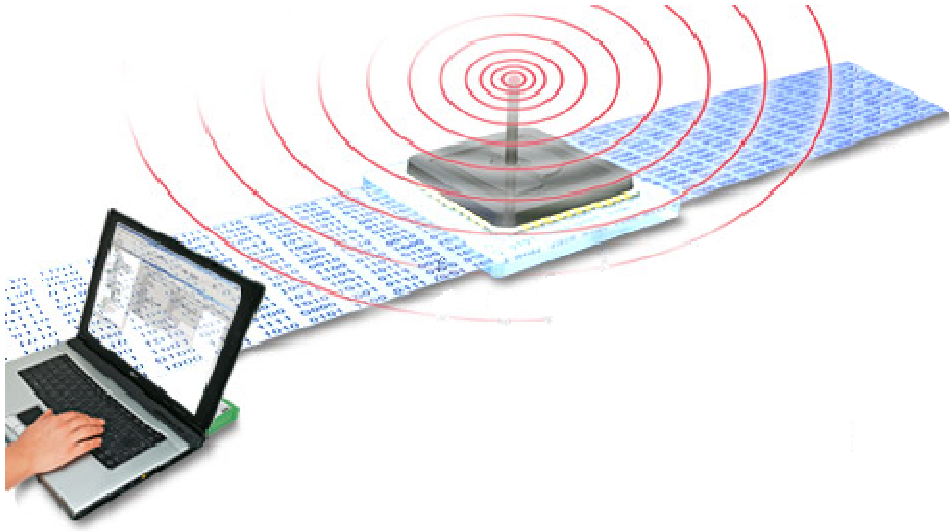
Design of a wireless data transmission system for Super E-paper

Implementing and testing the software and hardware

CONFIDENTIAL



M.D. de Jong
C.J. Kruit

Delft, June 2010.

Design of a wireless data transmission system

Implementing and testing the software and hardware

M.D. de Jong - 1354647
C.J. Kruit - 1354655

Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science
Mekelweg 4
2628 CD Delft
Tel: +31 (0)15 27 84568
ewi.tudelft.nl

Preface

As students Electrical Engineering from the TU Delft, we have had the opportunity to use our theoretical knowledge in practice for the last two months within the scope of the Bachelor thesis. Our goal was to create software and hardware to transmit an image from a computer to a Super E-paper wirelessly and test the software and hardware.

In this report, we expect the reader to have some basic knowledge of microcontrollers and RF communication. Readers who are mainly interested in creating the software are referred to chapter 4. If only the hardware is of interest, chapters 5 and 6 should be read. Chapters 7, 8 and 9 are especially interesting for readers who are interested in testing the software and hardware.

We would like to thank the following persons for their help with this thesis:

- Ryoichi Ishihara, for his guidance during the project.
- Jaber Derakhshandeh Kheljani, for his help with RF communication.
- Martin Schumacher, for his support during the development of the transmission kit.
- Willem Zwetsloot and Joost van Meerwijk, for developing the testing facility.
- Gerard Janssen, for his information on RF communication.
- Wim Blokzijl, for improving the structure of this thesis.
- All the participants in the test group, for testing our software.

Delft, June 11 2010
M. D. de Jong
C.J. Kruit

Summary

Whether it is possible to create a compact and user friendly kit in order to wirelessly send an image to a Super E-paper device, is investigated throughout this thesis. DIMES and the Compartment Engineering are developing a Super Electronic Paper Display (Super E-paper). The Super E-paper is an improved version of the old E-paper. The Super E-paper will have a paper-like appearance and will be in full colour. Next to this improvement it will also be able to integrate fully flexible hardware, for instance a solar cell and a Radio Frequency module.

The goal of our research is to determine whether it is possible to create a kit that uses *one-way radio-frequency communication* to transmit an image without loss of quality to a receiver connected to a Super E-paper, which is controlled by a *user-friendly* and *very simple* computer program. One-way communication is used, so one image can be 'broadcasted' to multiple E-papers. Trade offs are made which hardware components are used to get the smallest amount of power consumption and still have a high quality and error free wireless communication system. The desired kit consists of the following three components:

- An easy to use computer program to transmit an image
- A transmitter
- A receiver

We developed such a kit which uses the *low-budget radio-frequency module* nRF24L01+ developed by Nordic. Even though these modules support two-way communication, only a *one-way communication network* is used to determine whether this is possible.

A wireless transmission is never without errors. Therefore Error Control methods have to be used. Two methods are implemented in this system namely: a Cyclic Redundancy Check and a Repetition Code. This combination enables the RF modules to transmit an image with very high quality when the Bit Error Rate is $3{,}2 \times 10^{-3}$ or better.

Because the modules need to communicate with a computer and a Super E-paper, an Arduino Duemilanove Board is used as a bridge to the RF module. We chose a microcontroller for this function, because it simplifies the implementation of error control and it offers more testing capabilities.

To be able to send any kind of format of a picture or a PDF file, *a user friendly program* is designed in MATLAB. The program is capable of either sending a single picture or a slide show and is tested in two stages. The first test group was focused on the functionality where after a second group tested the revised version on user friendliness. Eventually an easy to use and bug free program is developed.

Tests have also shown, by improving the Quality factor and the speed of the transistors used to create the circuit, theoretically the receiver can be implemented onto the Super E-paper. Changing the receiver will not affect the rest of the developed kit and can therefore still be used.

Even though the used modules can only guarantee a high quality transfer within a radius of 28 meters, the developed kit does prove that one-way communication can be used in order to successfully transmit an image to a Super E-paper. By replacing the antenna of the transmitter with one that has a higher gain, the radius can easily be increased. Another possibility is to transmit with more power. Both changes will not change the outcome of this thesis and the rest of the kit can still be used. Theoretically it is also possible to implement the receiver onto the Super E-paper without losing compatibility with the developed kit.

Table of Contents

1. Introduction

The advertisement sector is eager to replace paper posters with electronically changeable E-papers. DIMES and the Compartment Engineering are developing a Super Electronic Paper Display (Super E-paper). The Super E-paper is an improved version of the old E-paper. The Super E-paper will have a paper-like appearance and will be in full colour. Next to this improvement it will also be able to integrate fully flexible hardware, for instance a solar cell and a Radio Frequency module.

Super E-paper is perfect for the advertisement sector. Advertising companies have to replace their posters very often, because their customers only rent them for a few days or weeks. If the advertisement companies were to replace their posters with Super E-paper, they wouldn't have to replace the posters anymore. Instead, they could simply refresh them every week with a new image. This way, they wouldn't have to waste money on paper, ink and distribution of the posters, which could easily save them hundreds of thousands of euro's a year.

Our studies [1] have shown that advertisement companies would like to use E-paper only indoors, for example in shopping malls, because there is very little vandalism in these malls. If they would use E-paper outdoors, vandals would break them. This would only increase costs for these firms. Because E-papers would be mostly used in malls, users need to transmit their advertisement images to the E-papers within a close range. This creates the need for a sending module, which allows an advertisement company to change the image on their E-papers. Accompanying software should make it possible to do this from a computer, connected to a transmitting module.

It would be useful to use one-way communication for this purpose. One-way communication means data is transmitted from a location to receiving location, but the receiver can't send data back to the transmitter. This is for example the case in television broadcast: Television shows are broadcasted from one location to all houses who can receive them. This would be useful for transmitting images, because one transmitter could change the screen of all E-papers in a shopping mall at once by 'broadcasting' the image to all E-papers.

The goal of our research is to determine whether it is possible to create a kit that uses *one-way radio-frequency communication* to transmit an image without loss of quality to a receiver connected to an E-paper, by using a *user-friendly* and *very simple* computer program. Such a kit would make it cost-effective and simple for an advertisement company to change the images of their posters by simply selecting an image on their computer, and transmitting it to their E-paper(s). We will develop such a kit, consisting of these components:

- A computer program, where the user can select an image to transmit.
- An RF transmitter.
- An RF receiver

In our research, we will investigate to what extent the computer program is user-friendly and how much quality is lost by transmitting an image using radio frequency (RF) modules. The first part will be investigated through a test group. All participants in this test group have to install the program on their computer and use it to transmit an image. A postal survey will show if the test group found our program user-friendly.

The second part will be investigated by reading the received data with a computer. By comparing the received data to the transmitted data, an *Error Rate* could be calculated. A low Error Rate means the quality of the received picture is still very high, where a high Error Rate

implies a very low quality. If the Error Rate is very high, we will investigate how to improve this.

There are a few very important requirements on the software. Mainly, the software should be very easy to use. The user should not be able to make mistakes which cause errors in the program. Also, the software should work on many different computers. Since most computers are operated by Windows, we will make the software compatible with Windows computers.

The main requirement on the hardware is the ability to transmit an image with one-way communication without the loss of quality. This is very difficult to achieve, because the receiver can't inform the transmitter if some data isn't received. That data loss will immediately cause loss of pixels and thus loss of quality. This has to be prevented.

This report is composed of the following chapters. Chapter 2 will describe the requirements and limiting conditions of this research in more detail. Chapter 3 will explain techniques for controlling errors in wireless communication. In chapter 4 it will be determined which port is best to use for communication between the computer and the transmitter module in this research. We will explain the working of the transmission software in chapter 4. Chapter 5 will describe how the RF modules work. To communicate with these modules, a microcontroller is used to form the bridge between the computer and the RF modules. The microcontroller will be discussed in chapter 6. Chapter 7 will investigate the user-friendliness of the software. In chapter 8 will be investigated how the quality of the transmission can be improved with the techniques for controlling errors, discussed in chapter 3. The results of this chapter are further investigated with a simulation of a Super E-paper in chapter 9. Finally, chapter 10 will discuss the results of our research and make recommendations for future development and investigation.

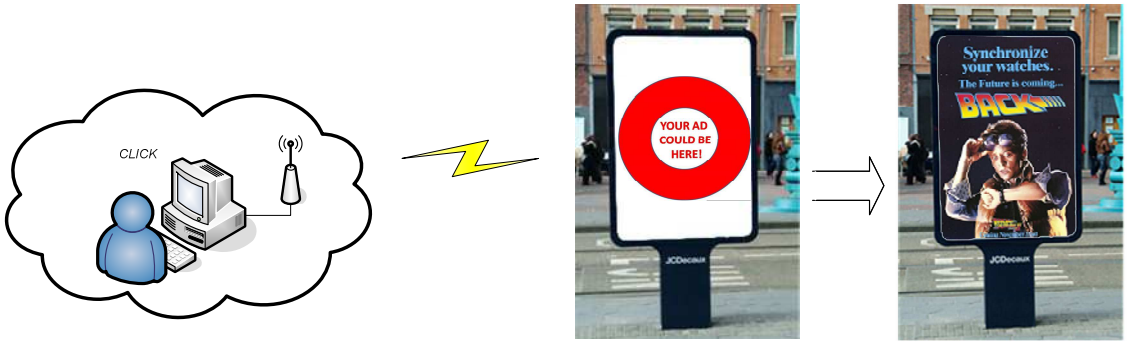


Figure 1.1: A possible setup for transferring a picture.

2. Requirements

The advertisement sector is eager to replace paper posters with electronically changeable E-papers that can be updated from a distance. We will develop a kit that uses one-way radio-frequency communication to transmit an image without loss of quality to a receiver connected to an E-paper, which is controlled by a user-friendly and very simple computer program. In order to make well-grounded decisions for developing this kit, all requirements on the system should be known. This chapter will describe these requirements on both the software and the hardware.

2.1 Requirements on the software

The software should meet the following requirements:

- The software should be *easy* to use.
- It should allow the user to select *any* image or PDF file on the computer.
- The software has to *transmit* the image or PDF to (an) E-paper(s) when the user wants.
- The software needs to *resize* the image to the right format.
- *Borders* should be added to the image if the user provides an image with very different proportions. Adding borders will maintains those proportions.
- The E-paper can be hung 2 ways: with the longest side vertically or horizontally (*orientation*). The software should be compatible with both ways.
- A *preview* area should be embedded in the software to show how the image will be displayed on the E-paper. The preview area must show the right orientation and the optional borders.
- It must be able to send *multiple pictures* in succession. The user should determine the time between the presentation of two images.
- The program must inform a user about the *progress* during the transmission.
- The user must be able to *abort* the transmission at any time.
- The software should access a *port* (for instance a serial, parallel or USB port) to write data to a transmitter.
- The software must convert the image into an RGB format, so the receiver does not need to perform any more operations on the received data.

2.2 Requirements on the hardware

The hardware should meet the following requirements:

- The hardware should be able to *read* the data coming from the computers port.
- The hardware should transmit this data wirelessly from one location to another using one-way communication.
- The hardware should be able to send an image without loss of quality.
- The hardware should work from the moment it is connected to the computer.
- Interaction with the software should not require user interaction.
- The data must be delivered in a special *order* and via a serial *UART* connection.

2.3 Requirements on the overall system.

The total system should be simple in use and send high quality images to the E-paper.

3. The different Error Control methods

Wireless connections are far more sensitive to external influences than wired connections. External influences could be other transmitting devices or weather conditions. Because of these influences the wireless connection will suffer from data losses. To be able to set up a reliable connection, some kind of Error Control method has to be implemented.

Paragraph 1 will evaluate a Automatic Repeat request techniques, useful in two-way communication. Paragraph 2 will discuss Forward Error Control, which can be used in one-way communication. Paragraph 3 compares the advantages and disadvantages of a one and two-way wireless connection. Finally, paragraph 4 will explain which Error Control method was used throughout this research.

3.1 Automatic Repeat reQuest techniques

In Automatic Repeat reQuest (ARQ) the receiver will send an acknowledgment for correctly receiving a data block, or request the transmitter for a retransmission otherwise. In order to decide whether the data is correct, the receiver has to be able to check it. That's why some Error Detection bits need to be added.

First we discuss two different ways to add Error Detection bits where after we discuss three ARQ protocols (a more detailed description about these topics can be found in [2]).

3.1.1 Adding a Parity Bit

Description
One of the simplest ways of encoding data is by adding a Parity Bit. A Parity Bit is a bit which is added to the data and checks to number of bits with the logical value '1'. There are two different ways of determining whether to set this bit or not. You can either set the Parity Bit when the sum of bits with value '1' is even or odd. This way the receiver can check if the data is 'right' or 'wrong' and can send an acknowledgement or a request for retransmission to the transmitter.

Disadvantages
Although this method is widely used and very easy to implement, it is not capable of detecting every occurring error. Summing all bits with value '1' can only detect an odd number of errors, because the Parity Bit will not change if an even number of errors occurs. In that case, the receiver will think the data is correct.

3.1.2 Adding a Cyclic Redundancy Check

Description
A different method of encoding data is by adding a Cyclic Redundancy Check (CRC). A CRC code is a bit more complicated than the Parity Bit.

First, the transmitter and receiver will determine a fixed series of bits with which they will divide the data before transmitting. These bits are called the CRC code. Using bitwise division a 'Remainder' will remain. Secondly, this Remainder will be added to the original data and transmitted. Finally, after receiving the data the Remainder will be calculated again and compared to the transmitted Remainder.

Disadvantages
Even though this method is capable of detecting multiple errors, also this technique can't exclude them all. For example, the CRC will be wrong whenever the data is correctly transmitted but the Remainder contains an error. Another situation in which the CRC could be wrongly evaluated is when the data and Remainder contains errors. In this situation it is possible the corrupted data still holds a valid Remainder and thus won't be detected as wrong.

3.1.3 Stop-and-Wait protocol
The Stop-and-Wait protocol is a very intuitive way of sending and receiving data. The transmitter will send his data and waits until, either he gets an acknowledgment or a time-out occurs (this happens whenever it took too long to get an acknowledgment). Upon arrival of an acknowledgment the transmitter will send new data.

This method looks a lot like a simple conversation between two people. When one of them asks a question he'll wait for a respond and if it takes to long he will probably ask it again.

Whenever the data is correctly transmitted but the acknowledgment is not, the transmitter will send the same data again. Because the receiver can't distinguish whether the acquired data is new or old, it will always process the data as new. To get rid off this problem, including a single-bit sequence number will suffice.

3.1.4 Go-Back-N protocol
A Go-Back-N protocol is a bit more difficult than the Stop-and-Wait protocol. The receiver includes within the acknowledgment the number of the next expected data block. Unlike the Stop-and-Wait protocol this protocol does not wait for the acknowledgments before sending new data.

Though the Go-Back-N protocol does not wait for the acknowledgments it does use them as confirmations. When a data block is sent, a timer is set and is terminated by the acknowledgment of this block. In the mean time the transmitter will keep on sending new data blocks with a predefined window, which is the maximum amount of data blocks that are send before receiving any acknowledgment. Whenever a timer runs out, because the right acknowledgment is not yet returned, the transmitter will "go back" and starts with transmitting the unacknowledged data block.

The disadvantage of this protocol is that the data blocks sent after an error has occurred have to be retransmitted as well.

3.1.5 Selective Repeat protocol
The Selective Repeat protocol is almost the same as the Go-Back-N protocol. Two things are essentially different. First, the receiver does not only store the data blocks that are sequentially received but all the error free data blocks. Which brings us to the second difference: only the data blocks that weren't correctly received are retransmitted.

3.2 Forward Error Correction techniques

Forward Error Correction (FEC) techniques encode the data in a way the receiver can detect and correct any errors that occurred during transmission. Because these methods don't need any acknowledgments it is very useful in a one-way connection.

Encoding data often means adding extra bits to the data stream. In this case these extra bits will help the receiver to detect and correct any errors in the data. There are many different

ways to encode data with FEC techniques. Three of the most used techniques are described in this paragraph.

3.2.1 A Repetition Code

Description
Repetition Codes are one of the simplest encoding schemes. As the name suggests, the transmitted binary code is repeated. This can be done in to ways. One-way is to transmit every byte multiple times. The second way is bitwise multiplication, which means that every bit separately is send over more than once, before transmitting the next bit. For example, the data block containing "10011010" can be send over as "111-000-000-111-111-000-111-000" if the multiplicity is equal to three.

Disadvantages
This way a single error can be detected and corrected within every triplet. It isn't hard to see that if you raise the number of the bitwise multiplicity, it is possible to correct more errors. Though this sounds rewarding to simple raise the number of bits to cope up with a higher Bit-Error-Rate (BER), you'll have to consider the fact that a longer data block needs a wider bandwidth or more time to be send over. See also [3].

3.2.2 The Single Error-Correcting Hamming Code

Description
Error Correction Hamming Codes can be described using matrices. In order to correct a single error the minimum distance of the correction code must be three (also see [3]).

The principle of this encoding scheme is as follows. First construct a parity check matrix. Its size depends on the data block size and can be calculated with: $n \geq k + \log_2(n+1)$. Here *n* will be the number of columns in the matrix and *k* is the data block size. Having calculated the column size the number of rows is easy to find: $number\ of\ rows = n - k$. The first *k* columns are arbitrarily chosen with the restriction they are nonzero, unique and filled with at least two ones. The last $n - k$ columns are filled with the identity matrix.

When the parity check matrix is chosen a *generator* matrix is made. The first *k* columns are the identity matrix where after the transpose of the parity check (without the identity matrix) is placed. Two possible matrices are stated in figure 3.1 for an eight bit data block.

$$H^T = \left[\begin{array}{cccc} 1&1&1&0\\ 0&0&1&1\\ 1&1&0&0\\ 0&1&1&0\\ 1&0&1&0\\ 1&0&0&1\\ 1&0&1&1\\ 0&1&1&1\\ \hline 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 0&0&0&1 \end{array}\right] \qquad G = \left[\begin{array}{cccccccc|cccc} 1&0&0&0&0&0&0&0&1&1&1&0\\ 0&1&0&0&0&0&0&0&0&0&1&1\\ 0&0&1&0&0&0&0&0&1&1&0&0\\ 0&0&0&1&0&0&0&0&0&1&1&0\\ 0&0&0&0&1&0&0&0&1&0&1&0\\ 0&0&0&0&0&1&0&0&1&0&0&1\\ 0&0&0&0&0&0&1&0&1&0&1&1\\ 0&0&0&0&0&0&0&1&0&1&1&1 \end{array}\right]$$

Figure 3.1: A possible transpose parity check matrix (H^T) and a generator matrix (G).

The generator matrix is used by the transmitter to encode the data blocks. This is done by multiplying the data block with the generator matrix. For example see figure 3.2.

$$D = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad E = DG = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 3.2: The original data block (D) and the encoded version (E) which is acquired by multiplying the original data block with the generator matrix (G).

The parity check matrix is used by the receiver to decode the incoming data blocks. Just as encoding, decoding also uses matrix multiplication to be able to determine whether the received data is correct or how to modify if not. This is done as shown in figure 3.3.

$$R = E = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad S = RH^T = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} = 0$$

Figure 3.3: The received data block (R) and the outcome of the multiplication of the received data block with the transpose parity check matrix (S) which is zero when no errors have occurred.

If the outcome of *S* is not equal to zero, an error has occurred. Comparing this result to the rows of H^T the position of the error bit can be found. If *S* equals the *i*th row an error occurred at the *i*th bit of the received data block. This is illustrated in figure 3.4. More information can be found in [3].

$$H^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$S = RH^T = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}$$

Figure 3.4: The received data block (R) has an error. The error bit can be determined by calculating S and determining which row in H^T has the same value. The number of the row for which this is true, is the number of the bit that is wrongly received (in this case the highlighted bit).

Disadvantages
Just like the Repetition Code this encoding scheme also needs a larger bandwidth when more than one error needs to be detected and corrected and it will take more time to transmit.

3.2.3 Binary Cyclic Encoding

Description
Binary Cyclic codes are a subclass of linear block codes as the one mentioned above. Even though these methods are more complicated than linear block codes, it does have advantages.

Because Binary Cyclic codes are complex but have a mathematical structure, more sophisticated Error Correction methods can be created. Another great advantage is that

encoding and decoding are easily realised using simple shift registers with feedback connections.

Disadvantages
Though Binary Cyclic codes do have some advantages it is still too complex for easy implementation in our software and beyond the scope of this thesis. More information about Binary Cyclic codes can be found in [3] and [4].

3.3 Using one-way or two-way communication

As explained in the previous paragraphs, ARQ is a useful technique if two-way transmission is possible. One-way transmission requires the FEC techniques. But what are the main advantages and restrictions of the two different transmission systems? This question will be answered in this paragraph.

3.3.1 The two-way transmission system
In order to have a two-way transmission both the transmitter and the receiver should be capable of transmitting and receiving data. When these two options are combined into one device, this device is called a transceiver. Having such a system enables the use of the ARQ methods, like the ones stated above.

The big advantage of an ARQ method is when an error is detected a request for retransmission is returned. When the data is correctly received the transmission will go on. Here also lies the restriction that you are only able to achieve a completely correct data transfer when the BER is sufficiently small since no correction code is included.

This restriction does not add up to every type of ARQ method. There is also a Hybrid Automatic Repeat reQuest (HARQ) method which combines the two methods, a FEC and ARQ method, together and thus is capable of recovering errors without asking for retransmission. Though, this type of Error Control is much more complex than the ones stated above and a more detailed explanation of a HARQ method can be found in [5].

3.3.2 The one-way transmission system
Unlike in a two-way transmission system only one transmitter and one receiver are needed in order to have a one-way transmission system. In order to correct any errors a FEC technique has to be used, because the receiver can't send anything back to the transmitter.

The down side of only being able to receive is that it's not possible to fix any errors that can't be repaired with the used Error Correction method. If the connection is of a pore quality there isn't much that can be done to still receive proper data.

Nevertheless, one way transmission systems are widely used because they are easy to implement, have lower power consumption than when using transceivers and are cheap.

3.4 Conclusion and application

Our main goal is to determine whether it is possible to send a photo or image over a one-way transmission system without having 'noticeable' errors. This means the transmission is a success if the received picture is almost 100% the same as the transmitted one. More on this topic can be found in chapter 8.

This research focuses on one-way communication, because this communication allows one transmitter to transmit data to multiple receivers at once. Two-way communication requires acknowledgements and retransmissions. This is only useful if one transmitter is communicating with one receiver. However, we are looking for a technique to 'broadcast' an image to multiple E-papers, since advertisement companies have to spread many copies of the same poster.

Error Control in this research

The transmission kit designed for this thesis uses a Cyclic Redundancy Check and a repetition code, by transmitting the same packet multiple times. The CRC will guarantee with great certainty that only valid data is forwarded to the E-paper. If a transmitted packet contains errors, the CRC will recognize them and the receiver will discard the packet. Because of the repetition code, the same packet will be retransmitted multiple times.

Depending on the deterioration of the signal, there is a good chance a packet is never transmitted correctly. In this case, data is missing and the image will miss pixels. By adding addresses to the pixel, the receiver can recognize which pixels are missing. This way, the E-paper can easily set the right data to the right pixel. Another advantage of adding addresses to the pixels is the ability to rearrange them if they were received out of order. There is a chance that packets are received out of order. This is detectable and correctable by adding addresses.

The Error Control used in this research is extensively tested. The results of these tests are discussed in chapter 8.

4. The Transmission software

This chapter gives a detailed description of the implementation and the working of the transmission software. The goal of the transmission software is to read an image, resize it and send it over a USB port to the Arduino microcontroller, which should be connected to the computer.

