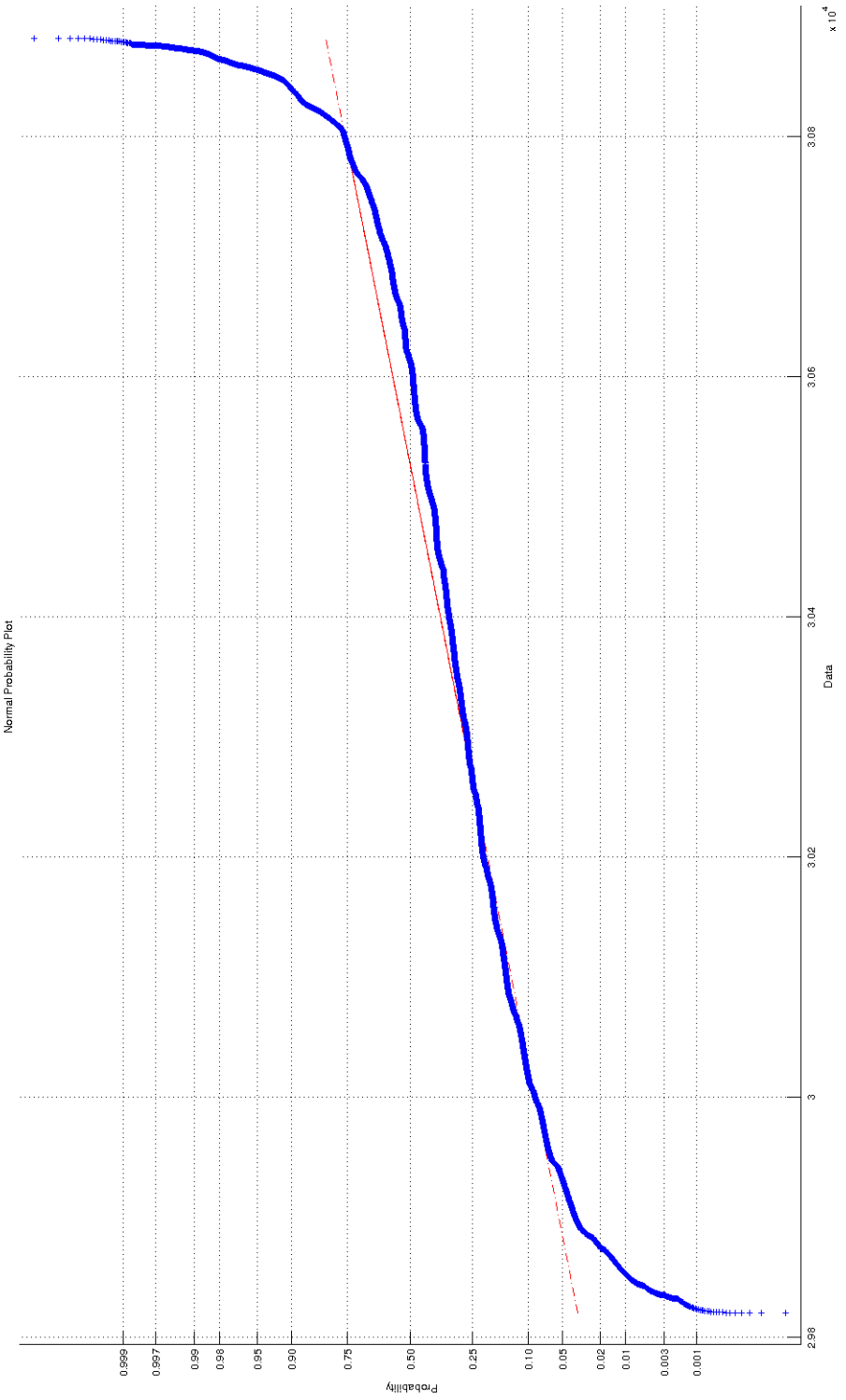

Figure A.3: Probability plot of the measured data points without Catmull.

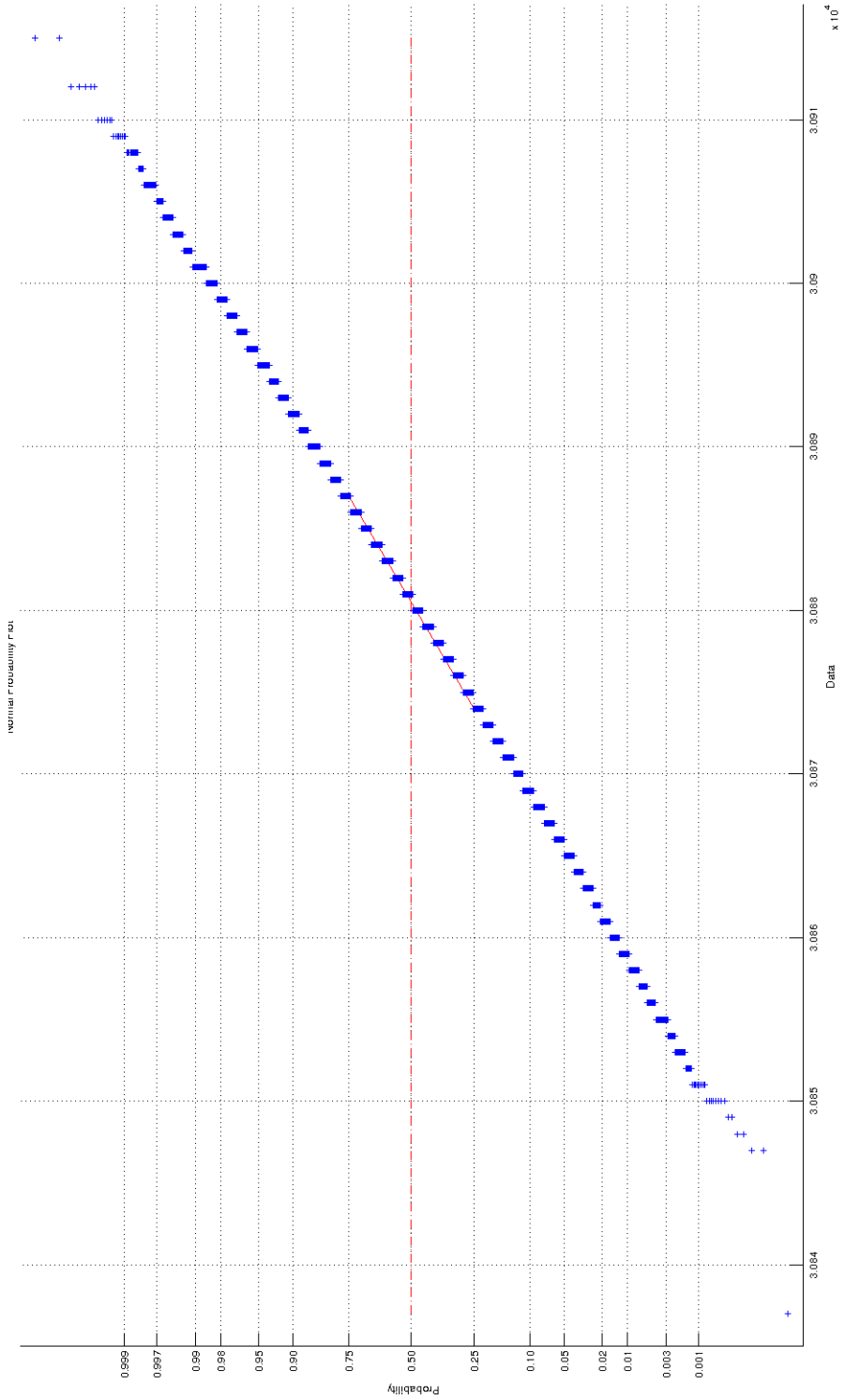

Figure A.4: Probability plot of the measured data points with Catmull.