The chapter is divided into seven paragraphs. In paragraph 1, the choice of the programming language will be explained. Paragraph 2 will explain how a user can install the software, followed by paragraph 3 about transmitting an image or a slideshow using the software's Global User Interface (GUI). Paragraph 4 will describe how the program is written by explaining its actions when a button is clicked in the GUI. In order to transmit the image, some other programs or functions are necessary. They will be discussed in paragraph 5. Paragraph 6 will deal with compiling the program and transforming it into a stand-alone application, so users can run the program without needing to buy any other software. To conclude this chapter, a short recapitulation will be given in paragraph 7.

4.1 The choice for writing the program in MATLAB

In the beginning of writing a program, the most important decision to make is the choice for a programming language. This choice will have great influence on the final result, since each programming language has its own advantages and disadvantages and enables different options. The choice for a programming language is based on the functionality of the language and the expertise of the programmer.

4.1.1 Demands on the functionality

There are three important demands on the functionality of the language. The first demand deals with the communication with the hardware. In order to transmit data over a wireless channel, a computer program has to be able to control an external device. A transmitter will be connected to one of the computer ports, so the program has to write data via this port to the transmitter. Multiple ports can be used for this purpose. A virtual COM port is used for this research. The choice for a virtual COM port and more information about this port is given in appendix C.

The second demand relates to the ability to read and edit an image and write it to the port. Preferably, the language should have predefined functions to read and edit images, because good understanding of image formats is beyond the scope of this thesis.

Finally, there need to be developing tools available for the programming language in order to create a GUI. End users will require a simple GUI to operate the program.

4.1.2 Expertise of the programmers

In order to write the program efficiently, the programmers should write the program in a language with which they are familiar. This limited the choice to a few programming languages. We have experience with Java, MATLAB, C++ and to a lesser degree with Basic.

4.1.3 Determining the best programming language.

Java, MATLAB and C++ meet all of the requirements stated above. Our final choice was a comparison between these three languages. We chose for MATLAB, because it seemed very convenient to write to and read from virtual COM ports and to read and edit images with MATLAB.

4.2 Installing the transmission software

The software can be installed from a CD. On this CD, a map and four files are present:

- MCRInstaller.exe
- gs871w32.exe
- PictureTransmitterV2.exe
- Help.doc

These files contain the software needed to run the program correctly. The map 'Driver' contains a driver that needs to be installed for correct communication with the transmitter. When the microcontroller is connected to the USB port of a computer for the first time, a dialog box will ask the user to install the driver. The user has to select the folder on the CD in this dialog box. When this is done, the same dialog box will open and this action needs to be taken again before the driver is properly installed.

When this is done, the user will have to execute the files on the CD in a specific order. First, the MCRInstaller.exe has to be executed, installing the MATLAB environment that is needed to run the entire program on the computer. If the user wants to be able to open PDF files, gs871w32.exe has to be executed next. This installs Ghostscript, a free interpreter of the PDF language. When these installations are completed, the actual transmission software should be copied to a folder on the computer. After this action, the software is ready to be opened by executing PictureTransmitter.exe.

4.3 Using the program with the Global User Interface

When the program is started, the user will have to log in with his personal username and password (see figure 4.1a). This is necessary to make sure that the owner of an E-paper can use the program to transmit an image only to his own E-paper. The program will recognize the user and prepare to send data to his E-papers. Next, the main window shown in fig. 4.1b opens. In this window are a few pushbuttons visible, some radio buttons, a large white box and some text fields.

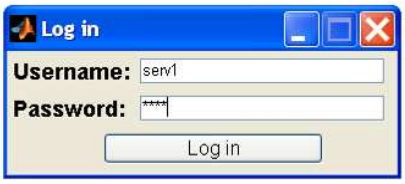


Fig.4.1a: Logging in

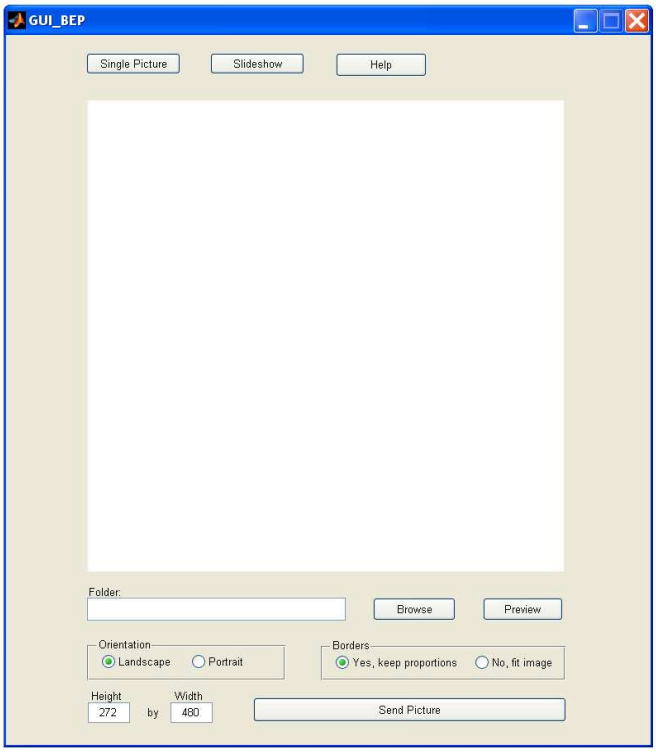


Fig. 4.1b: Main window

4.3.1 Sending a single picture

The program can transmit images as a single picture or as a slideshow. These different modes can be selected with the buttons on top. The default is the 'Single Picture' function, for sending a single image to an E-paper. The program allows the user to select a picture from their computer, show a preview, rotate it, add or delete borders and send it to an E-paper.

A picture is selected using the *Browse* button. Pushing this button opens a panel where the user can select an image in any image file format or a PDF. This window is shown in figure 4.2.

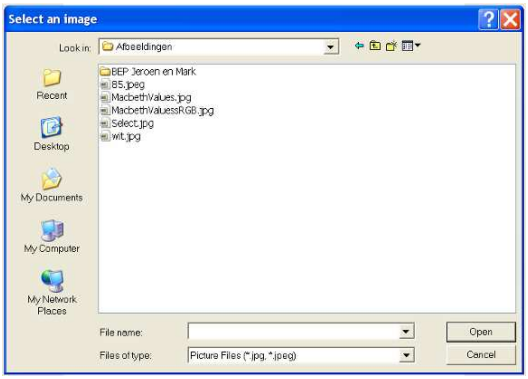


Fig. 4.2: Selection window

When a file is selected and opened using *Open*, the text field next to the *Browse* button will show the path of the file. Clicking *Preview* will show a preview of the image instead of the big white box. This is shown in figure 4.3. If the selected file was a PDF, the program will need to convert this file to an image. This will take some time and can't be aborted since the program 'Ghostscript' is used for this action (see paragraph 5). The user will be informed when the program is converting and when the program is done.

The program is going to show the image on an E-paper of 480 by 272 pixels. Pictures can be shown with the long side vertically (portrait) or with the long side horizontally (landscape). To give the correct preview, the user has to specify which rotation is intended. This can be done by selecting one of the two radio buttons in the *Orientation* box, *Landscape* or *Portrait*.

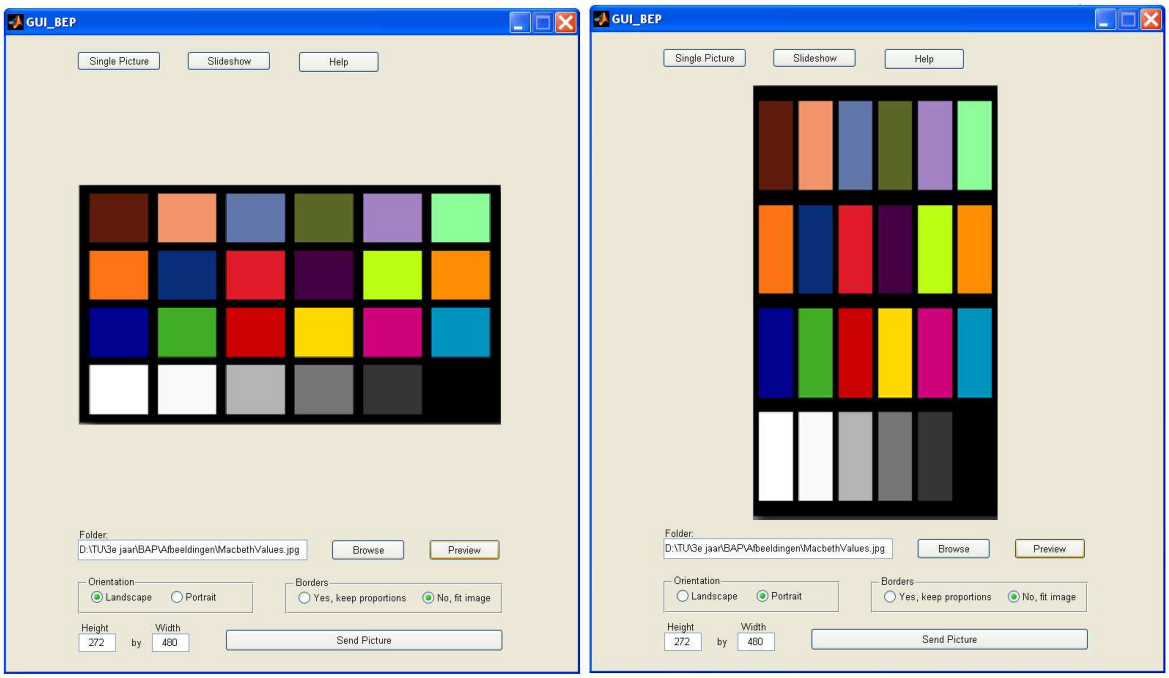


Fig. 4.3: A fitted image is shown in Landscape (left) and Portrait (right)

The size of an E-paper is predefined. The user can enter the size of his E-paper under ‘Height’ and ‘Width’, but since we can only test this program on a screen of 272 by 480 pixels, these values are locked in the program. The image will be resized to these proportions. This can be done by simply fitting the image into a window of 480 by 272 pixels, or by adding black borders to the picture and then resize it to the right size. In the first case,

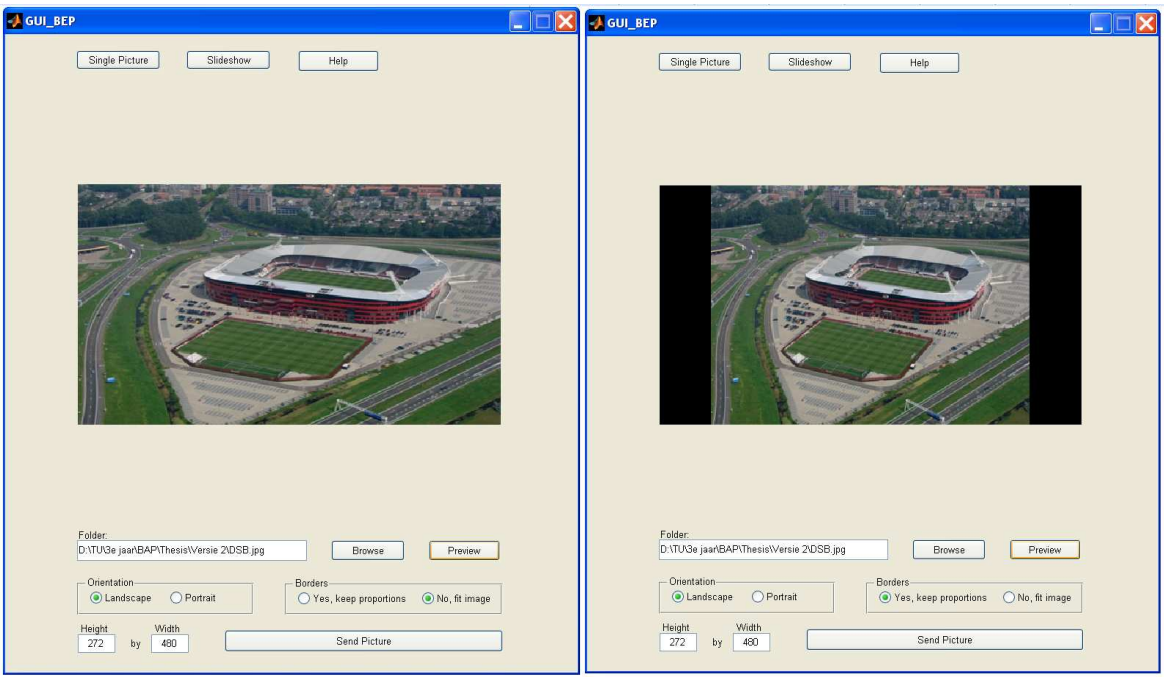


Fig. 4.4: A Landscape image is shown with borders (right) and fitted (left)

the picture will be altered in the way a television show is altered when it is watched on a wide screen TV: the image is stretched or contracted to the right proportions and then shrinked or enlarged to the right size. The original proportions will be lost. This will give a strange, distorted image if the proportions were very different from the proportions of the E-paper, but it will be a good option if the proportions agree almost. The second option will maintain the original proportions and fill the E-paper with an undistorted image. All the unused pixels are set to black. This is shown in figure 4.4.

Fig. 4.5: Dialog box

The *Send Picture* button actually sends the image to the E-paper. When this button is pressed, the program will first try to establish a connection with the transmitter. When the transmitter is found, the image will be prepared for the transmission. A dialog box will inform the user of this action, as seen in figure 4.5. Closing this dialog box will cancel the process. When the program is done, the transmission is started. A red wait bar will inform the user of the progression of the transmission (figure 4.6). If the user decides to abort the transmission, he can click on the *Cancel* button. The program will immediately stop transmitting and the new image will not be shown on the E-paper. Finally, a dialog box will inform the user if the transmission was successful.

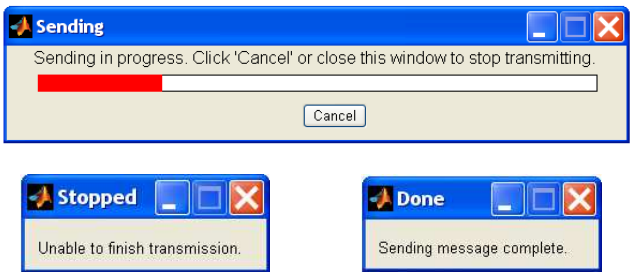


Fig. 4.6: A wait bar (top) indicates the progress made with the transmission. A dialog box indicates whether the transmission was aborted (bottom left) or completed (bottom

4.3.2 Sending a Slideshow

The window changes when the *Slideshow* button is clicked, as shown in figure 4.7. The preview area disappears and more *Browse* buttons and text fields appear. This window can be used to transmit a slideshow: a series of pictures that are subsequently displayed on the E-paper. Up to three pictures can be selected using the three Browse buttons as described in section 4.3.1. The text field under *Seconds between pictures* is used to tell the E-paper hardware how much time one picture should be showed before going on to the next picture. However, this function requires that the E-paper can store up to four pictures: the three pictures for the slideshow and a new picture when that is transmitted.

Our testing facility was made by other students and could only store one picture, so we altered our design. The slideshow is now showed as soon as the first picture is transmitted to the e-paper. The *Seconds between pictures* button isn't used at this moment, but that could be easily changed in future versions of this program.

Since all pictures of the slideshow are displayed on the same E-paper, the orientation is the same for all pictures. For simplicity, all pictures need to have the same resizing properties too. When the pictures are selected, you're ready to transmit. Clicking *Send Slideshow* will start the transmission.

The user receives the same information when sending a slideshow as when sending a single picture. The progress will be shown using a red wait bar and a dialog box will appear when the transmission completes or errors occur.

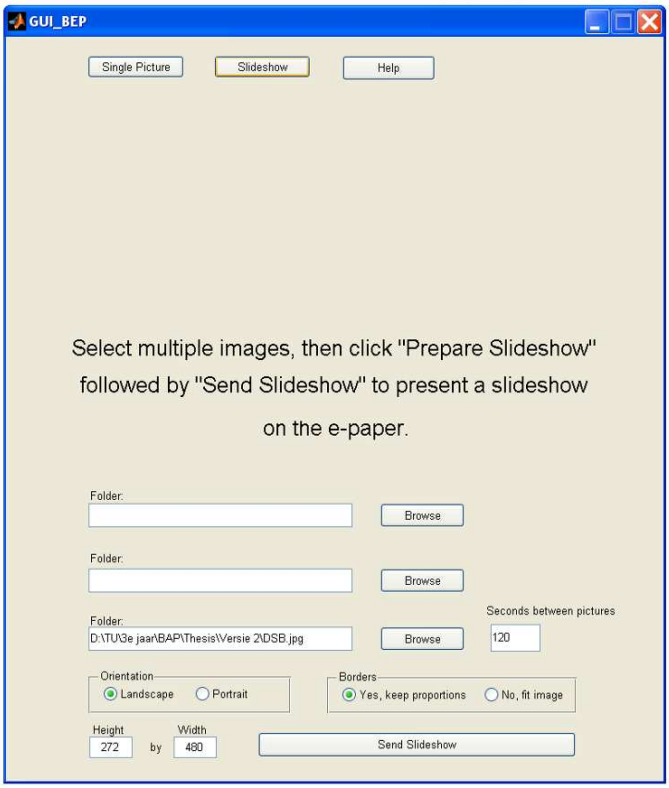


Fig. 4.7: Slideshow window

4.3.3 The Help-function

The *Help* button on top of the interface displays a simple help file when it is clicked. This gives the user a brief and simple explanation on how to use the program. For a more detailed explanation, the user is referred to the Readme.doc file.

4.4 Implementation of the program

The Global User Interface is started when the program is executed. This GUI controls the program and executes functions. To explain this, we will first give a small introduction into MATLAB. Readers who have experience with MATLAB may want to skip this part and continue with section 4.4.2. Some knowledge of programming and linear algebra is required for understanding the rest of this chapter. The actual code is available in appendix A.1, with a description of each function.

4.4.1 Brief introduction into MATLAB

MATLAB is a high-level language and interactive environment that enables users to perform computationally intensive tasks faster than with traditional programming languages [6]. MATLAB performs these tasks with functions: The user specifies input variables, a function and a place to store the output. MATLAB then calculates the output and executes other tasks given by the function. A function can be a simple task, such as adding two numbers. A more complex function might be calculating the inverse of a matrix. The input variable, or input argument, is the matrix to be inverted. The function is called `inv`. If the user stores a matrix in 'A' and wants to invert it and store the inverse matrix in 'B', then a valid MATLAB syntax would be `B = inv(A).`

A function can also be more complex and doesn't always need input arguments or return an output, such as the fwrite function. This function can be used for multiple purposes, depending on its input variables. We use this function in our program to send data over a serial COM port. If 's' represents an opened serial COM port, and we would like to write the number 1000 to this port, then this function would perform this task for us. A valid syntax would be `fwrite(s,1000)`. The function will not return an output, but instead it will perform the write operation: it will write the data (the number 1000) to the serial port and send it to an external device.

MATLAB users can make their own functions using m-files. An m-file is an ASCII text file with MATLAB code in it [7]. The user starts by giving the function a name and add input and output arguments if necessary. The rest of the m-file describes what task has to be performed when the function is called.

A function can be composed of other functions, just like a mathematical function. For example, calculating how much percent 20 is of 300 could be done by calling 2 functions. The first would divide 20 by 300, the second would multiply the result by 100. This could be encapsulated in one function called `percentage`:

```
function p = percentage(a,b)

fraction = a/b;
p = fraction * 100;
```

Now `p = percentage(20,300)` will return the result in 'p'. The semicolon at the end means we don't want MATLAB to print the result on the screen, but just store it in 'p'.
A Global User Interface is also a function in MATLAB. The function doesn't always need input or output variables, but it will open a window containing the GUI. Our GUI can be

opened by typing its name, GUI_BEP, in MATLAB's command window. The main window, shown in figure 4.1, will open.

A GUI has some properties which can be specified when it is created. An example of such a property is the position: the window can open anywhere on a screen, so the 'position' property specifies where it has to be opened. Another example of a property is its visibility: A GUI can be made invisible by setting the value of property 'visible' to 'Off'. Changing this value to 'On' makes the GUI visible and usable.

One of the outputs of a GUI is usually its 'handle'. A *handle* is a callable association to a MATLAB function [8]. In the case of a GUI, this means that a handle can be used to change properties of an already opened GUI, for example to change its 'visible' value. This could be done by the MATLAB code

```
set(<Handle>,'Visible','On');
```

4.4.2 Functions in the program

As discussed in the previous paragraph, a GUI is just another function in MATLAB. And just like any other function, it can call other functions and it can be called by other function. In our case, the GUI is called by the function `PictureTransmitter`.

PictureTransmitter

The function `PictureTransmitter` has no input or output variables, but opens the function Login. The user is asked to enter his username and password by this function. If the username and password are correct, `PictureTransmitter` opens GUI_BEP, containing the GUI. It immediately sets its 'Visible' property to 'Off', making the GUI disappear from the user's screen and preventing any interaction with it. PictureTransmitter then sets its position to the centre of the screen and makes the GUI visible, enabling the user to interact with it.

Login

The function Login will open a figure with two editable text fields where the username and password can be entered, two static text fields where the strings "username" and "password" are showed and a button that will check if the username and password are correct when it is clicked. This function shows all characters that are entered in the password box as an asterisk.

GUI_BEP

The function GUI_BEP may have a variable number of input arguments, setting some of its properties. When called, the function creates the window containing the GUI and all objects in it, such as the buttons, text fields and images shown in figure 4.1. All objects are created by creation functions, setting some of their properties. Visibility is one of these properties. In fact, all buttons shown in figure 4.1 and figure 4.7 are present at all times, but their creation functions determine whether they are visible when the program starts.

Every time a button is pushed, the program executes a function associated to that button, the so called ′callback′ function. The following sections will describe what happens when each button is clicked.

Single Picture

The button Single Picture sets all buttons, text fields and the white preview section from figure 4.1 visible and all other objects invisible. This way, only the objects useful for sending one single picture are visible and usable.

Slideshow
This button has the same functionality as the Single Picture button, but it only sets the objects visible that are needed for sending the slideshow.

Help
The *Help* button sets all buttons and text fields invisible, except for the buttons *Single Picture* and *Slideshow*. It also displays a text with some information on how to use the program.

Browse
Browse is the common word for searching a file on the computer. This is done with the function `uigetfile`, which opens the dialog box where the user can search for a picture to transmit. All image formats and the PDF format are supported, but the user can narrow the search down by selecting only one format to be shown. Which types are specified as input for the `uigetfile` function. The function returns the filename and its path, which is used to set the edit text next to the browse button. If no file is selected, the field is filled with the same path as before. If there was no path in the edit text, the field is filled with a reference to the browse button.

Preview
The *Preview* button checks which file is selected in the edit text and displays that file instead of the white box or any other image shown on that location. To do this, the handles of the radio buttons are read. These determine the orientation of the image and how the image is resized (with or without borders). If the file is a PDF, it is converted into an image. Further information on this topic can be found in paragraph 5.

If the user hasn't specified a file in the edit text, the *Preview* button takes no action. If a PDF file is previewed, the converted file will be stored so it won't have to be converted again before transmission.

Send Picture
Just as Preview, this button reads the edit text, converts a PDF to JPG and resizes the image. However, it doesn't show the image after this is done. Instead, the function `sendData` is called with the resized image and the baud rate as input arguments. This function takes the following actions:

1. *Search for the external transmission device using the function* `handshake`.
2. *Transforming the picture into 3 matrices containing the RGB values.* For example: element (i,j) from the matrix 'red' determines how much red is present in the pixel of the image on row *i* and column *j*. This is necessary, because the receiver is connected to a testing facility made by other students, that requires these values to show the image.
3. *Open and initialize the port found with* `handshake`. The address of the receiver is passed on to the transmitter. This address is connected to the username used to log in.
4. *Add an address byte to every pixel.* The receiver can use this byte to show a pixel in the correct place.

5. *Open a wait bar.*
6. *Write the data to the port and update the wait bar after every 1000 write operations.* If the *Cancel* button on the wait bar is clicked, close the wait bar and abort the write operation when the wait bar is updated.
7. *Delete the wait bar after the write operation is done or cancelled.*
8. *Close the port.*
9. *Inform the user of completion or failure of the write operation with a dialog box.*

During these actions, the user gets to see dialog boxes with information about the progress of the program. These dialog boxes are opened with the function `showinfowindow` [9], which sets the title of the dialog box and the content.
Two try/catch statements are used in `sendData`. These statements are used to make sure the program has predictable behaviour in case of errors and prevent undesirable behaviour. When an error occurs during a write operation, the try/catch statements make sure that the wait bar is deleted and the port is closed. Without these statements, the wait bar and port would remain open in case of an error until the operating system forces them to close.

Send Slideshow
This button has almost the same functionality as *Send Picture*. However, when multiple images are selected, these are transmitted after each other.

4.4.3 Output from the computer

As discussed in chapter four, the microcontroller is connected to the computer through a USB port. A driver that creates a virtual COM port needs to be installed on the computer. MATLAB can write data to this virtual COM port. The driver then sends this data to the USB port. The output will be further discussed in chapter seven.

4.5 Additional programs

The program uses Ghostscript [10] in order to convert a PDF file to a JPG file. Ghostscript is a free interpreter of the PDF language and can be called from MATLAB if it is installed on the computer of the end user. Our program uses the function `ghostscript` [11] to call the interpreter. This function needs only the ghostscript commands as input arguments and gives them to the interpreter if it is installed. The function `ghostscript` was written by Oliver Woodford and can be found in the export_fig package on MATLAB's File Exchange. We implemented this function into our program, because it searches Ghostscript on the user's computer in an efficient way, calls it using MATLAB's `system` function and lets it execute the commands given by the input arguments.

Our program uses the function `pdf2jpg` to convert a PDF to a JPG. This function calls `ghostscript` with the following input arguments:

```
ghostscript(['-sDEVICE=jpeg -dNOPAUSE -dBATCH -dSAFER -r600x600 -
dLastPage=1 "-sOutputFile=tempim' pdfName '.jpg" ' '"' path '"']);
```

`-sDEVICE=jpeg`	*Activate the driver that produces JPG files from PDF files.*
`-dNOPAUSE`	*Do not wait for user input, just execute the commands*

`-dBATCH`	*Exit the interpreter upon processing of the last file.*
`-dSAFER`	*Enables some security checks*
`-r600x600`	*Set output to 600dpi*
`-dLastPage=1`	*Convert only the first page*
`-sOutputFile= <fileName.jpg>`	*Name the output file filename.jpg*
`-Path`	*Store the output file in Path*

These commands were established with [12] and [13]. The input arguments are read by Ghostscript and will perform the conversion. The output is an image in JPG format, stored in the same directory as the original file. This file is deleted after it is used by the program.

The function `showinfowindow` was written by Panagiotis Moulos. His function displays a dialog box with a title and user-defined message. It is quite similar to MATLAB's message box, only it omits the *OK* button and it always opens in the centre of the screen.

The last program we used is the driver that creates a virtual COM port. We use this to write data to a USB port combined with MATLAB's ability to write data to serial ports. The driver was created by Future Technology Devices International Ltd. (FTDI) and can be downloaded for free at their site [14]. When the user inserts the microcontroller into the USB port, he is automatically prompted to install the driver. The user needs to give the location of the driver on its computer twice for the microcontroller to work properly.

4.6 Stand-alone application

The program MATLAB is needed to run m-files on a computer. The function `pictureTransmitter` is an m-file and can only be run by MATLAB. However, a stand-alone application can be created using the MATLAB Compiler and a C/C++ compiler, for example the Borland or Lcc-win32 compilers.

The MATLAB compiler uses the C/C++ compiler to create an executable file, which will compile the entire program and all used toolboxes. Eventually, multiple files are created. The most important file is PictureTransmitterV2.exe. This is the executable file, which starts the program.

A MATLAB stand-alone application actually isn't a stand-alone application. The program uses a lot of MATLAB's predefined functions. For this purpose, MATLAB created the MATLAB Compiler Runtime (MCR). This program can be deployed royalty-free and enables the execution of MATLAB files on computers without an installed version of MATLAB. The version of the MCR should be the same as the version of MATLAB used to create the application. For more information, see [15].

Running an executable usually opens a Command Prompt, the command-line interpreter of Windows. The Command Prompt tells the user to wait while it is extracting the CTF file and starts the GUI. When the GUI is running, any information that it would normally be displayed in MATLAB's command window is shown in the Command Prompt. However, the Command Prompt reminds users of the old and difficult MS-DOS operating system, which may confuse them. Our program suppresses the appearance of the Command Prompt, so only the GUI appears on the screen. This is done by adding the following command to the options file before compilation [16]:

```
set LINKFLAGS=%LINKFLAGS% -subsystem windows
```

The options file can be found and opened with these commands:

```
cd(prefdir)
edit compopts.bat
```

In a nutshell, this means that the user needs 2 files to run a MATLAB stand-alone application. He first needs to install the MCR by opening MCRinstaller.exe. When this is done, he can run the EXE file, PictureTransmitter.exe. This will open the GUI. Our users need to install Ghostscript as well if they would like to transmit a PDF file.

4.7 Summary

PictureTransmitter.exe will open a window where users can select images to transmit to their E-papers. It can send a single picture or a slideshow of up to three pictures with an adjustable pause between two pictures. The user is continuously provided with information about the progress of the program.

The program was written in MATLAB, because of MATLAB's image processing capabilities and write functions. The end user needs the MCR to execute PictureTransmitter.exe. For writing data to the serial port, a virtual COM port has to be created with the FTDI driver. In order to import PDF files, the user will need to install the program Ghostscript as well.

5. The RF modules

Radio frequency modules are used to transmit the image from the computer to the E-papers. This chapter gives a complete description on choosing and working with RF modules. The next chapter will discuss how the data from the computer will be send with these modules.

First of all, paragraph 1 will discuss which modules are used for this research. Paragraph 2 will give a brief introduction into the Serial Peripheral Interface, since good understanding of this communication interface is very important for the rest of this chapter. Paragraph 3 will explain how a microcontroller can communicate with the modules followed by a description of the packets that are transmitted in paragraph 4. In paragraph 5 will be discussed how the modules should be configured before they can be used in this research. Paragraph 6 will give a short recapitulation of this chapter, before the implementation of the configuration and communication with a microcontroller will be discussed in chapter 7.

5.1 Choosing the modules

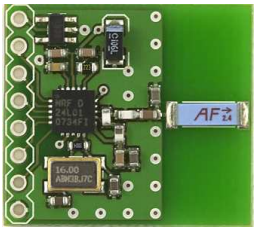


Fig. 5.1: A nRF24L01+ Transceiver. Source: Sparkfun Electronics [17]

First of all, we need to choose RF modules that can transmit the data from one microcontroller to another. Since our experience with RF modules is limited, we will need modules with very good documentation such as a clear datasheet, examples of application circuits and a manual on how to configure them.

The search for cheap but reliable RF modules that meet these demands led to the Nordic nRF24L01+ Transceiver modules [17]. A breakout board of these modules is available, which gives a module that only needs to be connected to a microcontroller. All other supporting circuitry such as a clock, voltage regulator and antenna is already present on the board.

The nRF24L01+ modules use a Serial Peripheral Interface (SPI) to communicate with the microcontroller. The next paragraph will give a brief introduction to SPI communication. Readers that are familiar with SPI may want to continue with paragraph 3.

5.2 Introduction to the Serial Peripheral Interface (SPI)

The following paragraph is based on [17] and [18].

The SPI-bus is a serial communications interface with four wires used by many microprocessors. These wires are:

- SCK – Serial Clock: a 50% duty cycle clock generated by the master.
- MOSI – Master Out Slave In: The line for writing data from the master to the slave.
- MISO – Master In Slave Out: The line for writing data from the slave to the master.
- CSn – Chip Select not: an optional control line, signalling the channel is active. Sometimes called Slave Select (SS).

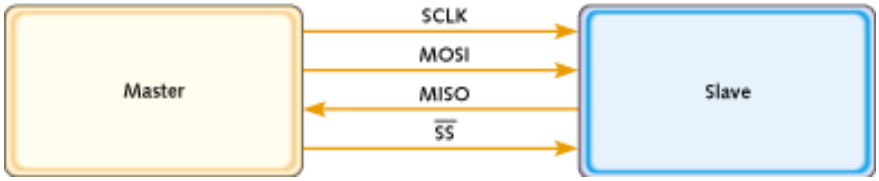


Fig. 5.2: SPI communication between one master and one slave.
Source: http://embedded.com/columns/beginerscorner/9900483?_requestid=245610 [19]

Two devices communicating with SPI operate the same clock. The master device creates the clock and the slave uses the clock to latch the data in or out. The transmitting device uses one edge of the clock to change the bit on the MOSI or MISO line. The receiver uses the other edge of the clock to read the bit on the same line. Communication is only possible when CSn is low.

Communication from master to slave can run parallel with communication from slave to master. This is called *full-duplex*. Both the master and the slave have an 8-bit shift register, connected to each other as depicted in figure 5.3. The slave writes a bit from the shift register to the masters' shift register using the MISO line. The master can shift this bit in and shift another one out using the MOSI line. This way, a distributed 16-bit register is formed.

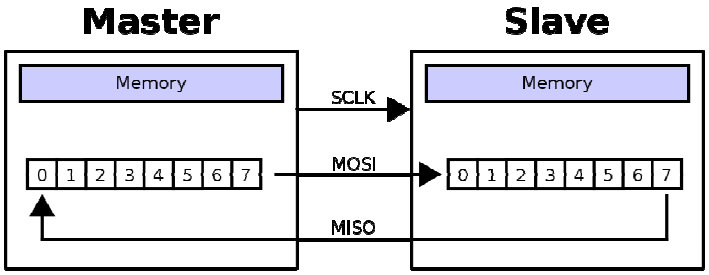


Fig. 5.3: SPI interface with shift registers.
Source: Wikipedia [20]

As mentioned before, the transmitter uses one edge of the clock to shift a bit out and the receiver uses the other edge of the clock to shift it in. A pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determines on which edge the data is shifted out and on which edge it is shifted in. Each of the two parameters has two possible states, which allows for four possible combinations. All combinations are incompatible, so a master/slave pair must use the same parameter values to communicate [21]. Figure 5.4 shows the effect of the clock polarity and phase on the communication.

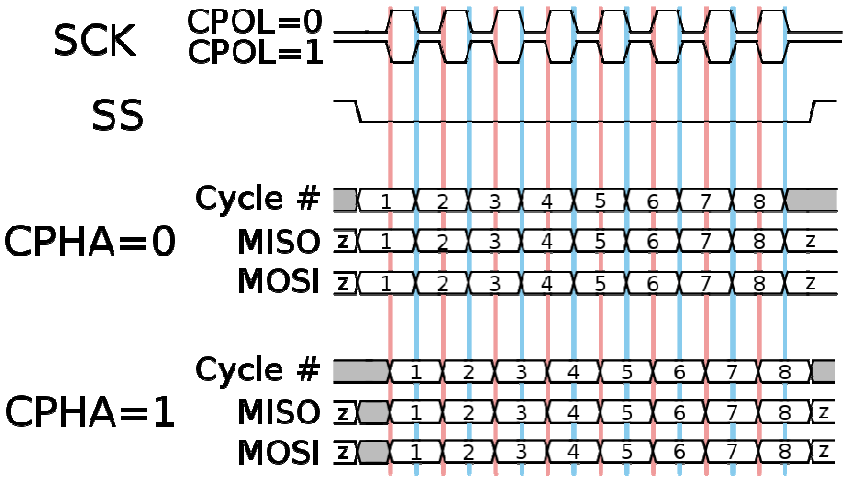


Fig. 5.4: Effect of clock polarity (CPOL) and clock phase (CPHA) on SPI communication. When CPHA = 0 the data is sampled on the red lines and changed on the blue lines. For CPHA = 1, the opposite is true. Source: Wikipedia [20]

5.3 Interfacing with the nRF24L01+ through SPI

The following is based on [21] and [22].

5.3.1 Pins on the nRF24L01+

As shown in figure 5.5, the transceiver breakout board has eight pins. Two pins, GND and VCC, are for powering the modules. A supply voltage of 3.3V to 7.0V is allowed. A microcontroller can communicate with the nRF24L01+ trough the SPI interface with the pins MISO, MOSI, SCK and CSN. The other two pins, CE and IRQ, are also used for communication with a microcontroller.

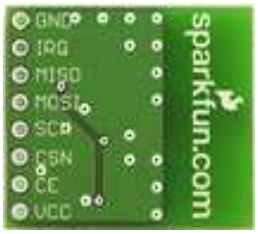


Fig. 5.5: A nRF24L01+ Transceiver. Source: Sparkfun Electronics [17]

CE is always an input for the transceiver. It has different functions, depending on what mode the transceiver is in. The module can either be in transmitter mode or in receiver mode. When it is a receiver, CE orders the module to monitor the air and receive packets when it is high. If the device is a transmitter, CE orders the device to send a packet.

IRQ is the Interrupt ReQuest pin. This pin can be used to signal to the microcontroller that:

- A packet is received
- A packet is transmitted correctly
- A packet can't be transmitted correctly

The microcontroller should consult the module to figure out which of these situations has occurred. As will be discussed in paragraph four, the module can be configured to use the IRQ pin only for one or two of these events. Using the IRQ pin is optional. The microcontroller can also use polling: keep checking if one of the events has occurred by 'asking' the module.

5.3.2 The instruction set

Communicating with the transceivers is done by sending instructions from the instruction set (or command set) to the transceiver. The complete instruction set can be found in the datasheet, see [22]. The most important instructions are showed in table 5.1. These instructions read data from or write data to some of the registers in the transceivers. The nRF24L01+ modules have 30 registers, each having an address of five bits. Some of these registers are used to specify the properties of the data transmission, such as the frequency and target address. Others are used to determine whether the device is a transmitter or a receiver. The STATUS register contains information about what's going on with the module at a given moment. More on these registers can be found in paragraph four.

To send a command, CSn should be made low to enable communication. Then the command byte has to be sent. Whenever a command is sent, the nRF24L01+ will always return the contents of the STATUS register. If the microcontroller performs a read command, it will have to send more bytes to the module than just the command itself. For every byte the microcontroller wants to receive from the module, it has to write a dummy byte to the module containing no information. This is necessary, because SCK is needed to send data from the slave to the master. When the microcontroller (master) sends a byte to the module (slave), it automatically uses SCK. This enables the module to send data back.

Command name	Command word (binary)	# Data bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read command and status registers. AAAAA = 5 bit Register Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write command and status registers. AAAAA = 5 bit Register Map Address Executable in power down or standby modes only.
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed.
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Table 5.1: A part of the instruction set for the nRF24L01+ modules. Source: Nordic Semiconductor [22]

The following instructions can be sent to the nRF24L01+:

- R_REGISTER: The microcontroller can call the content of a register with the instruction R_REGISTER. As can be seen in table 5.1, this is done by sending 000A AAAA to the module, where A AAAA is the address of the register. The module instantly returns the content of the STATUS register as usual. When the microcontroller sends more bytes, the content of the register is returned. If the register contains five bytes, the microcontroller will have to send five dummy bytes to the module to get all information from the module.
- W_REGISTER: Data can be written to a register using the instruction W_REGISTER. The command byte for this instruction is 001A AAAA, where A AAAA is again the address of the register. After the instruction, the data that needs to be stored in the register is sent.
- R_RX_PAYLOAD: When data is received, this will be stored in RX FIFO (first in, first out). The FIFO can contain data from three received *packets* (see paragraph 5.4). To read this data into the microcontroller, the instruction R_RX_PAYLOAD is used. To execute this command, CE will have to be used. Remember that in receiver mode, CE is high when the module is scanning for packets. To read received packets, CE has to be made low temporarily. After this is done, the operation can be executed by sending the command byte, followed by as many dummy bytes as the payload width is. The FIFO will return the first received data. If more data is present in the FIFO, this should be read as well. When the FIFO is empty, CE can be made high again to make the module monitor the air.
- W_TX_PAYLOAD: To transmit data, the TX FIFO is used. This FIFO contains the data that has to be transmitted to the receiver. This FIFO can be filled using the instruction W_TX_PAYLOAD. The number of bytes in the payload should match the payload length of the receiver. When there is data in the TX FIFO, the microcontroller needs to order the module to transmit it. This is done with the CE. In transmit mode, CE is normally kept low. By making it high, the transmitter sends all packets in the FIFO to the receiver.
- FLUSH_TX & FLUSH_RX: These commands may be used to clear all data in the FIFO's. This might be useful do after setup.
- NOP: this means *no operation*. Nothing will change, but the transceiver will return the SETUP register. This is a very useful instruction if polling is used.

The nRF24L01+ has more instructions. Those instructions are however not used in our research and knowledge of these instructions isn't useful for a better understanding of the modules. Therefore, these instructions won't be discussed here. Interested user can find them in the datasheet of the nRF24L01+, see [22].

5.4 Structure of a packet

The data in the TX FIFO is transmitted as a *packet*. This is a block of bytes that is send to the receiver and contains the data. The structure of such a packet is shown in figure 5.6. This figure shows that a packet consists of:

- A preamble
- An address
- A Packet Control Field
- The Payload
- CRC field

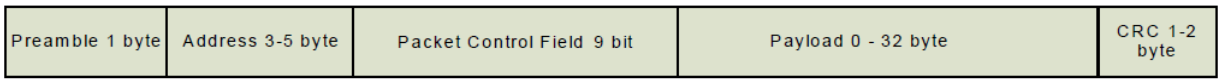


Fig. 5.6: Structure of a packet. Source: Nordic Semiconductor [29]

The *preamble* is a byte of alternating zeros and ones. It is used to let the receiver know that what it is hearing is the beginning of a packet and not just noise.

The transmitter will add an *address* to the data to send it to just one receiver, viz. the receiver that listens to that address. The address of our transmitter should therefore match the address of our receiver. Compare this to calling someone's name in a crowd, before giving that person a message. If you wouldn't call his name first, nobody would know who you are addressing. If you use only his first name, there is a strong possibility that there is another person with the same name who will also listen to the message and think he is addressed. When you use his first- and surname, this possibility decreases significantly. This could be compared to using a longer address (i.e. more bytes) for the receiver: the more bytes used, the smaller the probability of an error.

The *Packet Control Field* tells the receiver how many bytes the payload is long, how many times this packet has been sent before (in case of retransmission combined with auto acknowledgement) and if the auto acknowledgement function is enabled.

The *payload* is the actual data. This field may be one to 32 bytes long. The receiver needs to know the length of this field to understand the packet.

The last field is the *CRC* field. CRC means Cyclic Redundancy Check and is discussed in chapter three.

5.5 Configuring the nRF24L01+ modules

As was discussed before, the nRF24L01+ modules have 30 registers. All of them have an address of five bits. Some of these registers are used to specify the properties of the data transmission, for example the frequency, the data rate and the structure of a packet. Others can be used to determine whether the device is a transmitter or a receiver. This paragraph will discuss the registers that are used in this research. These registers need to be set before and during the transmission, to configure the nRF24L01+ modules and to determine how the modules will interact with each other. Only a brief overview of these registers will be given. Readers who would like more detailed information are referred to the tutorial from Ball [21] and the datasheet [22].

Before we start our overview of the registers, it should be mentioned that this research only requires a part of the functionality of the nRF24L01+ modules. Our goal is to determine if it is possible to transmit a picture from an RF transmitter to an RF receiver, using one-way communication. Because the nRF24L01+ modules are transceivers, it is possible to make them alternate between transmitter mode and receiver mode. This way, the receiver could send an acknowledgement back to the transmitter once the data is received correctly or it could ask the transmitter to send the data again in case of errors. Since we are using one-way communication, the receiver can't ask for a retransmission. Furthermore, it is possible to use

multiple transmitters to send data to one transmitter using multiple *pipes*. These pipes may be compared to multiple roads, all leading to the same location, but coming from a different location. With pipes, it is possible to receive messages from different transmitters. Even though this could all be done with these modules, that functionality is not wanted in this research and needs to be disabled. The registers where this is done will be discussed in section 5.5.2, but we will not give detailed information about these functions.

5.5.1 Registers for necessary functionality

Address 0x00 refers to the register CONFIG. This register must be given a value different from its reset value to make the device do anything. It contains the PWR_UP bit, which is initially zero. This bit must be set to make communication possible. This register also determines whether the device is a transmitter or a receiver by setting the PRIM_RX bit. As was mentioned before, there are three events that can cause an interrupt: receiving a packet, sending a packet correctly and failing to send a packet correctly. Not all of these events necessarily cause an interrupt on the IRQ pin. The CONFIG register determines which events do and which don't. Finally, CONFIG also sets the CRC scheme.

Address 0x03 refers to the register SETUP_AW. This is where the address width is set. The actual address is set in register 0x0A for the receiver and in 0x10 for the transmitter. Registers 0x0B to 0x0F set receiver addresses for other pipes, but this function isn't used in this research as mentioned before.

The length of the payload is set with register RX_PW_P0 with address 0x11. Off course, every pipe has its own register for setting the payload length. These have addresses 0x12 to 0x16, but aren't used in this research. All of these registers have reset value 0x00, meaning that no data can be transmitted since the payload length is zero. Some other value must be written to RX_PW_P0 to enable communication.

Register 0x05 tells the device which frequency is used. The lowest frequency is 2,400 GHz, but this register allows the user to use other frequencies up to 2,527 GHz.

RF_SETUP is the register that contains all necessary parameters to set up the RF section of the 24L01, such as the RF power level and the data rate. All reset values are fine for this research, so it is not necessary to write to this register. Only the data rate could be decreased from 2 Mbps to 1Mbps, to decrease the error rate at the cost of bandwidth.

Finally, we discuss one of the most important registers: STATUS. If an interrupt occurs, this register will tell the microcontroller what actually happened. If the bit RX_DR is set, data is received and RX FIFO can be read. TX_DS signals that a packet was transmitted successfully. MAX_RT is set when the maximum number of retries is reached. This is only used in combination with acknowledgements, which aren't used in this research. TX_FULL will be set when TX FIFO is full.

5.5.2 Registers for redundant functionality

The registers discussed in this section, need to be set once to disable functionality that is not used in this research. An example of such a register is EN_AA with address 0x01. This enables 'Auto Acknowledgment', which obviously isn't possible in one-way communication. All bits in this register need to be cleared.

Since there are no acknowledgements, retransmission is out of the question. This means the number of retries needs to be set to zero by clearing all bits in the register SETUP_RETR at address 0x04.

Only one pipe will be used to communicate from the transmitter to the receiver, all other pipes have to be disabled. This is done by writing the value 0x01 to the register EN_RXADDR with address 0x02.

The nRF24L01+ has more registers. Those registers are not used in our research and knowledge of these registers isn't useful for a better understanding of the modules. Therefore, these registers won't be discussed here. Interested user can find them in the datasheet of the nRF24L01+, see [22].

5.6 Summary

Nordic nRF24L01+ transceiver modules have been used for this research. These modules communicate with a microcontroller using SPI. An instruction set is used to enable communication between the microcontroller and the transceiver. Before the transceiver can be used, it will have to be configured with these instructions. Data can be sent to a transceiver in transmitter mode. The transceiver will transmit the data in a packet to a receiver, which is another transceiver in receiver mode. This module can store the data and send it to another microcontroller. The receiver would be able to send an acknowledgement back to the transmitter when the packet is delivered correct. However, this research focuses on one-way communication. This means that acknowledgements are out of the question, so this functionality has to be disabled.

6. Controlling the RF modules with an Arduino

A Serial Peripheral Interface (SPI) is needed to communicate with the transmitter. Our research used Arduino microcontrollers for this conversion. This chapter will describe how the Arduinos are programmed for the conversion and how they are used to simplify the implementation of Error Control.

An Arduino isn't the only device that allows USB to SPI conversion. Therefore, paragraph 1 will explain why we chose an Arduino for this function. Paragraph 2 discusses how the Arduino's are programmed. This paragraph will show that the Arduino also configures the RF modules and adds redundancy to the data.

6.1 The reason we used an Arduino Board

The transmitter requires a SPI interface to read and transmit the data. However, the computer will write the data to a USB port. Therefore, we need to build a bridge between the computer and the transmitter that converts the USB output to a SPI output.

For this purpose, there are two sorts of devices one could use. The first is a SPI shortcut. This device allows a user to control SPI pins over USB without involving any programming. See for an example [23]. The second device is a microcontroller with both USB and SPI connections. This device can perform many tasks, but it will have to be programmed before it will perform this simple conversion.

The SPI shortcut would obviously be sufficient for the conversion. However, a microcontroller costs about the same and has many other applications. For example, it would enable us to write a small test program to test the transmitter without using the computer. This could be very helpful in the designing process, because the transmitter and receiver could be tested without the transmission program. Because the program was under construction and could not yet test the transmitter, the decision was made to use a microcontroller for the conversion. On top of that, a microcontroller might simplify the implementation of error control.

There are many types of microcontrollers on the market. We would require a microcontroller that could test the RF modules and the data coming from the USB. A microcontroller that is programmable with a high-level programming language would be preferred, since this creates the possibility to write simple test programs in little time.

The most popular and best supported microcontrollers are the Microchip PIC and the Atmel AVR. The functionality for these is about the same, so we made a decision based on the complexity of writing a program. We decided to use an Arduino board, which is inexpensive and has a simple and clear programming environment. It only needs wires to be connected to a USB port and to the transmitter. The Arduino has a FT232R chip which changes a USB to a serial UART interface. As we will see in the next paragraph, this has great advantages for testing the signal coming from the computer. But most importantly: the USB and the SPI connections are present on the board, so the Arduino meets all of our demands.

6.2 The end result

This paragraph will explain how the Arduino's are programmed. The first Arduino is programmed for Serial to SPI conversion, which is needed for the communication between the computer and the transmitter. The second Arduino is programmed for SPI to Serial conversion, which is needed for the communication between the Receiver and an FPGA board.

The Computer-Transmitter communication program will be discussed first, where after the Receiver-FPGA board communication program is described.