Appendix B

SNR and Shannon Capacity calculations

A sine wave with frequency 12 $000Hz$ was sent over the channel and recorded by the testing device (Server and Client). From the sampled signal two sequences, S_1 and S_2, with the length of L samples was chosen. S_1 and S_2 were chosen to contain two sampled version of identical segments of the analog sine wave. From S_1 and S_2 the noise power (N_p)was calculated as:

$$N_p = \frac{1}{2L}\sum_{k=1}^{L}(S_1(k) - S_2(k))^2 \tag{B.1}$$

The signal power was then calculated as S_p where $S = S_1$:

$$S_p = \frac{1}{L}\sum_{k=1}^{L}S_1(k)^2 \tag{B.2}$$

The signal to noise ratio and Shannon capacity was then calculated as described in section 2.8 and 2.7. The results of the calculation are presented in the table below:

Unit type	Signal to noise ratio (dB)	Shannon capacity (kilobyte/s)
Server	65	53
Client	44	36

The value of the resulting capacity should be considered as an indication of what bit rate that might be possible on the channel and not a result proven with statistical certainty. The signal to noise ratio was measured for only one frequency and the sampled signal used in the measurement might be subjected to timing jitter and quantization errors, which could have contaminated the measurement.

References

[1] *Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface*, Ye-Sheng Kuoi, et. al.http://web.eecs.umich.edu/~prabal/pubs/papers/kuo10hijack-islped.pdf, 2012-06-16

[2] *Square skewed*, http://cosmodro.me/blog/2011/mar/25/rhombus-square-iskewedi/, 2012-06-16

[3] *PCM Tutorial*, http://telemetry-products.com/sites/default/files/PCM_Tutorial.pdf, 2012-01-31

[4] *PCM Tutorial*, http://telemetry-products.com/sites/default/files/PCM_Tutorial.pdf, 2012-01-31

[5] *Digital Communications - Signal Processing* - chapter 6, http://www.stanford.edu/group/cioffi/book/chap6.pdf, 2012-01-31

[6] *Digital Communications - Signal Processing* - chapter 1, http://www.stanford.edu/group/cioffi/book/chap1.pdf, 2012-02-29

[7] *Introduction to Digital Communications*, Göran Lindell, August 2006

[8] *Data Communications and Networking*, Behroux A. Forouzan, Fourth Edition

[9] *Interactive Computer Graphics*, Edward Angel, Fifth Edition

[10] *Image Analysis, Lecture 2, Lund University*, Magnus Oskarsson, http://www.maths.lth.se/media/FMA170/2011/f02_.pdf, 2012-06-16

[11] *Numerical Analysis, Lecture 11, Lund University*, Carmen Arevalo, http://www.maths.lth.se/na/courses/FMN011/media/material/fmn011-11-11__.pdf, 2012-06-16

[12] *Digital Signal Processing - Principles, Algorithms and Applications*, John G. Proakis, Dimitris G. Manolakis, Fourth Edition

[13] *Fotograf BETA75*,
`http://www.beta75.se/wordpress/?p=442`, 2012-04-18

[14] *Real Time Systems, Lecture 11, Lund University*,
Karl-Erik Årzen, `http://www.control.lth.se/media/Education/EngineeringProgram/FRTN01/2011/L11_11_Multi.pdf`, 2012-04-18

[15] *The care and feeding of digital pulse-shaping filters*,
Ken Gentile, `www.feng.pucrs.br/~decastro/pdf/ShapingFilterDesign.pdf`, 2012-06-11

[16] *Inter Symbol Interference (ISI) and raised cosine filtering*,
Charan Langton, `www.complextoreal.com/chapters/isi.pdf`, 2012-06-12

[17] *Pulse-Shape Filtering in Communications Systems*,
National Instruments, `http://www.ni.com/white-paper/3876/en`, 2012-06-12

[18] *iPhone 3G - Technical Specifications*,
Apple, `http://support.apple.com/kb/SP495`, 2012-06-19

[19] *ALSA Programming HOWTO*,
Matthias Nagorni, `http://www.suse.de/~mana/alsa090_howto.html`, 2012-04-18

[20] *Fletcher's Checksum*,
John Kodis, `http://www.drdobbs.com/database/184408761`, 2012-04-23

[21] *Core Audio Essentials*,
Apple Inc, `http://developer.apple.com/library/ios/#documentation/MusicAudio/Conceptual/CoreAudioOverview/CoreAudioEssentials/CoreAudioEssentials.html`, 2012-06-10

[22] *libfixmath*,
Ben Brewer, `http://code.google.com/p/libfixmath`, 2012-06-14

[23] *Kiss FFT*,
Mark Borgerding, `http://sourceforge.net/projects/kissfft/`, 2012-06-19