A tutorial how to program an Arduino can be found in Appendix B.

6.2.1 Communication program 1: Computer to Transmitter

The main goal of the first program is: convert the incoming serial data into outgoing SPI data. Other important functions of this program are to configure the Transmitter (the nRF24L01+ module), to add redundancy and create a payload that consists of four bytes.

A global overview of the program is given in figure 6.1 and it is explained in three steps below. The actual program is stated in Appendix B.2.

Step 1: Initialisation

In order to create a SPI output the `transfer()` function is used which can be found in the `Spi` library. This function automatically generates a synchronised clock with the output data as described in chapter 5.2: *Introduction to the Serial Peripheral Interface (SPI)*.

Additional libraries are needed to initialise the transmitter: the `mirf` and `nRF24L01` library. Using the functions provided by these libraries, the number of bytes in a packet and the communication channel are set. The `config()` function configures the applied settings and powers up the nRF24L01+ module in the Receiver Mode.

Because the nRF24L01+ module is a transceiver, a couple of settings have to be changed in order to exclusively use it as a transmitter. First, the automatic acknowledgment function has to be disabled in order to set up a one-way communication network. Next to disabling the automatic acknowledgments the number of communication pipes has to be reduced to one, for the same reason. As an effect of these changes the retransmission function has to be disabled as well. Finally, the module has to be set into the Transmitter Mode.

Step 2: Setting the transmission address

When the initialisation is done, the program waits for the user to login in order to set the transmission address. If the login was successful, the upcoming serial bytes will be converted into SPI and transferred to the transmitter.

Step 3: Transmission

The payload in each transmitted packet consists of four bytes: an Address byte and the Red, Green and Blue values of a pixel. This means every packet contains the data to set one pixel. This is useful, because the packets might be received out of order. This could change the order of the Red, Green and Blue values, and the Address. Transmitting the data of one pixel in one packet makes sure that the data is received in the right order. The address byte is included in order to cope with the loss of packets due to wireless transmission and discarding of wrong packets by using the CRC. With the address, the receiver is still able to reconstruct the image by setting the pixels in the right order.

To create such a packet the four in sequence bytes are stored in an array and transmitted as one packet. Because a Repetition Code is used as well, the packet is send

multiple times instead of once. More information about the CRC and Repetition code can be found at chapter 3. Chapter 9 describes the increase in quality due to this Error Control.

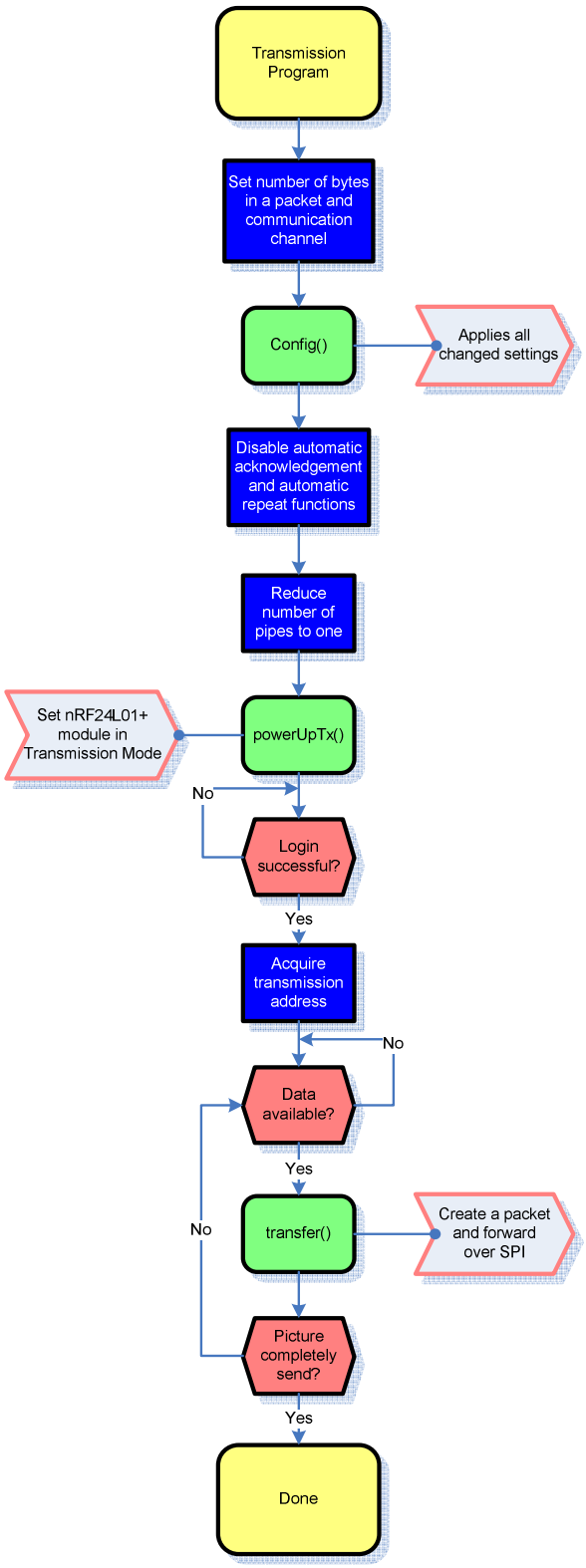


Figure 6.1: A state diagram of the program for the communication between Computer and Transmitter.

6.2.2 Communication program 2: Receiver to FPGA board

The main purpose of the second program is: convert the incoming SPI data into Serial output data. Another important task is selecting the right packets to forward.

The program is schematically explained in figure 6.2 where a more detailed explanation is given below. The actual program can be found in Appendix B.3.

Step 1: Initialisation

To be able to convert the incoming SPI data into Serial output data the `print()` function is used, which is a standard function within the Arduino program. This does not mean that no additional libraries are needed. The same libraries are used as in the first communication program: `mirf`, `nRF24L01` and `Spi`.

These libraries are needed in order to change the nRF24L01+ module into a Receiver. First the number of bytes in a packet and the communication channel need to be set to the same properties as in the first program. Second, the receive address is set equal to the transmitting address of the Transmitter. Where after the `config()` function is used in order to apply the changed settings.

By disabling the automatic acknowledgment and automatic repeat functions and reducing the number of pipes to one; the module is changed into a Receiver.

Step 2: Receiving

After initialisation the Receiver waits for incoming packets. Because of the Repetition Code a single packet is received multiple times. Storing the previous five addresses (which is the first byte of a packet) and comparing them to the received one will allow the program to forward a packet only once.

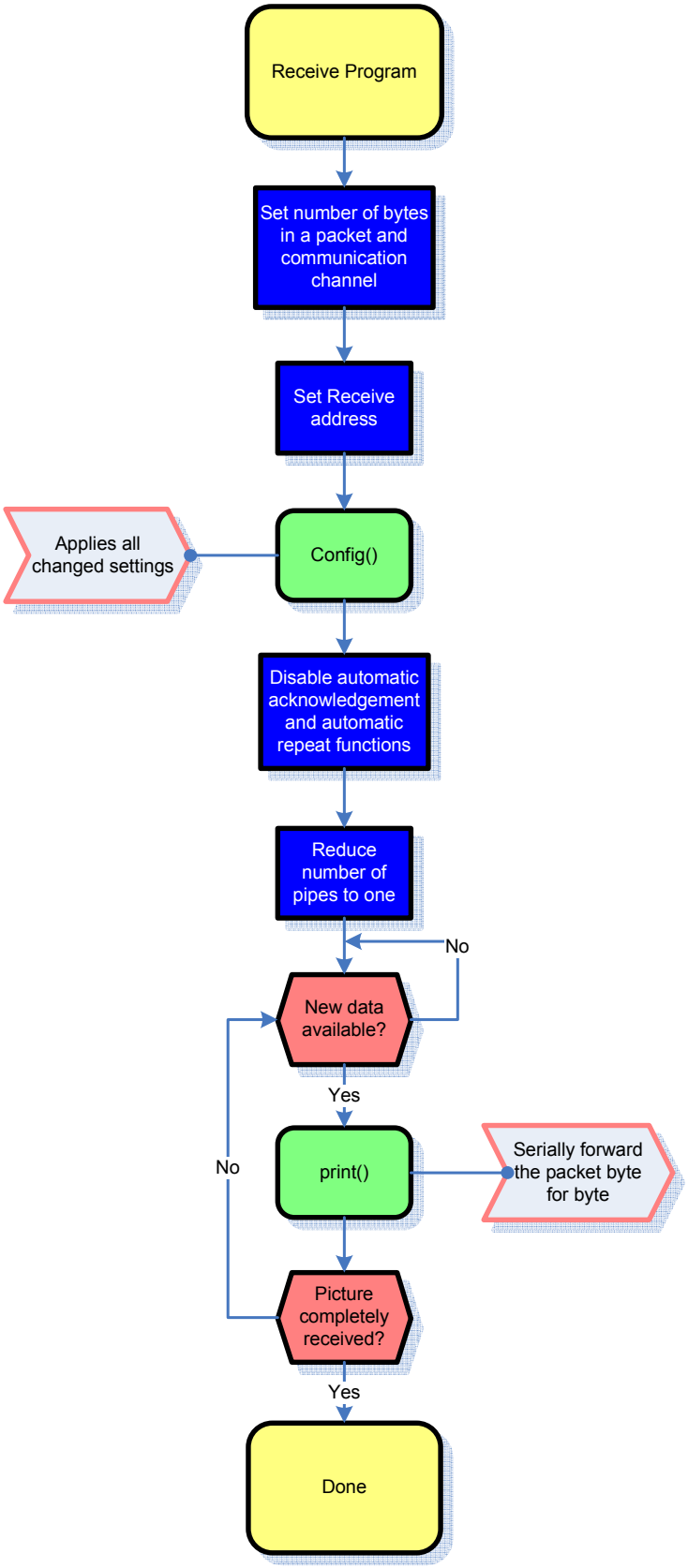


Figure 6.2: A state diagram of the communication program between the Receiver and FPGA board.

7. Testing the Transmission Software

The goal of our research is to determine whether it is possible to create a kit that uses a low-budget radio-frequency module to transmit an image without loss of quality to a receiver connected to an E-paper, which is controlled by a user-friendly and very simple computer program.

This chapter will describe how the user-friendliness of the software was tested before and after the final version of the software was created. The program described in chapter 4 was actually the second version, since the first version of the program still contained some bugs and wouldn't work properly on every system. In this chapter, version 1.0 will refer to an earlier version of the same computer program. Version 2.0 is the current version.

7.1 Testing Method

The first version of the program was working perfectly on the computers it was created on. However, it could still contain some bugs when it would be used on other computers. For this purpose, the first version of the program will be tested on functionality on other computers by independent persons. Any bugs will be fixed and other tips from the test group will be considered for the next version of the program.

To test the functionality in this first stage, some persons that weren't involved with the development of the software were given a CD with all needed software (including the drivers), a manual on how to install and use the program and a form to fill in all bugs they found and all tips they had for improving the program. One of the programmers will always be present to observe the testers and to help them proceed after they found a bug.

By fixing all bugs in the first testing stage, the operation of version 2.0 should be flawless. In this second stage, a group of 15 persons is asked to test the user-friendliness of the program. They will receive all needed software and hardware just like the employees of the advertisement companies. After they used the program to transmit an image, their opinion about the user-friendliness will be asked with a survey. This survey is shown in appendix E. The result of the survey will be discussed in paragraph 3.

If the testers in the second stage are unsatisfied with the software, a third version will be developed and tested in the same way the second version was developed and tested. This process will be repeated until a program is developed that satisfies the demands on user-friendliness and simplicity. If the testers suggest minor improvements but are satisfied with the rest of the program, those minor improvements will be implemented, but there is no need to test the software again.

7.2 Testing and improving Version 1.0

For testing version 1.0 of the program, a group of eight people was loaned a CD with all needed software, a USB cable and the Arduino microcontroller. The microcontroller was needed in order to test the `handshake` function of the transmission software: the software needs to find the Arduino on its own, without user intervention.
A few bugs were found during the first testing stage. Especially importing and editing PDF files needed some debugging. These bugs were found during the first testing stage:

1. Ghostscript couldn't be installed from the CD.

2. Not all PDF files could be opened by the program.
3. Converting PDF files of multiple pages could take very long and couldn't be stopped.
4. The transmission of the image to the Arduino stopped after some time.
5. The converted PDF file couldn't be saved.

The test group also had three tips to improve the user-friendliness in version 2.0:

6. Remember to which port the transmitter is connected, so it only has to be found once.
7. Open the GUI in the centre of the screen directly after start-up.

7.2.1 Ghostscript couldn't be installed from the CD

The installation of Ghostscript was done by opening the file gs871w64.exe. This resulted in the error shown in figure 7.1. The error was caused by using the wrong Ghostscript version. gs871w64.exe should only be used on computers with a 64-bits processor and a 64-bits version of windows. Many computers still use a 32-bits version of windows. This requires the file gs871w32.exe in order to install Ghostscript. Since gs871w32.exe can also be executed on 64-bits versions of Windows, this file will be supplied with the transmission software. The testers could download the right file from the Ghostscript website, so they were still able to test the rest of the program.

Fig. 7.1: Error during Ghostscript installation.

7.2.2 Not all PDF files could be opened by the program

Some users had no problem selecting and opening PDF files with the transmission software. However, other users got nothing on there screen when they wanted to preview a PDF file. In the command window, the following error from Ghostscript was found:

```
GPL Ghostscript 8.54 (2006-05-17)
Copyright (C) 2006 artofcode LLC, Benicia, CA.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
Error: /undefinedfilename in (pdf.pdf.jpg)
Operand stack:

Execution stack:
  %interp_exit .runexec2 --nostringval-- --nostringval-- --nostringval-- 2 %stopped_push --
nostringval-- --nostringval-- --nostringval-- false 1 %stopped_push
Dictionary stack:
  --dict:1124/1686(ro)(G)-- --dict:0/20(G)-- --dict:70/200(L)--
Current allocation mode is local
Last OS error: No such file or directory
GPL Ghostscript 8.54: Unrecoverable error, exit code 1
```

Ghostscript tells the user that the filename he selected doesn't exist, even though the selected file does exist. The problem appeared to be caused by passing the wrong filename to Ghostscript in the function `jpg2pdf`.

After investigating which files were converted by the program and which files weren't, the simple conclusion could be drawn that files with interspaces in the filename or the path weren't converted and files without interspaces were converted. Ghostscript uses a space between commands and apparently reads the part of the filename after the space as a new command. Only the part before the space is read as the filename. This problem was solved by adding quotes around the commands that are passed to Ghostscript. This way, everything between quotes is read as an entire command, solving our problem.

7.2.3 Converting PDF files of multiple pages could take very long and couldn't be stopped

A tester accidentally selected a PDF file containing a hundred pages. Ghostscript tried to create a picture that had all pages underneath each other. Since there is no use for displaying multiple PDF pages, a command was added to `jpg2pdf` that made sure only the first page of a PDF file will be converted to an image and only that page will be used by the rest of the program. This command is `-dLastPage=1` and is explained in chapter 4.

7.2.4 The transmission of the image to the Arduino stopped after some time

During the transmission of the data to the Arduino, the transmission was stopped and an error message was shown to the user. This bug is caused by the function `fwrite` in MATLAB 2007a, see [24]. The function `fwrite` will timeout sporadically. Since the transmission software uses this function thousands of times, chances that it will timeout and produce an error are very high. The problem was fixed by adding a patch to MATLAB, but the compiler can't compile this patch into the stand-alone application. The only way to solve this bug was to use MATLAB 2008a or higher to compile the program and create a stand-alone application. We used MATLAB 2008b to recompile our program, which fixed the bug. However, version 2008b of the MATLAB Compiler Runtime (MCR) is needed to run the program now.

7.2.5 The converted PDF file couldn't be saved

This problem was actually caused by the computer running the program. When a PDF file is converted to a JPG file, the result needs to be saved on the computer before MATLAB can access it. Since PictureTransmitter.exe was copied from the CD to a folder on the computer, this usually doesn't give any problems because the user has privileges to write files to this folder. However, when the user tries to convert a PDF file while PictureTransmitter.exe is found in a folder that is write-protected, Ghostscript cannot save the image it created and will give an error. This problem cannot be solved, since Ghostscript needs to save the image. We added a line to the user manual, saying the program needs to be copied to a folder that is not and will not be write-protected.

7.2.6 Finding the transmitter only once

When Send Picture or Send Slideshow is pressed, the software calls the function `handshake` to find the transmitter. By storing the output of `handshake` in the GUI, there is no need to call this function again when another picture is transmitted.

7.2.7 Open the GUI in the centre of the screen

When the GUI is opened, it first shows itself on the right of the screen before it is set to the centre. This gives the user a strange first impression of the program, so this was fixed in version 2.0.

7.3 Testing Version 2.0

For testing version 2.0 of the program, a group of fifteen people was asked to test the software at the development location or to test the software at home. In the last case, a tester was loaned a CD with all needed software, a USB cable and the Arduino microcontroller. The microcontroller was still needed in order to test the `handshake` function from the transmission software. However, the improvement from section 7.2.6 worked and the handshake function is only called once. The test group was asked to use the program to transmit an image, and to

fill out an enquiry about the user friendliness of the software. These enquiries can be found in appendix E.

7.3.1 Testing results

The most important results are as follows:

- The software is very user-friendly. The style and layout is very basic, but that fits the basic functionality of the program.
- No testers had problems with finding the Slide Show.
- One tester clicked on the button *Single Picture* without realizing he was already there, but soon figured out the purpose of this button after he pressed *Send Slideshow*.
- All errors with finding PDF files have been solved.
- Installing the MCR and starting up the software takes much time.
- Using the software does not require a manual.
- It would be preferable to log in by pressing 'enter' instead of clicking 'Log in' in the *Login* screen.
- The cursor in the password box is always set back to the beginning.
- The average rating the software got from the testers was an 8 on scale from 1 to 10.

7.3.2 Evaluation

The users thought the software was easy to use and was properly designed for this functionality. However, they did recommend some improvements. We investigated whether it was possible to improve the program based on their remarks. This led to the following conclusions:

- Making it possible to log in by pressing enter was a minor improvement, this was easily implemented in the software and there is no need to test this again.
- The cursor in the password box must always be set back to the beginning of the box after the asterisk is showed. This was not the case in MATLAB 2007a, the program we developed the software in. However, MATLAB 2008b must be used to create the standalone application and this version of MATLAB always sets the cursor back to the beginning of the box [25]. On our request, Mathworks is working on a fix for this problem. Until they fixed this bug, the cursor is always set back to the beginning of the field.
- MATLAB standalone applications need the MATLAB Compiler Runtime (MCR) to execute. There is no workaround for this and thus there is no way to speed up the installation or the start-up time of the program since this is a problem of the MCR.

7.4 Conclusion

Based on the testing results and the evaluation of these results, we can conclude that the software is indeed very user-friendly. Logging in is now also possible by pressing enter, but there is no need to start another testing procedure to test this new functionality. The installation takes some time, but needs to be done once and has no influence on further use of the program. Therefore, we conclude that the software has passed this test for user-friendliness.

8. Testing the quality of the transmission

One of the goals of this research is to determine whether it is possible to create a kit that uses one-way radio-frequency communication to transmit an image without loss of quality to a receiver connected to an E-paper. This chapter will describe how the quality of the transmission was tested and will discuss the results.

Paragraph 1 starts with a short recapitulation of the methods of error control, as described in chapter three. Paragraph 2 will discuss how the tests will be done. We will see that determining the quality of an image is tricky. Paragraph 3 will explain how the quality can be determined objectively and explain how can be tested whether the quality is sufficiently high. A hypothesis is formulated in paragraph 4 founded on probability calculus. Paragraph 5 will show the results of the testing procedure. Finally, paragraph 6 will check the hypothesis and draw conclusions.

8.1 Summary of error control

Wireless data transmission will inevitably lead to loss of data. Nevertheless, it is still possible to show an image with high quality. In this research, this was done by using CRC, adding redundancy and adding an address byte to the data. The CRC checks the received data for errors. If a package is transmitted with errors, it is discarded. Because of the redundancy, the same data will be transmitted multiple times. This is done to lower the probability a packet is not received.

If none of the packets with data for the same pixel is received, the pixel can't be set. The address byte lets the receiver know which pixel is missing. Because of this addressing, the receiver can still set the rest of the data to the right pixel. Since the address of the data is stored in one byte, 256 different addresses can be assigned. That way, 256 pixels in a row need to be missing for the pixels to be shifted.

8.2 Testing method

The transmission will be tested in three stages. The goal of these tests is to determine whether the developed kit and the used methods of error control can transmit an image to an E-paper without loss of quality. The first stage will investigate which *Bit Error Rate* (BER) can be achieved with the nRF24L01+ modules. The second stage will investigate at which BER it is still possible to transmit an image without errors using Error Control. The third and final stage will test at which BER it is still possible to transmit an image with sufficiently high quality.

8.2.1 Stage 1: Investigating the Bit Error Rate

The quality of a wireless transmission is normally tested by comparing the received data with the transmitted data. In digital systems, the measure of deterioration is usually taken to be the *Bit Error Rate* (BER). This is the number of bits that were received wrong, relative to the total number of transmitted bits. When large numbers of bits are transmitted, the BER will approach the probability of bit error P_e. The lower the BER, the better the transmission.

A good BER can be achieved by using an antenna with high gain, by increasing the power with which the data is transmitted and by decreasing the propagation loss during the transmission. The nRF24L01+ modules have an ANT-2.45-CHP antenna and transmit data with a maximum output power of 0 dBm. These are constants throughout the investigation. The propagation loss depends on the path between the transmitter and the receiver and is

different when the modules are used in different environments. The propagation loss will be high when the distance is big and there are a lot of obstacles between the transmitter and receiver. When the modules are in close proximity of each other, the loss will be very low. Noise created by other wireless transmissions will also increase the propagation loss. If the propagation loss is very high, the BER will increase as well.

To get an objective overall picture of the capabilities of the nRF24L01+ modules, the BER is measured at different locations in a gallery when there are no obstacles between the transmitter and receiver. However, because the measurements are indoors, reflection from the walls will play an important part. By increasing the distance between the modules and measuring the number of received packets at every location, the BER can be found at multiple distances.

The measurements will be done by transmitting three images of 30 by 30 pixels from a varying distance. After these three transmissions, the distance is increased and the images will be transmitted again. The BER can be calculated from the number of received packages as follows:

> Let P_e (BER) be the probability that a bit is transmitted wrong. The probability that a packet is transmitted correctly equals 1- P_e. There are 97 bits in one packet (see chapters 5 and 6). The probability that a packet is transmitted correctly is given by:
>
> $$P[\text{Packet Correct}] = (1 - P_e)^{97} \qquad (8\text{-}1)$$
>
> If every packet is transmitted once and in total 900 packets are transmitted for every image, the expected number of received packets is given by:
>
> $$E[\#\text{ Correct Packets}] = P[\text{Packet Correct}] \times \#\text{ Transmitted Packets} \qquad (8\text{-}2)$$
>
> If many packets are transmitted, the number of correctly received packets will approach the expected value described above. This way, P_e can be calculated with the following formula:
>
> $$P_e = 1 - \sqrt[97]{\frac{\#\text{ Correct Packets}}{\#\text{ Transmitted Packets}}} \qquad (8\text{-}3)$$

8.2.2 Stage 2: Full quality

The second stage will investigate at which BER it is still possible to transmit an image without errors using Error Control. This is the minimum BER needed to transmit an image if full quality is required by the receiver. To test this, 60 images of 30 by 30 pixels (900 packets) will be transmitted.

Obviously, this minimum BER will increase if more redundancy is added. In RF communication, a BER of 10^{-3} can be achieved indoors [26] if an antenna with a good gain and enough power is used. It would be preferable to achieve Full Quality transmission with this BER. Paragraph 3 will explain how much redundancy theoretically should be added to achieve this. The hypothesis that is formulated in that paragraph will be tested in this stage.

Since transmitting bits is a stochastic process, errors will always occur when large amounts of data are transmitted. Therefore, we will define ‘error free’ transmission as a transmission where 95% of the transmitted images will be without errors. The BER will vary with the distance between the receiver and transmitter. To find the BER where an image can be transmitted without errors, the measurements need to be done at different locations. The

wanted BER is found at the largest distance where error free transmission is possible. By transmitting another 60 images without error control, the BER can be determined accurately at this location.

8.2.3 Stage 3: High quality

The third and final stage will test at which BER (and thus at which location) it is still possible to transmit an image with sufficiently high quality. To do this, a scientific definition of "high quality" is necessary, which allows us to determine objectively when an image is transmitted with sufficient quality. Furthermore, a test program is needed to test the received image for this property. Both the definition of "high quality" and the development of the testing program will be discussed in paragraph 3.

Measuring at which BER the quality is sufficiently high is done in the same way as full quality is measured, but with the use of the test program. 60 images of 30 by 30 pixels will be transmitted. The received images will be compared to the transmitted images using the test program. The program will determine whether or not the quality is high enough. The final result will be the BER where 95% of the images passed this test.

8.3 Testing the quality

8.3.1 Missing pixels

The address bytes allow a receiver to set received data to the right pixel. When data for a pixel is missing, this pixel can be skipped so the other pixels can be set with the right data. However, a value *has* to be assigned to the missing pixel when the image is showed. To achieve maximum quality, the missing pixel is set to the value of an adjacent pixel. This increases the quality, because adjacent pixels are often alike. This correcting protocol is showed in figure 8.1.

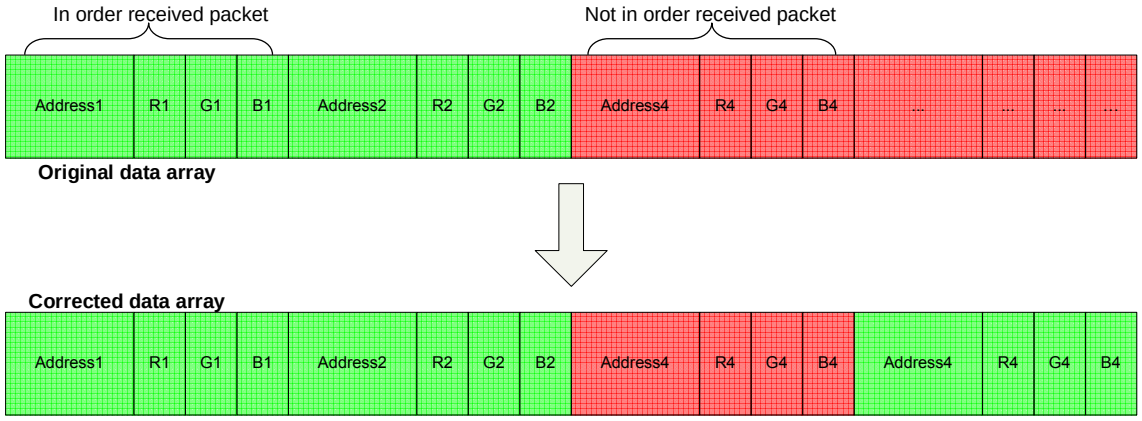


Figure 8.1: In the original data array the first two packets are received in order but packet number three is missing which will dislocate all the other packets and ruins the image. In the second data array the addresses are rearranged in the correct order filling up the not received packets with the packet next to it.

8.3.2 'High quality'

To test if the quality of the image is high enough, a testing program is necessary that also takes into account the visual aspects and that evaluates the quality of the picture objectively. Because the correcting protocol might set missing pixels to a value near its actual value, a simple *Packet Error Rate* (PER) check wouldn't suffice to check the image for high quality. To determine whether the quality is high enough, two aspects have to be tested:

- Are there enough packets received?
- Were the missing packets corrected properly?

The first property is tested by calculating the PER: how many pixels could be set correctly with the received data? The second property is more difficult to test. When are the missing packets corrected properly?

The received image will be presented in eight bit RGB colours. This means every pixel can be set with three bytes of data, where every byte represents the intensity of respectively red, green and blue. Changes in the lower four bits of one byte will not change the image drastically. This is shown in figure 8.2. The left half of this figure is filled with RGB values (255,50,50). The right side is filled with (240,50,50). It is hardly noticeable if a missing pixel is assigned the colour of the right half, when it should have the colour of the left half.

Fig. 8.2: Changes in the lower four bits are hard to see.

Therefore, we define properly corrected pixels as pixels where the upper four bytes are correct. An image with many missing pixels may still be of high quality if all the errors in the image can be found in the lower four bits of a byte. On the other hand, a program with very few errors can still be of low quality if all errors can be found in the upper four bits. Now, a proper definition of 'high quality' can be given.

The definition of 'high quality' used from now on is the following: less than 1% of the upper four bits in *every* byte of the image may be wrong compared to the original image and more than 95% of the packets must be received. Images that meet these requirements can hardly be distinguished from the original images. Obviously, it requires some difficult calculations to determine whether an image meets these requirements. The individual bits of RGB values have to be compared. This requires a testing program, which will be discussed in the next section.

8.3.3 The testing program

The operation of the testing program is showed by a state diagram in figure 3. This program was developed in MATLAB by means of three functions: `received.m`, `errorCorrection.m` and `getRGB.m`. This section will discuss the operation of the program and the three functions, but not the actual implementation of these functions. The code for this program can be found in appendix A.2.

The main function that is used is `received.m`. This function needs the COM port on which the Arduino is linked, the specific Baud rate, the RGB-values of the transmitted images for comparison and a Show bit which can be set when the received images need to be shown. With the first three variables the received data, the percentages and numbers of successfully received bits and the errors that occurred in the lower and higher four bits can be determined.

The COM port and the Baud rate are used to communicate with the Arduino and therefore used to receive any data. When the number of received bytes is lower than expected a second function is used. The `errorCorrection.m` function. This function uses the address, which is send with every pixel, in order to rearrange the received packets. An example of this process is given in figure 8.1. This way the received pixels will be in the right place on the screen and the missing pixels will be replaced with the same pixel that is closest to it. After rearranging the pixels the addresses are removed. The `errorCorrection.m` function only needs the incoming data as input and will return an array without the addresses as output.

After the correction, the array is compared to the original RGB-values which are located in the third input variable. When no correction was needed, the same

`errorCorrection.m` function will only strip the received data from the addresses where after the comparison is done.

During the comparison the number of well received bits per byte are counted and stored in an array. Also the number of errors in the lower four bits is monitored per byte. Having these two arrays, simple calculations can be done to get the desired percentages and numbers of successfully received bits and the errors in the lower/higher four bits.

In the end of the `receive.m` function the `getRGB.m` function is called if the Show bit is set. This function rearranges the received data array into RGB-values to recreate the received image. The function needs the number of columns and the received data without the addresses as its inputs.

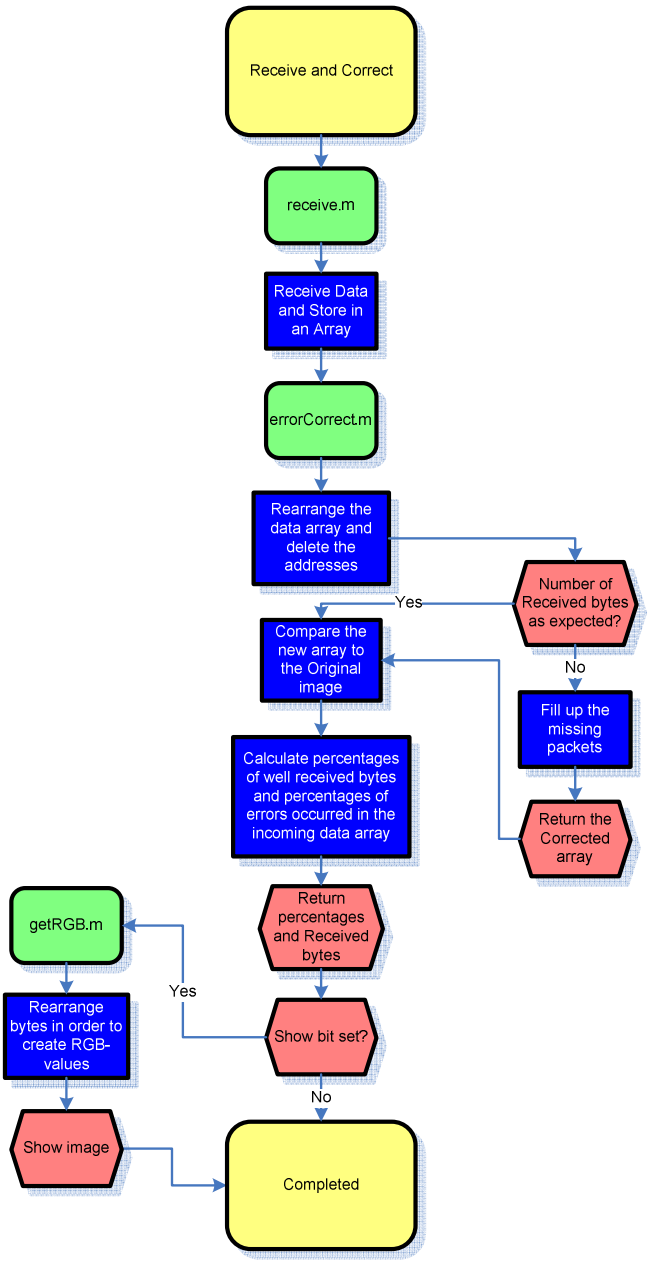


Figure 8.3: A State Diagram of the whole process of receiving an image.

8.4 Hypothesis

8.4.1 Stage 1: Investigating the Bit Error Rate

The nRF24L01+ modules use very little power and a pretty simple antenna to transmit the data. According to the vendors' website, the modules can reliably transmit data over 100 meters with a Baud rate of 250.000 baud using auto acknowledgment. However, we are using one-way communication, test the modules indoors and use a higher Baud rate. It is therefore expected that the range will be substantially lower than 100 meters. On top of that, it is expected that the BER will deteriorate if the distance between the transmitter and receiver is increased.

8.4.2 Stage 2: Full quality

It would be preferable to show a full quality picture when the Bit Error Rate is equal to or larger than 10^{-3}. This can be achieved by adding more redundancy to the data, but this automatically means the transmission will take longer. The redundancy that should be added to the data to achieve Full Quality transmission with BER $\geq 10^{-3}$ can be calculated as follows:

Let P_e (BER) be again the probability that a bit is transmitted incorrect. The probability that a packet is transmitted correctly is given by:

$$P[\text{Packet Correct}] = (1 - P_e)^{97} \qquad (8\text{-}4)$$

The probability that a packet is not transmitted correctly is

$$P[\text{Packet Wrong}] = 1 - (1 - P_e)^{97} \qquad (8\text{-}5)$$

If redundancy is added, the same packet is transmitted n times. The data for one pixel isn't received if all n packets contain errors. The probability that this happens and therefore a pixel is missing is given by

$$P[\text{Pixel Missing}] = [1 - (1 - P_e)^{97}]^n \qquad (8\text{-}6)$$

and

$$P[\text{Pixel Data Received}] = 1 - P[\text{Pixel Missing}] \qquad (8\text{-}7)$$

Full Quality is achieved when 95% of the images is transmitted without errors. Each image is composed of 900 packets. This means:

$$P[\text{Pixel Data Received}]^{900} \geq 0{,}95 \qquad (8\text{-}8)$$

Rewriting the above formula's and solving for n yields:

$$n = \frac{Log\left(1 - \sqrt[900]{0{,}95}\right)}{Log([1 - (1 - P_e)^{97}])} \qquad (8\text{-}9)$$

For $P_e = 10^{-3}$ this yields $n = 4.1$, meaning each packet has to be sent 5 times to transmit an error free image with this BER.

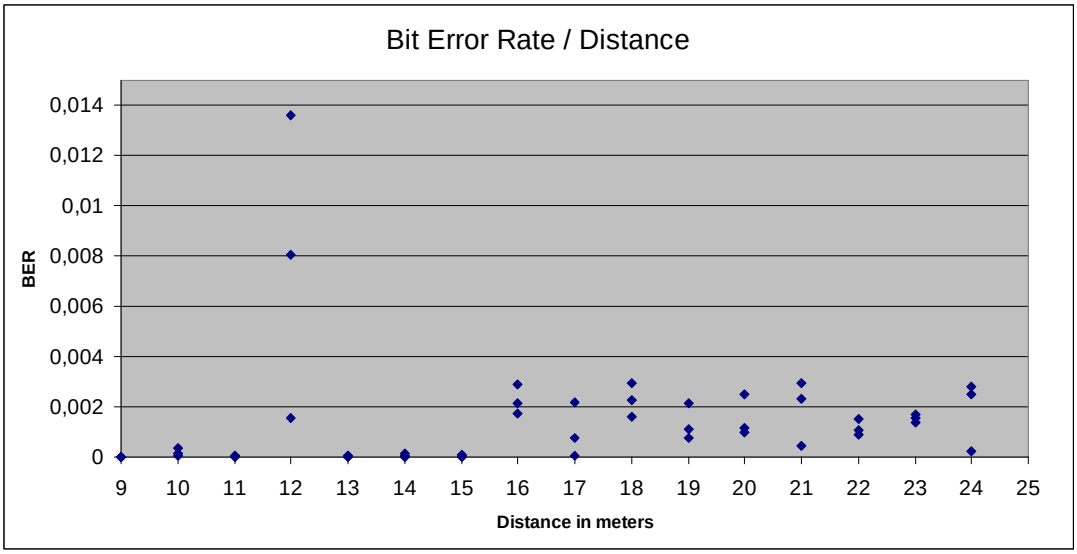
8.4.3 Stage 3: High quality

For most applications it is not necessary to have Full Quality. A higher BER would be allowed if High Quality is good enough. It is to be expected that the same redundancy level as in stage 2 would allow a higher BER in stage 3.

8.5 Test Results

8.5.1 Stage 1: Investigating the Bit Error Rate

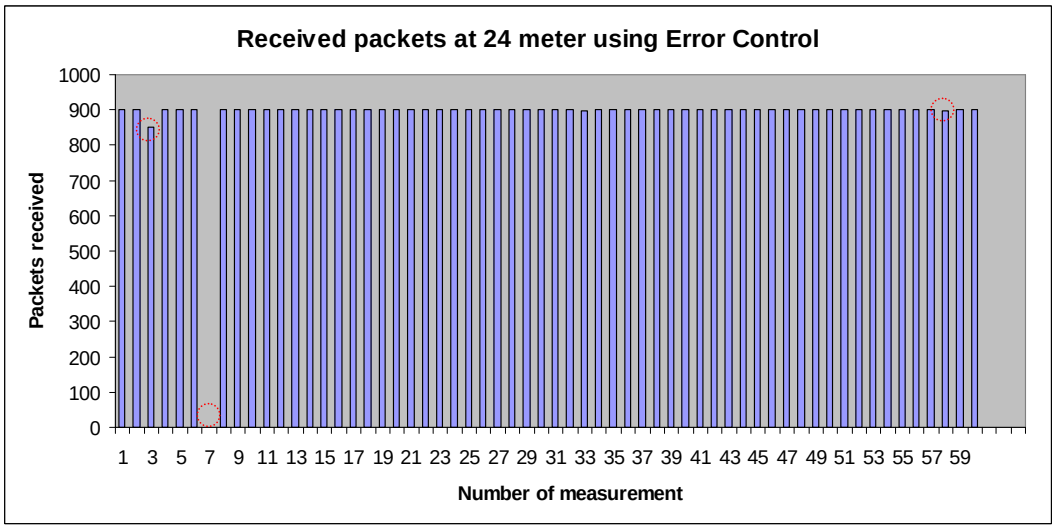
The results from stage 1 are showed in Graph 8.1. The number of received packets was measured as discussed in paragraph 2. From this received number of packets, the Bit Error Rates have been calculated and are shown as the blue dots in the graph. Except for the two measurements at a distance of 12 meters, the BER was never higher than 0,004 when the distance between the transmitter and receiver was no more than 25 meter.



Graph 8.1: The Bit Error rate measured with a varying distance between the transmitter and the receiver.

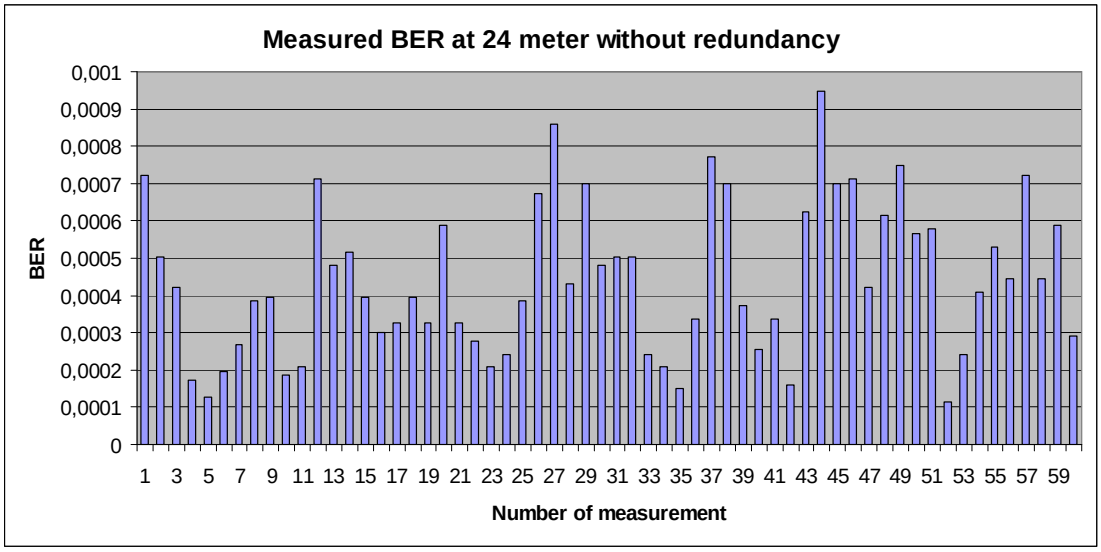
8.5.2 Stage 2: Full Quality

In this stage was investigated at which BER it is still possible to transmit an image without errors using Error Control. The hypothesis was that sending each packet five times would lead to Full Quality reception at a BER of 10^{-3}. 24 meters was the largest distance where Full Quality was found with this redundancy. As shown in graph 8.2, only three out of 60 measurements led to a reception of less than 900 packets. This is within the 95% interval.



Graph 8.2: Number of received packets.

In order to calculate the Bit Error Rate at this point, 60 images of 900 packets were transmitted. From the received number of packets, the BER was calculated using the method from stage 1. The result of this measurement is shown in graph 8.3. The average BER found at this distance was 4,4 x 10^{-4}. This is a better BER than 10^{-3}, which means that the hypothesis needs to be rejected: more redundancy is needed to achieve Full Quality when the BER is 10^{-3}.

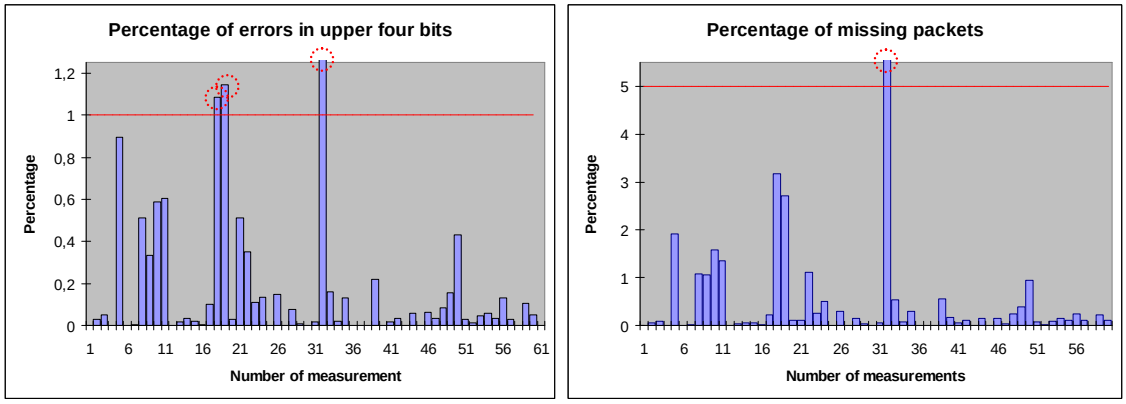


Graph 8.3: The Bit Error rate measured at 24 meters without using redundancy.

8.5.3 Stage 3: High Quality

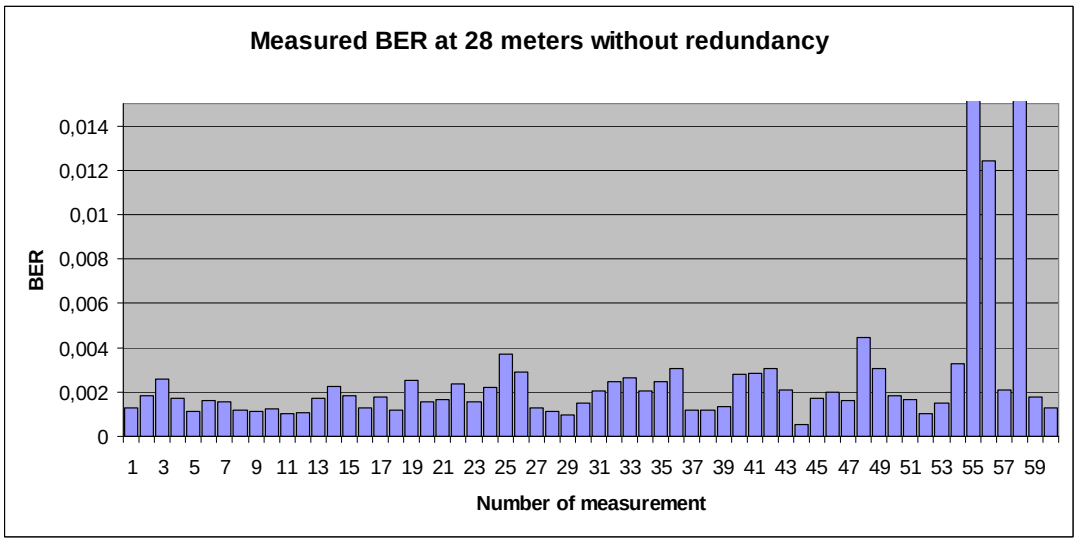
The third stage tested with which BER it is still possible to transmit an image with sufficiently high quality. The definition of 'High Quality' was: less than 1% of the upper four bits in *every* byte of the image may be wrong (a) compared to the original image and more than 95% of the packets must be received (b). 28 meters was the largest distance where High Quality was found. Graph 8.4a shows the result of part a. Three measurements did not pass this test. Graph

8.4b shows the results of test b. Only one measurement did not pass this test, but this was one of the measurements that did not pass test a. These results are within the 95% interval.



Graph 8.4a (left) shows the percentage of errors in the upper four bits at a distance of 28 meters. Graph 8.4b (right) shows the percentage of missing packets. Both measurements were performed with Redundancy. Only three measurements did not pass the test for High Quality.

In order to calculate the Bit Error Rate at this point, another 60 images of 900 packets were transmitted. From the received number of packets, the BER was calculated using the method from stage 1. The result of this measurement is shown in graph 8.5. The average BER found at this distance was 3,2 x 10^{-3}. This is worse than 10^{-3}, which means that unlike Full Quality, High Quality is achievable when the BER is 10^{-3}.



Graph 8.5: The Bit Error rate measured at 28 meters without using redundancy.

8.6 Conclusion

The goal of these tests was to determine whether it is possible to transmit an image without loss of quality to a receiver using the developed kit and the methods of error control. In stage 1, an overall picture of the nRF24L01+ capabilities was obtained. The modules have a BER less than 0,004 when the distance between the transmitter and receiver was no more than 25 meter. In stage 2, the hypothesis that Full Quality could be achieved with a BER of 10^{-3} by transmitting every packet 5 times had to be rejected: a BER of $4{,}4 \times 10^{-4}$ is necessary to obtain Full Quality. Stage 3 proved that High Quality can be achieved with a BER of 10^{-3}, confirming the hypothesis that a lower BER is needed for High Quality with respect to Full Quality.

Now what does this mean for the main problem of this thesis? The nRF24L01+ modules are clearly only capable of High Quality transmission when the transmitter and receiver are in very close proximity of each other. Without obstacles, a distance of 28 meters was the maximum. If these modules were used in a mall, the obstacles between the transmitter and receiver and the noise due to other wireless transmissions would probably even decrease this maximum distance. However, the tests also proved that this Error Control is capable of transmitting an image with High Quality when a BER of 10^{-3} is achieved. By using an antenna with more gain and by using more power for the transmission, a BER of 10^{-3} is certainly achievable with RF communication [26]. So with better hardware and more power, it should be possible to transmit an image without loss of quality to a receiver connected to an E-paper using one-way radio-frequency communication.

9. Testing the Modules in combination with a PSP Screen

The transmission characteristics were calculated in the previous chapter. Another test was done in combination with an FPGA board which is connected to a PSP screen. With the help of Joost Meerwijk and Willem Zwetsloot [see 27] we were able to simulate the communication with a Super E-paper.

In the first paragraph the setup is explained where after the test results will be evaluated. The second paragraph will be about the changes that have to be made in order to communicate with the actual Super E-paper. In the end a conclusion will be given whether it is possible to implement a receiver on a Super E-paper.

9.1 The Setup and Test Results

In order to test whether the communication between the Transmitter and Receiver which is attached to a FPGA board still works, the following setup is created:

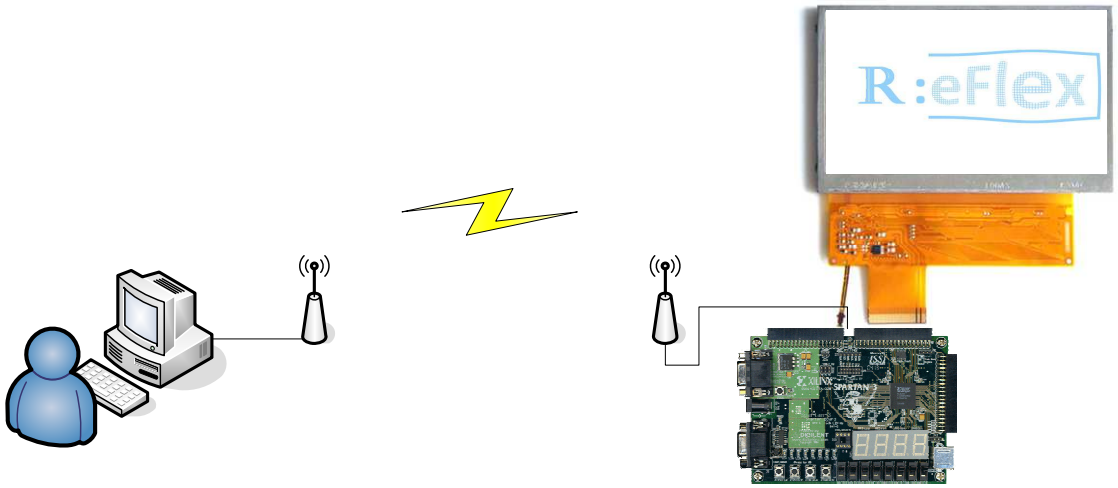


Figure 9.1: The global setup we used in order to test the communication with an FPGA board, which is connected to a PSP screen, and the Transmitter. The combination of the FPGA board [28] and PSP screen [29] simulate the actual Super E-paper.

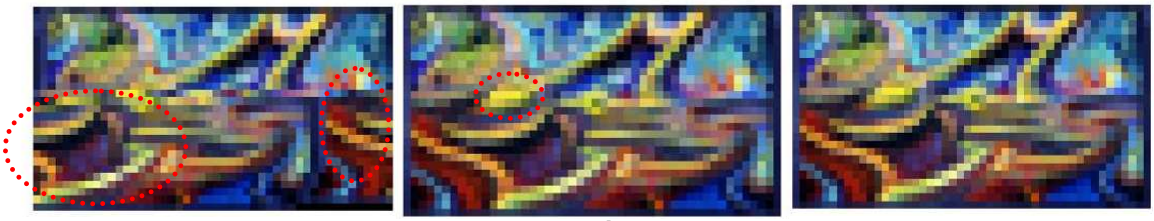
9.1.1 Applied changes with reference to the first test program

The same programs created in the previous chapter were used during this test with the exception of one. The developed "Super E-paper" system does not yet support the rearranging technique which uses the addresses in order to reallocate the pixels. Though this technique can not be used, all other redundancy methods still can. Therefore, only the addresses needed to be removed from transmission.

The order in which the data was send also needed to change, because of the manner in which the data was stored by the FPGA. Instead of Red, Green and Blue the order needed to change into Green, Blue and Red.

9.1.2 The Test Results

As expected the Bit Error Rate did not change with respect to the results given in the previous chapter. As a consequence of the absence of the rearranging function the received picture did shift whenever a pixel was missing. See for example figure 10.2.



a *b* *c* *c*

Figure 9.2: A picture without the rearranging function (a) and the same picture which does use the rearranging function designed in the previous chapter (b). The original picture is displayed in c.

Despite no rearranging function is used the test did prove a picture can be send to a “Super E-paper”. If all packets are received the picture was shown perfectly. Therefore, the test was still a success.

9.2 How to implement on the real Super E-paper

The test was successful but still has to be decided whether it is possible to implement the receiver onto the real Super E-paper. This decision will be based on whether the used receiver can be implemented on the Super E-paper.

First the circuit of the Receiver will be explained where after the possibility for implementing it on the Super E-paper will be evaluated.

9.2.1 The circuit of the used Receiver

In figure 9.2 the total circuit of the Receiver is shown where in figure 9.2 a block diagram is given only of the nRF24L01+ chip.

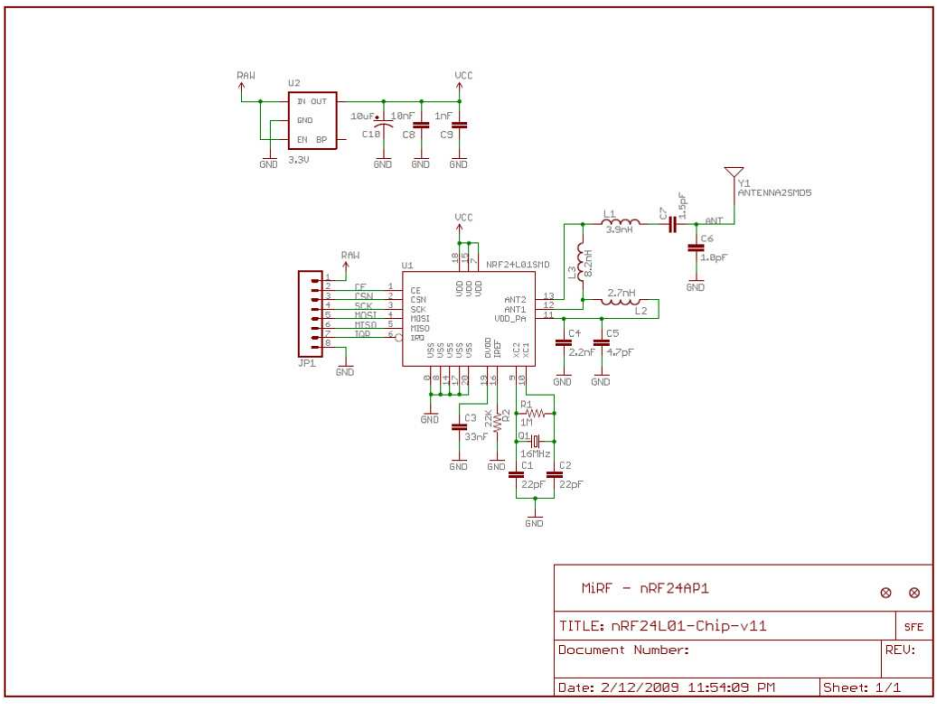


Figure 9.2: A schematic of the Receiver [30].

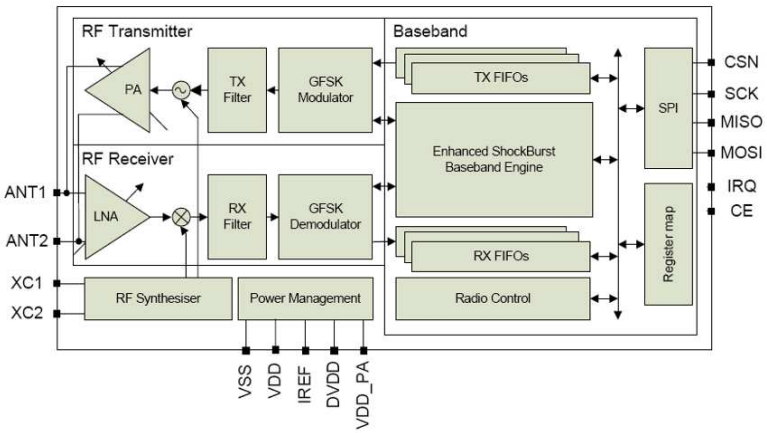


Figure 9.3: A block diagram of the nRF24L01+ chip which is located on the Receiver [22].

More information about the used Receiver can be found at [22].

The crucial parts that are needed in order to receive are: an Antenna, a LNA, a RF Synthesiser, a filter and a GFSK Demodulator.

Obviously an antenna is needed in order to receive any incoming data. Because the incoming data is a low power signal it needs to be amplified in order to recover the original signal. The incoming data will therefore be amplified by the Low Noise Amplifier (LNA). To be able to filter the useful information out of the incoming signal, the amplified signal is mixed with a Radio Frequency Synthesiser. The signal is past through a Gaussian Frequency Shift Keying Demodulator in order to create binary data. Once the input signal is converted to binary data, it is stored into the FIFOs (First In First Out) and can be read out via SPI.

9.2.2 The possible circuit on the Super E-paper

As explained in the previous paragraph the most important components needed to recreate the Receiver are an Antenna, a LNA, an RF Synthesiser, a filter and a GFSK Demodulator. Only these components are necessary to implement on the Super E-paper, because the binary data created by this circuit can directly be stored into the memory or immediately used. Are these components realisable on the Super E-paper? This question will be answered in this paragraph.

Research has shown it is possible to create a 433MHz ISM band RF amplifier (more information can be found at [31]). The needed RF receiver has to be at the 2.4Ghz band. To create such a receiver the Quality factor (Q-factor) has to be improved and the speed of the transistors has to be increased.

The Q-factor is defined as:

$$Q = \frac{f_0}{B} \tag{9-1}$$

f_0 is the resonant frequency and B is the 3-dB bandwidth. The larger the value of the Q-factor, the smaller the bandwidth will be [4].

To be able to process the incoming data the transistors, which are used to create the whole circuit, need to be fast enough. The speed at which the 433MHz RF amplifier still worked is to slow to create a 2.4GHz receiver. To increase the speed of the transistors the

single grain Silicon substrate can be replaced by an insulating glass or plastic substrate. Another way to increase the speed is to use smaller transistors.

The answer to the question whether it is possible to implement the necessary components on the Super E-paper is: *Yes*, when these changes (a higher Q-factor and a higher operating frequency of the transistors) are met, theoretically a 2.4GHz RF receiver can be implemented on the Super E-paper.

9.3 Conclusion

The test with a simulation of the Super E-paper was a success. Even though the rearranging function could not be used, the test did prove a picture could be send to a “Super E-paper”.

By evaluation of the necessary components in the nRF24L01+ module, the Receiver, and with the help of previous research results, another conclusion can be drawn. When the Q-factor and the operating speed of the transistors are raised, theoretically a 2.4GHz RF receiver can be implemented on the Super E-paper. After implementation on the Super E-paper everything else, the Transmission program and Transmitter, can still be used.

10. Conclusions and recommendations

The goal of this research was to determine whether it would be possible to create a kit that uses one-way radio-frequency communication to transmit an image without loss of quality to a receiver connected to an E-paper, by using a user-friendly and very simple computer program. This chapter contains the conclusions that can be drawn based on this research and recommendations for further research and improvements on the developed kit.

Paragraph 1 will discuss the conclusions based on the test results from chapters 8 and 9. Paragraph 2 will explain how the developed kit could be further improved and how further research could lead to better Error Control.

10.1 Conclusions

It is possible to use one-way radio-frequency communication to transmit an image without loss of quality to a receiver connected to an E-paper. We developed a kit to test this, which consists of four components:

- A computer program, where the user can select an image to transmit.
- An RF transmitter.
- A microcontroller, which acts as a bridge between the computer and the transmitter.
- An RF receiver, which has to the pass the image to the hardware of the E-paper.

For testing purposes, we added another microcontroller to acts as a bridge between the RF receiver and the hardware of the E-paper. The performance of both the hardware and the software was tested and will be discussed in the following sections.

10.1.1 Software

The developed software was tested by a test group. This test group was very positive about the software. The software is very user-friendly. It was developed in MATLAB and turned into a standalone application so the software can be used without having to install MATLAB. The software allows a user to send any image or PDF file on his computer to all his E-papers. By logging in with a username and password, the software makes sure that only the users' E-papers receive the image. The program continuously informs the user about the progress during the transmission and allows the user to abort the transmission at every given moment accept when converting a PDF file to an image.

10.1.2 Hardware

The developed hardware is able to transmit an image with very high quality over distance shorter than 28 meters using one-way radio-frequency communication. Increasing the distance will increase the Bit Error Rate, which means that too many errors will occur and not enough data will be received to show an image of high quality. If an advertisement company wants to change all its E-papers within a larger radius, that will only be possible if an antenna with more gain and more power is used. In reality, using an antenna with more gain and more power should be able to guarantee an error rate less than 10^{-3}, which means that less than 0,1% of the transmitted bits may contain errors. If this error rate is achieved, the Error Control used in this research will be able to guarantee high quality transmission. Therefore it is possible to create a kit that uses one-way radio-frequency communication to transmit an

image without loss of quality, but the kit developed in this research can only do this within a short range.

10.2 Recommendations

Even though the goals of this thesis are met, some further research and development can be done to make the software more user-friendly and to increase the quality of the transmission.

10.2.1 Improving the software

The test group responsible for testing the software had one important remark to improve the user-friendliness of the software. Most of all, they would like the software to open faster and install easier. This can't be achieved in MATLAB. Writing the program in a different language might solve this problem, but that might also make the image processing more complicated.

The speed of the transmission could be improved as well. One obvious improvement might be applied. In this thesis, the picture is transmitted by storing the image in three matrices: one for the intensity of red, one for green and one for blue. This was done, because the receiving side needed the image in this format to show it on a screen. However, file compression could reduce the size of an image enormously. Our software should be able to transmit an image in a different format, for instance jpeg. This would mean less data has to be transmitted to show the same image and thus the transmission time would decrease linearly with the decrease in file size. However, this requires E-paper hardware that understands jpeg files and that is able to show a jpeg file, even if some bytes are missing due to transmission errors.

The last improvement that could be made requires knowledge of USB ports. We used a virtual COM port to send the data from the computer to the transmitter. This requires the installation of drivers. Furthermore, the device has to be connected to the computer before the software is started. It would be preferable to make full use of the USB port functionalities. USB is capable of recognizing a connected device at any given moment when the computer is running. It should be able to implement this so called *hot plugging* into the software. See appendix C for more information on this topic.

10.2.2 Improving the Error Control

When a packet is send with the current protocol, the transmission is either a success or a failure. Each packet is transmitted multiple times. If the transmission is a success, the packet is passed on to the E-paper. If it is a failure, the packet is discarded. This might be improved by using redundancy in a different way. The following alternatives might be tested:

- Instead of retransmitting packets, another repetition code might be used as forward error control by transmitting every bit multiple times within a byte (like discussed in section 3.2.1). CRC is no longer necessary with this technique. The big advantage of that repetition code is the big increase in received number of packets. Packets that contain only a single error are no longer rejected by a CRC. On top of that, the repetition code might be able to fix the error. However, this technique also has big disadvantages. For instance, there is no guarantee that the received data is correct. This means that a pixel might be given a completely wrong value if one of the four upper bits from a byte is transmitted wrong. Furthermore, an error in the address of a pixel might set the data to the wrong pixel. We wrote a transmission and testing

program to test which redundancy method would be optimal. The bitwise redundancy could easily be implemented into our program. However, due to problems with timing the decision was made to exclude these tests from our research. The files necessary to perform this research can be found in appendix A.

- A combination of both techniques is also a possibility: transmitting packets with a repetition code and a CRC in the payload. When the same number of bits is transmitted, this might increase or decrease the quality of the transmission compared to the error control used in this research.
- Another option might be varying the length of the payload within a packet. When a packet is lost, the entire payload is lost. If the payload contains 32 bytes, a whole lot more data is lost compared to a payload of 4 bytes. However, every payload requires the same number of address bytes to target the right receiver and the same number of bytes in the Packet Control Field. The packet will be lost if there are errors in these fields. By increasing the payload size, fewer bits have to be transmitted to the receiver per byte of data. This might mean fewer errors can occur and more packets will make it to the receiver. However, increasing the packet size will increase the chance on a failed transmission significantly. An optimal packet size should be found theoretically and experimentally.

10.2.3 Improving the hardware

The hardware can be improved in two ways. One way is to use an antenna with a higher gain. The same transmitter module could be used as in this thesis, but with a different antenna. Secondly, more power could be used to transmit the data.

Both changes would improve the Bit Error Rate. With these changes, it should be possible to transmit data over a larger distance with a Bit Error Rate of less then 10^{-3}, making one-way communication using radio-frequency possible.

Bibliography

Chapter 1:
[1] M. de Jong, C.J. Kruit and others (June 2010) "*Business Plan Reflex*." TU Delft.

Chapter 3:
[2] Piet van Mieghem, *Data Communications Networking.*.Amsterdam: Techne Press, 2006, pp. 54-71.

[3] K. Sam Shanmugam, *Digital and Analog Communication Systems.* New York: John Wiley & Sons, 1979, pp. 443-496.

[4] Leon W. Couch, II, *Digitaland Analog Communication Systems*, 7th ed. Upper Saddle River: Pearson Prentice Hall, 2007, pp 19-24 and 246-247.

[5] Emina Soljanin, Ruoheng Liu, Predrag Spasojevic. "Hybrid ARQ with Random Transmission Assignments". In *Advances in Network Information Theory: DIMACS Workshop Network Information Theory*, v. 66. Piscataway: DIMACS series in discrete mathematics and theoretical computer science, 2004, pp. 321-335 .

Chapter 4:
[6] The Mathworks (May 16 2010) MATLAB - The Language Of Technical Computing. Available: http://www.mathworks.com/products/matlab/

[7] The Mathworks (May 17 2010) Creating Scripts with the MATLAB Editor/Debugger. Available: http://www.mathworks.com/academia/student_center/tutorials/creating_scripts.html

[8] Matlab Support (April 20 2010). Function Handles. Available: http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/f2-38133.html

[9] Moulos, Panagiotis (June 13 2007). Showinfowindow. MATLAB Central. Available: http://www.mathworks.com/matlabcentral/fileexchange/25471-maximize

[10] Artifex Software (May 11 2010). Gs871w32. Ghostscript. Available: http://www.ghostscript.com/

[11] Woodford, Oliver (April 20 2010). Ghostscript. MATLAB Central. Available: http://www.mathworks.com/matlabcentral/fileexchange/23629-exportfig

[12] University of Wisconsin (April 28 2010). How to use Ghostscript. Available: http://pages.cs.wisc.edu/~ghost/doc/cvs/Use.htm

[13] Pfeifle, Kurt (April 28 2010). Tech Tip: Using Ghostscript to Convert and Combine Files. Linus Journal. Available: http://www.linuxjournal.com/content/tech-tip-using-ghostscript-convert-and-combine-files

[14] FTDI (March 31 2010) D2XX Drivers. http://www.ftdichip.com/Drivers/D2XX.htm

[15] The Mathworks (May 16 2010) Working with the MCR. http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/f12-999353.html

[16] The Mathworks (May 12 2010) Bug 217007. Bug report. Available: http://www.mathworks.com/support/bugreports/217007

Chapter 5:

[17] Sparkfun Electronics (May 24 2010). Available: http://www.sparkfun.com/commerce/product_info.php?products_id=691

[18] ePanorama (24 May 2010). "Serial buses information page." Available: http://www.epanorama.net/links/serialbus.html

[19] D. Kalinsky & R. Kalinsky (January 2 2010). "Introduction to Serial Peripheral Interface." Embedded systems design. Available: http://embedded.com/columns/beginerscorner/9900483?_requestid=245610

[20] Wikipedia (24 May 2010). "Serial Peripheral Interface Bus," Available: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[21] Brennen Ball (2007). "*Tutorial 0: Everything You Need to Know about the nRF24L01 and MiRF-v2."* DIY Embedded. Available: http://www.diyembedded.com/tutorials/nrf24l01_0/nrf24l01_tutorial_0.pdf

[22] nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0 (September 2008). Available: http://www.nordicsemi.com/files/Product/data_sheet/nRF24L01P_Product_Specification_1_0.pdf

Chapter 6:

[23] Sparkfun (April 27 2010) Available: http://www.sparkfun.com/commerce/product_info.php?products_id=9235

Chapter 7:

[24] Mathworks (May 19 2010). Bug 250986. Bug reports. Available: http://www.mathworks.com/support/bugreports/250986

[25] Davide Ferraro. Correspondance about cursor behaviour in edit text. June 8 2010. See appendix D.

Chapter 8:

[26] G. Janssen. Correspondence about improving the Bit Error Rate. June 1 2010.

Chapter 9:

[27] Joost Meerwijk, Willem Zwetsloot (June 2010) "*Design of an electronic billboard. Complementing R:eFlex' business plan"* TU Delft.

[28] DigilentInc (June 8 2010) Available: http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,799&Prod=S3BOARD

[29] MicksXboxModS (June 8 2010) Available: http://www.micksxboxmods.com/store/images/psp_tft_1000_series.JPG

[30] Sparkfun (December 2009) Available: http://www.sparkfun.com/datasheets/Wireless/Nordic/nRF24L01-Chip-v11.pdf

[31] Ryoichi Ishihara and others, "An Assessment of μ-Czochralski, Single-Grain Silicon Thin-Film Transistor Technology for Large-Area, Sensor and 3-D Electronic Integration", *IEEE Journal of Solid-State Circuits*, vol. 43, no. 7, pp. 1563-1576, July 2008.

Appendix B:

[32] Arduino (Mei 12 2010) Available: http://arduino.cc/en/Main/ArduinoBoardDuemilanove

Appendix C:

[33] Discovery (19 May 2010) "*How Parallel Ports Work,"* How stuff works, Available: http://computer.howstuffworks.com/parallel-port1.htm

[34] Computer Hope, (19 May 2010) *"What's hot-swappable or can be unplugged while computer is on?"* Available: http://www.computerhope.com/issues/ch001059.htm

[35] Wikipedia (19 May 2010)*"Parallelle Poort,"* Available*:* http://nl.wikipedia.org/wiki/Parallelle_poort

[36] Discovery (19 May 2010) "*How Serial Ports Work,"* How stuff works, Available: http://computer.howstuffworks.com/serial-port.htm/printable

[37] Ergo Canada, (19 May 2010) Available: http://www.ergocanada.com/ergo/tips/serial_port.jpg

[38] D. Anderson and D. Dzatko, "Universal Serial Bus System Architecture," Addison-Wesley, 2001, pp. 13-24 and pp. 141-156.

[39] RAD Data Communications (5 May 2010) Available: http://www3.rad.com/networks/2000/usb/maintxt.htm#USB_Protocol

[40] A. Ricci Bitti (19 May 2010) *"USB type-A plug,"* Available: http://www.riccibitti.com/pc_therm/usb_pc_therm.htm

Appendices

Appendix A: *MATLAB Codes*

A.1 The transmission software

change_value.m

```
%CHANGE_VALUE  Change the value assigned to a unique variable in a file
%
% Examples:
%   fail = change_value(value)
%   fail = change_value(value, variableName)
%   fail = change_value(value, variableName, filePath)
%
% Function to change the value assigned to a variable in a text file. The
% assignment must exist already, and must be on a line on its own. For
% example:
%    variableName = 'string';
%    variableName = -0.756e-8;
% Note that there must be one or more spaces either side of the = sign.
% Only the first such assignment is changed.
%
% IN:
%   value - The value to be assigned to the variable.
%   variableName - String containing the name of the variable whose value
%                  is to be set. Default: name of variable given as value.
%   filePath - Full path of the file to change. Default: path of calling
%              file.
%
% OUT:
%   fail - true if change failed, false otherwise.
%==========================================================================
% Copyright (c) 2009, Oliver Woodford
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the
%       distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.
%

function fail = change_value(value, variableName, filePath)
% Check for missing inputs
if nargin < 3
    % Get the filename of the calling function
    filePath = dbstack;
    filePath = which(filePath(2).file);
    if nargin < 2
        % Get the variable name
        variableName = inputname(1);
    end
end
fail = true;
% Read in the file
```

```
fh = fopen(filePath, 'rt');
if fh < 0
    return
end
fstrm = fread(fh, '*char')';
fclose(fh);
% Find the path
first_sec = regexp(fstrm, ['[\n\r]+ *' variableName ' += +'], 'end',...
            'once');
second_sec = first_sec + regexp(fstrm(first_sec+1:end), ';? *[\n\r]+',...
            'once');
if isempty(first_sec) || isempty(second_sec)
    return
end
% Create the changed line
if ischar(value)
    str = '''%s''';
else
    str = '%1.50g';
end
str = sprintf(str, value);
% Save the file with the value changed
fh = fopen(filePath, 'wt');
if fh < 0
    return
end
fprintf(fh, '%s%s%s', fstrm(1:first_sec), str, fstrm(second_sec:end));
fclose(fh);
fail = false;
return
```

contains.m

```
function pos = contains(string, part)

% function pos = contains(string, part)
% If the charachters in "part" are contained in "string" in the same order,
% a '1' is returned. '0' otherwise.

Lp = length(part);
Ls = length(string);

if Lp > Ls
    pos = 0;
else
    pos = 0;
    for i = 1:(Ls - Lp + 1)
        for k = 1:Lp
            if string(i+k-1) ~= part(k)
                break
            end

            if k == Lp
                pos = 1;
                break
            end
        end

        if pos == 1
            break
        end


    end
end
```

endsWith.m

```
function pos = endsWith(string, part)
```

```
% If "string" ends with the charachters in "part", a '1' is
% returned. 0 otherwise.

Lp = length(part);
Ls = length(string);

if Lp > Ls
    pos = 0;
else
    pos = [];
    for i = 1:Lp
        if string(Ls - Lp + i) == part(i);
            pos = 1;
        else
            pos = 0;
            break
        end
    end
end
```

get3bytes.m

```
function [byte1 byte2 byte3] = get3bytes(byte)

% This function will encode bytes. Every bit in a byte will be send three
% times, which will be usefull for transmitting data over a channel with a
% high BER. The start- and stopbits won't be send three times, but still
% before and after every byte of data.


bit = 0;
byte1 = uint8(0);
byte2 = uint8(0);
byte3 = uint8(0);

for j = 1:8
    bit = bitget(byte,j);
        if bit == 1
            if j == 1
                byte1 = bitset(byte1,1);
                byte1 = bitset(byte1,2);
                byte1 = bitset(byte1,3);
            elseif j == 2
                byte1 = bitset(byte1,4);
                byte1 = bitset(byte1,5);
                byte1 = bitset(byte1,6);
            elseif j == 3
                byte1 = bitset(byte1,7);
                byte1 = bitset(byte1,8);
                byte2 = bitset(byte2,1);
            elseif j == 4
                byte2 = bitset(byte2,2);
                byte2 = bitset(byte2,3);
                byte2 = bitset(byte2,4);
            elseif j == 5
                byte2 = bitset(byte2,5);
                byte2 = bitset(byte2,6);
                byte2 = bitset(byte2,7);
            elseif j == 6
                byte2 = bitset(byte2,8);
                byte3 = bitset(byte3,1);
                byte3 = bitset(byte3,2);
            elseif j == 7
                byte3 = bitset(byte3,3);
                byte3 = bitset(byte3,4);
                byte3 = bitset(byte3,5);
            else
                byte3 = bitset(byte3,6);
                byte3 = bitset(byte3,7);
                byte3 = bitset(byte3,8);
            end
        end
```

```
end
```

getName.m

```
function Name = getName(path)

% Name returns the filename of a function specified by the path in "path".
% Example:
%    Name = getName('H:\Desktop\BAP\Test.pdf')
%
%    Returns: Name = Test.pdf

Name = path;

L = length(path);
for i = 0:(L-1)
    if path(L-i) == '\'
       Name = path(L-i+1:L);
       break
    end
end
```

ghostscript.m

```
function varargout = ghostscript(cmd)
%GHOSTSCRIPT  Calls a local GhostScript executable with the input command
%
% Example:
%   [status result] = ghostscript(cmd)
%   ghostscript('-sDEVICE=jpeg -dNOPAUSE -dBATCH -dSAFER -r600x600
%   -sOutputFile=p%08d.jpg Testpdf.pdf')
%
% Attempts to locate a ghostscript executable, finally asking the user to
% specify the directory ghostcript was installed into. The resulting path
% is stored for future reference.
%
% Once found, the executable is called with the input command string.
%
% This function requires that you have Ghostscript installed on your
% system. You can download this from: http://www.ghostscript.com
%
% IN:
%   cmd - Command string to be passed into ghostscript.
%
% OUT:
%   status - 0 iff command ran without problem.
%   result - Output from ghostscript.


%% Disclaimer
% Copyright (c) 2009, Oliver Woodford
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the
%       distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.
%% End disclaimer

% Thanks to Jonas Dorn for the fix for the title of the uigetdir window on
% Mac OS.

% Thanks to Nathan Childress for the fix to the default location on 64-bit
% Windows systems.

% Call ghostscript
drawnow;
[varargout{1:nargout}] = system(sprintf('"%s" %s', gs_path, cmd));
drawnow;
return

function path = gs_path
% Return a valid path
% Start with the currently set path
path = current_gs_path;
% Check the path works
if check_gs_path(path)
    return
end
% Check whether the binary is on the path
if ispc
    bin = 'gswin32c.exe';
else
    bin = 'gs';
end

drawnow;

if check_store_gs_path(bin)
    path = bin;
    return
end
% Search the obvious places
if ispc
    default_location = 'C:\Program Files\gs\';
    dir_list = dir(default_location);
    if isempty(dir_list)
        % Possible location on 64-bit systems
        default_location = 'C:\Program Files (x86)\gs\';
        dir_list = dir(default_location);
    end
    executable = '\bin\gswin32c.exe';
    ver_num = 0;
    % If there are multiple versions, use the newest
    for a = 1:numel(dir_list)
        ver_num2 = sscanf(dir_list(a).name, 'gs%g');
        if ~isempty(ver_num2) && ver_num2 > ver_num
            path2 = [default_location dir_list(a).name executable];
            if exist(path2, 'file') == 2
                path = path2;
                ver_num = ver_num2;
            end
        end
    end
else
    path = '/usr/local/bin/gs';
end
if check_store_gs_path(path)
    return
end
% Ask the user to enter the path
while 1
    if strncmp(computer, 'MAC', 3) % Is a Mac
        % Give separate warning as the uigetdir dialogue box doesn't have a
        % title
        uiwait(warndlg('Ghostscript not found. Please locate the program.'))
    end
```

```
    base = uigetdir('/', 'Ghostcript not found. Please locate the program.');
    if isequal(base, 0)
        % User hit cancel or closed window
        break;
    end
    base = [base filesep];
    bin_dir = {'', ['bin' filesep], ['lib' filesep]};
    for a = 1:numel(bin_dir)
        path = [base bin_dir{a} bin];
        if exist(path, 'file') == 2
            break;
        end
    end
    if check_store_gs_path(path)
        return
    end
end
showinfowindow('Make sure Ghostscript is installed.','Ghostscript not found')
drawnow;
error('Ghostscript not found.');

function good = check_store_gs_path(path)
% Check the path is valid
good = check_gs_path(path);
if ~good
    return
end
%Update the current default path to the path found
if change_value(path, 'current_gs_path_str', [mfilename('fullpath') '.m'])

    return
end
return

function good = check_gs_path(path)
% Check the path is valid
[good message] = system(sprintf('"%s" -h', path));
good = good == 0;
return

function current_gs_path_str = current_gs_path
current_gs_path_str = 'C:\Program Files\gs\gs8.54\bin\gswin32c.exe';
return
```

GUI_BEP.m

```
function varargout = GUI_BEP(varargin)
%% GUI_BEP M-file for GUI_BEP.fig
%   GUI_BEP opens is the Gobal User Interface that allows you to
%   transmit an image to an e-paper.
%
%   H = GUI_BEP returns the handle to a new GUI_BEP or the handle to
%   the existing singleton*.
%
%   GUI_BEP('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in GUI_BEP.M with the given input arguments.
%
%   GUI_BEP('Property','Value',...) creates a new GUI_BEP or raises the
%   existing singleton*.  Starting from the left, property value pairs are
%   applied to the GUI before GUI_BEP_OpeningFcn gets called.  An
%   unrecognized property name or invalid value makes property application
%   stop.  All inputs are passed to GUI_BEP_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_BEP

% Last Modified by GUIDE v2.5 09-Jun-2010 18:09:57

%% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_BEP_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_BEP_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


%% --- Executes just before GUI_BEP is made visible.
function GUI_BEP_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_BEP (see VARARGIN)

% Choose default command line output for GUI_BEP
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI_BEP wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_BEP_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


%% --- Executes on button press in pushbutton1 = Browse.
function pushbutton1_Callback(hObject, eventdata, handles)
[fileName, Path] = uigetfile( {'*.jpg;*.jpeg;',...
                'Picture Files (*.jpg,*.jpeg)';'*.gif',...
                'GIF bestanden (*.gif)';'*.bmp',...
                'Bitmap (*.bmp)';'*.pdf',...
                'Adobe PDF files';'*.*','All Files (*.*)'},...
                'Select an image');
if fileName == 0
    % Als het uigetfile window wordt weggeklikt, komt er 'Browse->>' in de
    % edit text te staan.
    if isempty(get(handles.edit1,'string'))
    set(handles.edit1,'string','Browse ->>');
    setappdata(handles.pushbutton2,'PDFCheck',0);
    end
else
    % Als er een bestand wordt geselecteerd, komt dat in de edit text te
    % staan.
    set(handles.edit1,'string',[Path fileName]);
    setappdata(hObject,'Check',1);
    setappdata(handles.pushbutton2,'PDFCheck',0);
end

%% --- Executes on button press in pushbutton2 = Preview.
function pushbutton2_Callback(hObject, eventdata, handles)
fileName = get(handles.edit1,'string');       % Get filename from edit text
deletepdf = 0;

if isempty(fileName)|| contains(fileName,'Browse ->>')
```

```
    return;
end


    PDFCheck = getappdata(handles.pushbutton2,'PDFCheck');

    Lengte = get(handles.edit7,'string');
    % No comma's should be used. Changed to points by this for loop.
        for i = 1:length(Lengte)
            if Lengte(i) == ','
                Lengte(i) = '.';
            end
        end
        Lengte = floor(str2double(Lengte));    %Rounded
        if isempty(Lengte)
            Lengte = 120;
        end

    Breedte = get(handles.edit8,'string');
    % No comma's should be used. Changed to points by this for loop.
        for i = 1:length(Breedte)
            if Breedte(i) == ','
                Breedte(i) = '.';
            end
        end
        Breedte = floor(str2double(Breedte));  %Rounded
        if isempty(Breedte)
            Breedte = 120;
        end

    if endsWith(fileName,'.pdf')                % Test if a pdf is selected
        deletepdf = 1;
        if PDFCheck == 1
            deletepdf = 0;
            PDF = getappdata(handles.pushbutton2,'PDF');
            fileSource = PDF;
        else
            pdffileName = pdf2jpg(fileName);               % Convert pdf to image
            fileSource = uint8(imread(pdffileName));
            delete(pdffileName);
        end
        setappdata(handles.pushbutton2,'PDFCheck',1);
        setappdata(handles.pushbutton2,'PDF',fileSource);

    else
        fileSource = uint8(imread(fileName));
    end


    if get(handles.No, 'Value') == 0
        if get(handles.Portrait,'Value') == 1
            % Used for Portrait with black borders. 3 means turned over 270
            % degrees.
           New_Image = turnImage(resize(fileSource,1,1,Lengte,Breedte),3);
        else
            % Used for Landscape with black borders.
            New_Image = resize(fileSource,0,1,Lengte,Breedte);
        end
    else
        if get(handles.Portrait,'Value') == 1
            % Used for Portrait without borders. 3 means turned over 270
            % degrees.
           New_Image = turnImage(resize(fileSource,1,0,Lengte,Breedte),3);
        else
            % Used for Landscape without borders.
            New_Image = resize(fileSource,0,0,Lengte,Breedte);
        end
    end

    imshow(New_Image);


%% --- Callback for edit text 'browse'
function edit1_Callback(hObject, eventdata, handles)
setappdata(handles.pushbutton2,'PDFCheck',0);
```

```
%% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%% --- Executes on button press in pushbutton3 = Send Data.
function pushbutton3_Callback(hObject, eventdata, handles)
fileName = get(handles.edit1,'string');

if isempty(fileName)|| contains(fileName,'Browse ->>')
    return;
end

global username
foundCom = getappdata(handles.pushbutton10,'foundCom');
PDFCheck = getappdata(handles.pushbutton2,'PDFCheck');
deletepdf = 0;

if endsWith(fileName,'.pdf')     % Test if a pdf is selected
    deletepdf = 1;
        if PDFCheck == 1
            deletepdf = 0;
            PDF = getappdata(handles.pushbutton2,'PDF');
            fileSource = PDF;
        else
            pdffileName = pdf2jpg(fileName);               % Convert pdf to image
            fileSource = uint8(imread(pdffileName));
            delete(pdffileName);
        end
    setappdata(handles.pushbutton2,'PDFCheck',1);
    setappdata(handles.pushbutton2,'PDF',fileSource);
else
    fileSource = uint8(imread(fileName));
end

Lengte = get(handles.edit7,'string');
% No comma's should be used. Changed to points by this for loop.
    for i = 1:length(Lengte)
        if Lengte(i) == ','
            Lengte(i) = '.';
        end
    end
    Lengte = floor(str2double(Lengte));
    if isempty(Lengte)
        Lengte = 120;
    end

Breedte = get(handles.edit8,'string');
% No comma's should be used. Changed to points by this for loop.
    for i = 1:length(Breedte)
        if Breedte(i) == ','
            Breedte(i) = '.';
        end
    end
    Breedte = floor(str2double(Breedte));
    if isempty(Breedte)
        Breedte = 120;
    end

    if get(handles.No, 'Value') == 0
        if get(handles.Portrait,'Value') == 1
            % Same as in preview, only this time the image isn't turned
            % because the screen is written row by row.
           New_Image = resize(fileSource,1,1,Lengte,Breedte);
        else
            New_Image = resize(fileSource,0,1,Lengte,Breedte);
        end
    else
        if get(handles.Portrait,'Value') == 1
           New_Image = resize(fileSource,1,0,Lengte,Breedte);
        else
            New_Image = resize(fileSource,0,0,Lengte,Breedte);
        end
```

```
    end

% Send data to Arduino
foundCom = sendData(New_Image, 28800,hObject,username,foundCom);
setappdata(handles.pushbutton10,'foundCom',foundCom);

%% --- Executes during object creation, after setting all properties.
%% --- Initializes axes 4
function axes4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4
axes(hObject);
New_Image = 255*ones(480,480);            % Create a white window
imshow(New_Image);

%% --- Executes on button press in pushbutton8 = middelste Browse
function pushbutton8_Callback(hObject, eventdata, handles)

[fileName, Path] = uigetfile( {'*.jpg;*.jpeg;','Picture Files (*.jpg, *.jpeg)';'*.gif','GIF
bestanden (*.gif)';'*.bmp','Bitmap (*.bmp)';'*.pdf','Adobe PDF files';'*.*','All Files
(*.*)'},'Select an image');
if fileName == 0
    % Als het uigetfile window wordt weggeklikt, komt er 'Browse->>' in de
    % edit text te staan.
    if isempty(get(handles.edit4,'string'))
    set(handles.edit4,'string','Browse ->>');
    end
else
    % Als er een bestand wordt geselecteerd, komt dat in de edit text te
    % staan.
    set(handles.edit4,'string',[Path fileName]);
    setappdata(hObject,'Check',1);
end


function edit4_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'Visible','Off');


%% Pushbutton9 callback = bovenste browse
% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)

[fileName, Path] = uigetfile( {'*.jpg;*.jpeg;','Picture Files (*.jpg, *.jpeg)';'*.gif','GIF
bestanden (*.gif)';'*.bmp','Bitmap (*.bmp)';'*.pdf','Adobe PDF files';'*.*','All Files
(*.*)'},'Select an image');
if fileName == 0
    % Als het uigetfile window wordt weggeklikt, komt er 'Browse->>' in de
    % edit text te staan.
    if isempty(get(handles.edit5,'string'))
    set(handles.edit5,'string','Browse ->>');
    end
else
    % Als er een bestand wordt geselecteerd, komt dat in de edit text te
    % staan.
    set(handles.edit5,'string',[Path fileName]);
    setappdata(hObject,'Check',0);
end


function edit5_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'Visible','Off');


%% --- Executes on button press in pushbutton10 = Send Slideshow
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fileName3 = get(handles.edit1,'string');
fileName2 = get(handles.edit4,'string');
fileName1 = get(handles.edit5,'string');

global username

 try
    [fileName1 fileName2 fileName3 n] = simplify(fileName1,fileName2,fileName3);
 catch
     showinfowindow('No files selected, please select one or more pictures.','Error');
     return;
 end

Lengte = get(handles.edit7,'string');
% No comma's should be used. Changed to points by this for loop.
    for i = 1:length(Lengte)
        if Lengte(i) == ','
            Lengte(i) = '.';
        end
    end
    Lengte = floor(str2double(Lengte));
    if isempty(Lengte)
        Lengte = 120;
    end

Breedte = get(handles.edit8,'string');
% No comma's should be used. Changed to points by this for loop.
    for i = 1:length(Breedte)
        if Breedte(i) == ','
            Breedte(i) = '.';
        end
    end
    Breedte = floor(str2double(Breedte));
    if isempty(Breedte)
        Breedte = 120;
    end

foundCom = getappdata(hObject,'foundCom');

deletepdf1 = 0;
deletepdf2 = 0;
deletepdf3 = 0;

if endsWith(fileName1,'.pdf') && n > 0         % Test if a pdf is selected
    deletepdf1 = 1;
    fileName1 = pdf2jpg(fileName1);            % Convert pdf to image
    pdfim1 = fileName1;
end

if endsWith(fileName2,'.pdf') && n > 1         % Test if a pdf is selected
    deletepdf2 = 1;
    fileName2 = pdf2jpg(fileName2);            % Convert pdf to image
    pdfim2 = fileName2;
end

if endsWith(fileName3,'.pdf') && n > 2         % Test if a pdf is selected
    deletepdf3 = 1;
    fileName3 = pdf2jpg(fileName3);            % Convert pdf to image
    pdfim3 = fileName3;
end
```

```
if n == 0
    return;
elseif n == 1
    fileName1 = imread(fileName1);
    if get(handles.No, 'Value') == 0
        if get(handles.Portrait,'Value') == 1
            % Same as in preview, only this time the image isn't turned
            % because the screen is written row by row.
           New_Image = resize(fileName1,1,1,Lengte,Breedte);
        else
            New_Image = resize(fileName1,0,1,Lengte,Breedte);
        end
    else
        if get(handles.Portrait,'Value') == 1
           New_Image = resize(fileName1,1,0,Lengte,Breedte);
        else
            New_Image = resize(fileName1,0,0,Lengte,Breedte);
         end
    end

foundCom = sendData(New_Image,28800,hObject,username,foundCom);

elseif n == 2
     % Same as in preview, only this time the image isn't turned because
     % the screen is written row by row.
    fileName1 = imread(fileName1);
    fileName2 = imread(fileName2);
    New_Image = zeros(Lengte,Breedte,3,n);
    if get(handles.No, 'Value') == 0
        if get(handles.Portrait,'Value') == 1
           New_Image(:,:,:,1) = resize(fileName1,1,1,Lengte,Breedte);
           New_Image(:,:,:,2) = resize(fileName2,1,1,Lengte,Breedte);
        else
            New_Image(:,:,:,1) = resize(fileName1,0,1,Lengte,Breedte);
            New_Image(:,:,:,2) = resize(fileName2,0,1,Lengte,Breedte);
        end
    else
        if get(handles.Portrait,'Value') == 1
           New_Image(:,:,:,1) = resize(fileName1,1,0,Lengte,Breedte);
           New_Image(:,:,:,2) = resize(fileName2,1,0,Lengte,Breedte);
        else
            New_Image(:,:,:,1) = resize(fileName1,0,0,Lengte,Breedte);
            New_Image(:,:,:,2) = resize(fileName2,0,0,Lengte,Breedte);
        end
    end

foundCom = sendData(New_Image,28800,hObject,username,foundCom);

else
    fileName1 = imread(fileName1);
    fileName2 = imread(fileName2);
    fileName3 = imread(fileName3);
    New_Image = zeros(Lengte,Breedte,3,n);

    % Same as in preview, only this time the image isn't turned because the
    % screen is written row by row.
    if get(handles.No, 'Value') == 0
        if get(handles.Portrait,'Value') == 1
           New_Image(:,:,:,1) = resize(fileName1,1,1,Lengte,Breedte);
           New_Image(:,:,:,2) = resize(fileName2,1,1,Lengte,Breedte);
           New_Image(:,:,:,3) = resize(fileName3,1,1,Lengte,Breedte);
        else
            New_Image(:,:,:,1) = resize(fileName1,0,1,Lengte,Breedte);
            New_Image(:,:,:,2) = resize(fileName2,0,1,Lengte,Breedte);
            New_Image(:,:,:,3) = resize(fileName3,0,1,Lengte,Breedte);
        end
    else
        if get(handles.Portrait,'Value') == 1
           New_Image(:,:,:,1) = resize(fileName1,1,0,Lengte,Breedte);
           New_Image(:,:,:,2) = resize(fileName2,1,0,Lengte,Breedte);
           New_Image(:,:,:,3) = resize(fileName3,1,0,Lengte,Breedte);
        else
            New_Image(:,:,:,1) = resize(fileName1,0,0,Lengte,Breedte);
            New_Image(:,:,:,2) = resize(fileName2,0,0,Lengte,Breedte);
            New_Image(:,:,:,3) = resize(fileName3,0,0,Lengte,Breedte);
```

```
        end
    end

foundCom = sendData(New_Image,28800,hObject,username,foundCom);
end



setappdata(hObject,'foundCom',foundCom);

if deletepdf1 == 1
    delete(pdfim1);
end

try

if deletepdf2 == 1
    delete(pdfim2);
end

if deletepdf3 == 1
    delete(pdfim3);
end

catch
    return;
end

%% Tabblad Single Picture
% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)

New_Image = 255*ones(480,480);                  % Create a white window
imshow(New_Image);

% Set only all necessary objects visible
set(handles.pushbutton2,'Visible','On');
set(handles.pushbutton3,'Visible','On');
set(handles.pushbutton8,'Visible','Off');
set(handles.pushbutton9,'Visible','Off');
set(handles.pushbutton10,'Visible','Off');
set(handles.edit4,'Visible','Off');
set(handles.edit5,'Visible','Off');
set(handles.edit6,'Visible','Off');
set(handles.edit7,'Visible','On');
set(handles.edit8,'Visible','On');
set(handles.text4,'Visible','Off');
set(handles.text5,'Visible','Off');
set(handles.text6,'Visible','Off');
set(handles.text7,'Visible','Off');
set(handles.text8,'Visible','Off');
set(handles.text9,'Visible','Off');

set(handles.pushbutton1,'Visible','On');
set(handles.edit1,'Visible','On');
set(handles.text1,'Visible','On');
set(handles.uipanel3,'Visible','On');
set(handles.uipanel4,'Visible','On');

set(handles.text11,'Visible','Off');
set(handles.text12,'Visible','Off');
set(handles.text13,'Visible','Off');
set(handles.text14,'Visible','Off');
set(handles.text15,'Visible','Off');
set(handles.text17,'Visible','On');
set(handles.text18,'Visible','On');
set(handles.text19,'Visible','On');


%% Tabblad Slideshow
% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)

z = 85*ones(1,480);             % Create a white window
h = imshow(z);
```

```
% Set only all necessary objects visible
set(handles.pushbutton2,'Visible','Off');
set(handles.pushbutton3,'Visible','Off');
set(handles.pushbutton8,'Visible','On');
set(handles.pushbutton9,'Visible','On');
set(handles.pushbutton10,'Visible','On');
set(handles.edit4,'Visible','On');
set(handles.edit5,'Visible','On');
set(handles.edit6,'Visible','On');
set(handles.edit7,'Visible','On');
set(handles.edit8,'Visible','On');
set(handles.text4,'Visible','On');
set(handles.text5,'Visible','On');
set(handles.text6,'Visible','On');
set(handles.text7,'Visible','On');
set(handles.text8,'Visible','On');
set(handles.text9,'Visible','On');

set(handles.pushbutton1,'Visible','On');
set(handles.edit1,'Visible','On');
set(handles.text1,'Visible','On');
set(handles.uipanel3,'Visible','On');
set(handles.uipanel4,'Visible','On');

set(handles.text11,'Visible','Off');
set(handles.text12,'Visible','Off');
set(handles.text13,'Visible','Off');
set(handles.text14,'Visible','Off');
set(handles.text15,'Visible','Off');
set(handles.text17,'Visible','On');
set(handles.text18,'Visible','On');
set(handles.text19,'Visible','On');



% --- Executes during object creation, after setting all properties.
function text5_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function pushbutton8_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');

% --- Executes during object creation, after setting all properties.
function text4_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');

% --- Executes during object creation, after setting all properties.
function pushbutton9_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');

% --- Executes during object creation, after setting all properties.
function text6_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');

% --- Executes during object creation, after setting all properties.
function text7_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function pushbutton10_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');
setappdata(hObject,'Check',0);
setappdata(hObject,'Transmitter',0);
setappdata(hObject,'n',0);
setappdata(hObject,'foundCom',[])


% --- Executes during object creation, after setting all properties.
function text8_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');
```

```
function edit6_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function text9_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');


%% --- Executes on button press in pushbutton16 = Help
function pushbutton16_Callback(hObject, eventdata, handles)
z = 85*ones(1,480);             % Create a white window
h = imshow(z);

% Set only all necessary objects visible
set(handles.pushbutton1,'Visible','Off');
set(handles.pushbutton2,'Visible','Off');
set(handles.pushbutton3,'Visible','Off');
set(handles.pushbutton8,'Visible','Off');
set(handles.pushbutton9,'Visible','Off');
set(handles.pushbutton10,'Visible','Off');
set(handles.edit1,'Visible','Off');
set(handles.edit4,'Visible','Off');
set(handles.edit5,'Visible','Off');
set(handles.edit6,'Visible','Off');
set(handles.edit7,'Visible','Off');
set(handles.edit8,'Visible','Off');
set(handles.text1,'Visible','Off');
set(handles.text4,'Visible','Off');
set(handles.text5,'Visible','Off');
set(handles.text6,'Visible','Off');
set(handles.text7,'Visible','Off');
set(handles.text8,'Visible','Off');
set(handles.text9,'Visible','Off');
set(handles.uipanel3,'Visible','Off');
set(handles.uipanel4,'Visible','Off');

set(handles.text11,'Visible','On');
set(handles.text12,'Visible','On');
set(handles.text13,'Visible','On');
set(handles.text14,'Visible','On');
set(handles.text15,'Visible','On');
set(handles.text17,'Visible','Off');
set(handles.text18,'Visible','Off');
set(handles.text19,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function text11_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function text12_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function text13_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');


% --- Executes during object creation, after setting all properties.
function text14_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');
```

```
% --- Executes during object creation, after setting all properties.
function text15_CreateFcn(hObject, eventdata, handles)
set(hObject,'Visible','Off');



function edit7_Callback(hObject, eventdata, handles)

%setappdata(handles.pushbutton2,'PDFCheck',0);

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit8_Callback(hObject, eventdata, handles)
%setappdata(handles.pushbutton2,'PDFCheck',0);

% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,...
        'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function pushbutton2_CreateFcn(hObject, eventdata, handles)

setappdata(hObject,'PDFCheck',0);
setappdata(hObject,'PDFfile',0);


% --- Executes on button press in Landscape.
function Landscape_Callback(hObject, eventdata, handles)

%setappdata(handles.pushbutton2,'PDFCheck',0);


% --- Executes on button press in Portrait.
function Portrait_Callback(hObject, eventdata, handles)

%setappdata(handles.pushbutton2,'PDFCheck',0);


% --- Executes on button press in Yes.
function Yes_Callback(hObject, eventdata, handles)

%setappdata(handles.pushbutton2,'PDFCheck',0);


% --- Executes on button press in No.
function No_Callback(hObject, eventdata, handles)

%setappdata(handles.pushbutton2,'PDFCheck',0);
```

handshake.m

```
function [Com message] = handshake(Rate,time,GUIHandle)

% Handshake tries to find the Arduino by sending a message over all
% available COM ports. Only the Arduino will reply with the correct
% response.
% The Arduino has to respond within 'time'-seconds, with 100 milliseconds as
% default value.

if nargin < 3
    GUIHandle = 1; % Handshake moet vanuit een GUI aangeroepen worden.
```

```
end
if nargin < 2
    time = 0.1;
end
if nargin < 1
    Rate = 28800;
end

drawnow;
msg = showinfowindow('Searching Transmitter Module. This might take some time. Close this
window to stop searching.');

if ~isempty(instrfind)
    fclose(instrfind);
end

% Initialize Com with a number, it will become a string if the Arduino is
% found
Flag = 0;

% Search for all available serial ports
serialinfo = instrhwinfo('serial');

% Put all ports as strings in 'ports'. For example: Ports = ['COM1';
% 'COM3']
ports = serialinfo.SerialPorts;
hello = uint8(122);
drawnow;

% Every time a wrong port is greeted, a warning will be send, so turn
% warnings off temporary.
warning off all;
try

    for i = 1:length(ports)
        shake = ports(i);

        % Create Virtual COM port
        s1 = serial(shake, 'Baudrate', Rate,'Timeout',time);
        try

            fopen(s1);     % Open Port
            try
                pause(2.1);    % Pause needed for the Arduino

                if ~ishandle(msg) || ~ishandle(GUIHandle)
                    Flag = 2;
                    break
                end
                drawnow;

                    fwrite(s1,hello);        % Send greeting
                    answer = fread(s1,1);    % Scan for response
                    if answer == uint8(85)   % Arduino should answer with
                                             % 85 = 01010101
                        Com = char(shake);   % If the answer is received: Arduino found
                        Flag = 1;
                        fclose(s1);   % Close the COM port connected to the Arduino
                        break
                    end
                    fclose(s1);
            catch
                fclose(s1); % Always close the COM port, even when there are errors
                drawnow;
            end
        catch
            drawnow;
        end
    end

    if ishandle(msg)
        close(msg)
    end

    warning on all;
```

```
catch
    warning on all;
    if ishandle(msg)
        close(msg)
    end
end

message = 1;

if Flag == 0 && ishandle(GUIHandle)
    message = showinfowindow('Transmitter Module not found. Exit this program and restart it
with the Transmitter Module connected.','Error');
    drawnow;
    Com = [];    % Test for this with 'isempty(Com)'
elseif Flag == 2
    Com = [];
elseif ishandle(GUIHandle)
    %message = ['Transmitter Module found at ' Com];
    %message = showinfowindow(message,'Transmitter found');
    drawnow;
else
    Com = [];
    drawnow;
end
```

Login.m

```
function [tf UNG] = Login(str,username,time_out,a,b)
N = nargin;

if N < 2
    str = '0000';
    username = 'serv1';
    time_out = 3600;  % Stay in uiwait for 1 hour.
    a = 600;
    b = 600;
elseif N < 3
    time_out = 3600;
    a = 600;
    b = 600;
elseif N < 5
    a = 600;
    b = 600;
end

defaultBackground = get(0,'defaultUicontrolBackgroundColor');
S.PWG = [];  % Store the pasword entered.
S.fh = figure('units','pixels',...
              'position',[a b 320 100],...
              'menubar','none',...
              'color',defaultBackground,...
              'name','Verify Password.',...
              'resize','off',...
              'numbertitle','off',...
              'name','Log in');
S.untx = uicontrol('style','text',...
                'units','pix',...
                'position',[5 75 105 20],...
                'string','Username:',...
                'fontweight','bold',...
                'horizontalalign','left',...
                'fontsize',11);
S.un = uicontrol('style','edit',...
                'units','pix',...
                'position',[110 75 200 20],...
                'backgroundcolor','w',...
                'tooltipstring',' Enter your username here.',...
                'tag','usernametxt',...
                'HorizontalAlign','left');
S.tx = uicontrol('style','text',...
                'units','pix',...
                'position',[5 45 105 20],...
                'string','Password:',...
```

```
                'fontweight','bold',...
                'horizontalalign','left',...
                'fontsize',11);
S.ed = uicontrol('style','edit',...
                'units','pix',...
                'position',[110 45 200 20],...
                'backgroundcolor','w',...
                'tooltipstring',' Enter your password here.',...
                'HorizontalAlign','left',...
                'KeyPressFcn',{@ed_kpfcn,S});
S.li = uicontrol('style','pushbutton',...
                'units','pix',...
                'position',[80 10 180 25],...
                'string','Log in',...
                'horizontalalign','left',...
                'fontsize',10,...
                'HorizontalAlign','left',...
                'callback',{@ed_call,S},...
                'KeyPressFcn',{@ed_callenter,S});

uicontrol(S.un)  % Put a blinking cursor in username box.
uiwait(S.fh,time_out)  % Suspend other execution until return is pressed.

if ishandle(S.fh)
    close(S.fh) % In case the user timed out.
end

    function [] = ed_callenter(varargin)
        [h,S] = varargin{[1,3]};  % Get calling handle and structure.
        CC = get(S.fh,'currentcharacter');  % The character user entered.
        num = int8(CC); % Change to a number

        if num == 13  % This is 'return'
            ed_call();
            return;
        end
    end



function [] = ed_call(varargin)
    UNG = get(S.un,'string');
    if strcmp(str,S.PWG) && strcmp(username,UNG)    % Correct password
        tf = true;
        close(S.fh);
        return;
    else
        tf = false;  % Incorrect password
        msg = msgbox('Incorrect username and/or password','Mistake');
    end
end

    function [] = ed_kpfcn(varargin)
        [h,S] = varargin{[1,3]};  % Get calling handle and structure.
        CC = get(S.fh,'currentcharacter');  % The character user entered.
        num = int8(CC); % Change to a number

        if num == 13  % This is 'return'
            ed_call();
            return;
        end

        E = get(h,'string');  % the string of the edit box.
        % Any key handling other than the return key should be handled
        % in the following if else block.
        if num == 8  % Backspace pressed, update password and screen.
            set(h,'string',E(1:end-1));
            S.PWG = S.PWG(1:end-1);
        % Do nothing when delete and arrows are pressed
        elseif num == 127 || num == 28 || num == 29 || num == 30 || num == 31
        elseif ~isempty(num)
            set(h,'string',[E,'*'])  ;  % Print an asterisk in gui.
            S.PWG = [S.PWG CC];
        end
        set(h,'KeyPressFcn',{@ed_kpfcn,S})
    end
```

```
end
```

pdf2jpg.m

```
function imageName = pdf2jpg(path)

% Transforms the pdf file in fileName to a jpg.
% The new jpg file has name "imageName".

con = showinfowindow('Converting PDF, please wait.','Converting');
drawnow;
pdfName = getName(path);
try
     ghostscript(['-sDEVICE=jpeg -dNOPAUSE -dBATCH -dSAFER -r600x600 -dLastPage=1 "-
sOutputFile=tempim' pdfName '.jpg" ' '"' path '"']);

    drawnow;
    imageName = ['tempim' pdfName '.jpg'];
    close(con);
catch
    if ishandle(con)
        close(con);
    end
    showinfowindow('Unable to convert PDF images.','Error 401');
    rethrow(lasterror);
end
```

PictureTransmitter.m

```
function PictureTransmitter()

% PictureTransmitter opens the login screen and opens the GUI if the
% GUI if the password and username are correct. PictureTransmitter then sets
% the position of the GUI to the middle of the screen, before it makes the
% GUI visible.

        global username;
        screensize=get(0,'screensize');
        winwidth=320;
        winheight=100;
        screenwidth=screensize(3);
        screenheight=screensize(4);
        winpos=[0.5*(screenwidth-winwidth),...
           0.5*(screenheight-winheight),winwidth,winheight];
        pause(0.1);
        drawnow;
        try
            [log UNG] = Login('0000','serv1',300,winpos(1),winpos(2));
        catch
            close all;
            return
        end

        if (log == 1)
            username = UNG;
            G = GUI_BEP('Units','pixels','Visible','off');
            screensize=get(0,'screensize');
            winsize=get(G,'Position');
            winwidth=winsize(3);
            winheight=winsize(4);
            screenwidth=screensize(3);
            screenheight=screensize(4);
            winpos=[0.5*(screenwidth-winwidth),...
               0.5*(screenheight-winheight),winwidth,winheight];
            set(G,'Position',winpos,'Visible','on');
            pause(0.1);
        else
            close all;
            return
```

```
        end
end
```

resize.m

```
function [New_Image] = resize(fileName, portrait, border, Lengte, Breedte)

% Function will resize the image specified by fileName to the size Lengte
% x Breedte.
%
% fileName: The name of the file to be resized including it's extension.
%           If the file is not in the Current Directory, the path should
%           be specified here as well. For example:
%           D:\TU\Afbeeldingen\MacbethValuessRGB.jpg
% portrait: 1 if picture has to be displayed as portrait, 0 for Landscape.
% border:   Can be omitted if Lengte en Breedta are omitted. Valid values
%           are 0 and 1.
%           For 1, the image is fitted into a Lengte x Breedte matrix. For
%           0, the original
%           proportions are preserved and black borders are added to the
%           image to make
%           it fit in the requested matrix.
% Lengte:   Defines the length of the requested image.
% Breedte:  Defines the width of the requested image.


if nargin < 5
    Breedte = 480;
end

if nargin < 4
    Lengte = 272;
end

if nargin < 3   %If border isn't specified, fit the image to 272 x 480 without borders
    border = 0;
end

if nargin < 2   % If Portrait is'nt specified, handle as Landscape picture
    portrait = 0;
end

%% Read image into a uint8 matrix, turn if Portrait
warning off all;
try

 %Image = uint8(imread(fileName));
 Image = fileName;

 if portrait == 1
     Image = turnImage(Image,1);
 end

%% Het resizen naar 272 x 480 zonder zwarte banden
 if border == 0
     New_Image = imresize(Image, [Lengte Breedte]);
    drawnow;

%% Het resizen naar 272 x 480 met zwarte banden

 elseif border == 1
    [imL imB imD] = size(Image);
    props = imL/imB;                    % Proportions of input image
    spec = Lengte/Breedte;              % Proportions of the wanted image

    if props > spec  % = Borders on the side

        % Create a black matrix to fit the image in
        New_Image = zeros(Lengte,Breedte,imD, 'uint8');

        % Calculate the new width of the image without borders
        nieuweBreedte = ceil(Lengte*imB / imL);
```

```
        % Create the new image without borders
        tempImage = imresize(Image, [Lengte nieuweBreedte]);

        % Create borders on both sides
        startb = ceil((Breedte - nieuweBreedte)/2);
        if startb == 0
            startb = 1;
        end
        New_Image(1:Lengte,startb:(nieuweBreedte+startb-1),:) = tempImage;
        drawnow;

    elseif props < spec   % = Borders on the top and bottom

        % Create a black matrix to fit the image in
        New_Image = zeros(Lengte,Breedte,imD,'uint8');

        % Calculate the new width of the image without borders
        nieuweLengte = ceil(Breedte*imL / imB);

        % Create the new image without borders
        tempImage = imresize(Image, [nieuweLengte Breedte]);

        % Create borders on both sides
        startl = ceil((Lengte - nieuweLengte)/2);
        if startl == 0
            startl = 1;
        end
        New_Image(startl:(nieuweLengte+startl-1),1:Breedte,:) = tempImage;
        drawnow;
    else
        % If proportions are already good,just resize.
        New_Image = imresize(Image, [Lengte Breedte]);
        drawnow;
    end



%% Een error geven indien border verkeerd gespecificeerd is
 else
    error('Invalid input character for ''border'' field');
 end

     warning on all;

catch
    warning on all;
    rethrow(lasterror);
end
```

sendData.m

```
function foundCom = sendData(New_Image, Rate, GUIHandle, username, Com)

 % New_Image: De afbeelding(en) die moet worden verzonden. Eerst imread
 %            gebruiken.
 % Rate:      De Baudrate waarmee de bits door de USB poort worden
 %            verzonden
 % GUIHandle  De Handle van de GUI waaruit deze functie aangeroepen wordt.
 % Username   Username, nodig om de juiste receiver te adresseren
 % Com        COM poort waar de data heen wordt geschreven.
 % foundCom   Als de COM poort gevonden is, wordt deze opgeslagen.

if nargin < 5
    Com = 'not';
end

if nargin < 4
    username = 'serv1';
end

if ~ishandle(GUIHandle)
    return
```

```
end

msg1 = 1;

if instrfind ~= 0
    fclose(instrfind);
end

%% Poorten zoeken
if ~contains(Com,'COM')
    [Com message] = handshake(Rate,0.1,GUIHandle);
    if  isempty(Com)
        foundCom = [];
         return
    end
else
    msg1 = showinfowindow('Connecting to transmitter.','Preparing');
    s = Serial(Com,'Baudrate',Rate);
    fopen(s)
    try
        pause(2)
        fwrite(s,uint8(122))
        %close(msg1);
        drawnow;
        back = fread(s,1);
        fclose(s)
    catch
        fclose(s);
        rethrow(lasterror);
    end
    if 85 == back
    else
            if ishandle(msg1)
                close(msg1);
            end
            [Com message] = handshake(Rate,0.1,GUIHandle);
        if  isempty(Com)
            foundCom = [];
             return
        end
    end
end
foundCom = Com;
drawnow;
% Com = 'COM8';

%% R,G en B waardes bepalen van Image
if ishandle(msg1)
    close(msg1);
end
msg1 = showinfowindow('Transmitter found, preparing transmission. Close this window to
abort.','Preparing');
drawnow;
if ~ishandle(GUIHandle)
    return
end

% imL = 272, imB = 480, imD = 1 voor zwart/wit en imD = 3 voor kleur
[imL imB imD aantal] = size(New_Image);

    if imL ~= 272 || imB ~= 480
        warning('Image Dimensions aren''t 272x480');
    end


    red = ones(imL,imB,1,aantal);
    green = ones(imL,imB,1,aantal);
    blue = ones(imL,imB,1,aantal);

for dias = 1:aantal

    if imD == 1   % Zwart/Wit
        red(:,:,:,dias) = uint8(New_Image(:,:,1,dias));
        green(:,:,:,dias) = uint8(New_Image(:,:,1,dias));
        blue(:,:,:,dias) = uint8(New_Image(:,:,1,dias));
    elseif imD == 3  % Kleur
```

```
        red(:,:,:,dias) = uint8(New_Image(:,:,1,dias));
        green(:,:,:,dias) = uint8(New_Image(:,:,2,dias));
        blue(:,:,:,dias) = uint8(New_Image(:,:,3,dias));
    else
        error('RGB values can''t be determined');
    end;
    drawnow;
    if ~ishandle(GUIHandle) || ~ishandle(msg1)
        return
    end
end



%% Poorten openen

    % Als er poorten open zijn, worden deze nu gesloten.
    if instrfind ~= 0
        fclose(instrfind);
    end

    % Virtuele Compoort uitgang aanmaken
    s1 = serial(Com, 'Baudrate', Rate,'outputBufferSize',5000);
    drawnow;
    % Poort openen
    fopen(s1);
    try
    drawnow;
    if ~ishandle(GUIHandle) || ~ishandle(msg1)
        fclose(s1);
        return
    end


        % Wachten i.v.m. microcontroller
        for i = 1:5
            pause(0.5);
            if ~ishandle(GUIHandle) || ~ishandle(msg1)
                fclose(s1);
                return
            end
        end
        %username = uint8('serv1');
        fwrite(s1,username);        % Set Receiver adsress
        if ishandle(msg1)
            close(msg1);
        end
        drawnow;

%% Waitbar openen
        defaultBackground = get(0,'defaultUicontrolBackgroundColor');
        sendMsg = 'Sending in progress. Click ''Cancel'' or close this window to stop
transmitting.';

        msg2 = waitbar(0,sendMsg,'Name','Sending','color',defaultBackground,
'CreateCancelBtn','setappdata(gcbf,''canceling'',1);');

        try
            setappdata(msg2,'canceling',0);
            abort = 0;
            drawnow;
            if ~ishandle(GUIHandle)
                fclose(s1);
                return
            end
%% RGB naar poort schrijven

            for dias = 1:aantal

            Adres = 0;
            Packet = zeros(1,4000);
            Teller = 0;
            %Vector = []; %For testing purposes only

            while (strcmp(s1.TransferStatus,'idle') == 0)
                drawnow;
```

```
                if getappdata(msg2,'canceling')
                    abort = 1;
                    drawnow;
                    break
                end
            end

            if abort == 1
                return
            end

            for i = 1:imL
                for j = 1:imB
                        if Teller ~= 1000

                            Teller = Teller + 1;
                            beg = (Teller * 4)-3;
                            Packet(beg) = Adres;
                            Packet(beg+1) = red(i,j,1,dias);
                            Packet(beg+2) = green(i,j,1,dias);
                            Packet(beg+3) = blue(i,j,1,dias);

                        else

                            Teller = 1;
                            %Vector = [Vector Packet]; % For testing only

                            while (strcmp(s1.TransferStatus,'idle') == 0)
                                drawnow;
                                 if getappdata(msg2,'canceling')
                                     abort = 1;
                                     drawnow;
                                     break
                                 end
                            end

                            fwrite(s1,[Packet]);

                            % Update Waitbar
                            msg2 = waitbar((((dias-1)*imL*imB)+(((i-1)*imB) +
j))/(imL*imB*aantal));
                            drawnow
                            if ~ishandle(GUIHandle)
                                abort = 1;
                                break
                            end
                            %Vector = [Vector fread(s1,80)]; % For testing only
                            beg = (Teller * 4)-3;
                            Packet(beg) = Adres;
                            Packet(beg+1) = red(i,j,1,dias);
                            Packet(beg+2) = green(i,j,1,dias);
                            Packet(beg+3) = blue(i,j,1,dias);

                        end


                      if Adres > 254
                          Adres = 0;
                      else
                          Adres = Adres + 1;
                      end
                         if getappdata(msg2,'canceling')
                                 abort = 1;
                                 drawnow;
                                 break
                         end

                end
                drawnow;
                if ~ishandle(GUIHandle)
                    abort = 1;
                    break
                end
                if abort == 1
                    break
                end
```

```
            end


               fwrite(s1,Packet(1:(4*Teller)));
               %Vector = [Vector Packet] % For testing only


                   if abort == 1
                        drawnow;
                        break;
                   end
                   msg2 = waitbar(dias/aantal);
                drawnow;
                if ~ishandle(GUIHandle)
                    abort = 1;
                    break
                end
                if dias < aantal
                 pause(15);
                end

            end
            pause(0.1);
            delete(msg2);

            %Always close a waitbar!
        catch
           delete(msg2);
           abort = 1;
        end
            fclose(s1);

    catch
        fclose(s1);   % S1 moet altijd gesloten worden, ook als er errors zijn.
        if ishandle(msg1)
            close(msg1);
        end
        abort = 1;
        drawnow;
    end
    drawnow;



%% Bevestiging
            if abort == 0 && ishandle(GUIHandle)
                showinfowindow('Sending complete.','Done');
                drawnow;
            elseif ishandle(GUIHandle)
                showinfowindow('Unable to finish transmission.','Stopped');
                drawnow;
            else
                close all;
            end
```

showinfowindow.m

```
function f = showinfowindow(msg,wtitle)

%
% SHOWINFOWINDOW creates a small dialog window which displays a
% user-defined message. It is quite like MATLAB's msgbox but without an OK
% button. Although msgbox could be used for the simple display of a message
% that informs the user about a process being performed (and is not
% expected to last long, else a waitbar/timebar could be used but at a cost
% of running time), the presence of the OK button that closes the window on
% press could be annoying. SHOWINFOWINDOW resolves this problem by removing
% the button. Additionally, if you wish to write a script or create a GUI
% where you do not wish the user to accidenatlly hit the close button in the
% upper right corner of the window, you can remove the comment on line 82 of
% the code concerning the 'CloseRequestFcn' property of the dialog. However,
% if you do that you must assign a handle to showinfowindow so you
% can change its 'CloseRequestFcn' property to 'closereq' when your process
```

```
% is done or else you might end up with a non-closing window! If that
% happens, one way to resolve this could be to use the findobj function:
%
% non_closing_window_handle = findobj('CloseRequestFcn','');
% set(non_closing_window_handle,'CloseRequestFcn','closereq')
% close(non_closing_window_handle)
%
% Syntax:
%
% showinfowindow
% h = showinfowindow
% showinfowindow(msg)
% h = showinfowindow(msg)
% showinfowindow(msg,wtitle)
% h = showinfowindow(msg,wtitle)
%
% showinfowindow and h = showinfowindow without any input arguments create
% an example.
%
% showinfowindow(msg) and h = showinfowindow(msg) display the user-defined
% message msg
% which should be a character string or a cell array of strings.
%
% showinfowindow(msg,wtitle) and h = showinfowindow(msg,wtitle) display the
% user-defined message msg which should be a character string or a cell
% array of strings. This time the window has the title wtitle
%
% Examples
%
% h1 = showinfowindow('Running - Please wait...');
%
% h2 = showinfowindow({'This is a long message for a long process.',...
%                      'It is displayed in two lines.'},'Long message');
%
%============================================================
%
% Author        : Panagiotis Moulos (pmoulos@eie.gr)
% First created : May 7, 2007
% Last modified : June 11, 2007
%
% Copyright (c) 2007, Panagiotis Moulos
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.

%

% Check input arguments
if nargin<1
    msg='Example of showinfowindow.m';
    wtitle='Example';
elseif nargin<2
    wtitle='Info';
end
if ~iscell(msg)
    msg={msg};
```

```
end

% Set default sizes
DefFigPos=get(0,'DefaultFigurePosition');
MsgOff=7;
FigWidth=125;
FigHeight=50;
DefFigPos(3:4)=[FigWidth FigHeight];
MsgTxtWidth=FigWidth-2*MsgOff;
MsgTxtHeight=FigHeight-2*MsgOff;

% Initialize dialog window
f=dialog('Name',wtitle,'Units','points','WindowStyle','normal','Toolbar',...
         'none','DockControls','off','MenuBar','none','Resize','off',...
         'ToolBar','none','NumberTitle','off');%,'CloseRequestFcn','');

% Initialize message
msgPos=[MsgOff MsgOff MsgTxtWidth MsgTxtHeight];
msgH=uicontrol(f,'Style','text','Units','points','Position',msgPos,...
               'String',' ','Tag','MessageBox','HorizontalAlignment',...
               'left','FontSize',8);
[WrapString,NewMsgTxtPos]=textwrap(msgH,msg,75);
set(msgH,'String',WrapString)
delete(msgH);

% Fix final message positions
MsgTxtWidth=max(MsgTxtWidth,NewMsgTxtPos(3));
MsgTxtHeight=min(MsgTxtHeight,NewMsgTxtPos(4));
MsgTxtXOffset=MsgOff;
MsgTxtYOffset=MsgOff;
FigWidth=MsgTxtWidth+2*MsgOff;
FigHeight=MsgTxtYOffset+MsgTxtHeight+MsgOff;

DefFigPos(3:4)=[FigWidth FigHeight];
set(f,'Position',DefFigPos);

% Create the message
AxesHandle=axes('Parent',f,'Position',[0 0 1 1],'Visible','off');
txtPos=[MsgTxtXOffset MsgTxtYOffset 0];
text('Parent',AxesHandle,'Units','points','HorizontalAlignment','left',...
     'VerticalAlignment','bottom','Position',txtPos,'String',WrapString,...
     'FontSize',8,'Tag','MessageBox');

% Move the window to the center of the screen
set(f,'Units','pixels','Visible','off');          % Made invisible by Mark
screensize=get(0,'screensize');
winsize=get(f,'Position');
winwidth=winsize(3);
winheight=winsize(4);
screenwidth=screensize(3);
screenheight=screensize(4);
winpos=[0.5*(screenwidth-winwidth),0.5*(screenheight-winheight),...
        winwidth,winheight];
set(f,'Position',winpos,'Visible','on');            % Made visible by Mark

% Give priority to displaying this message
drawnow;
```

simplify.m

```
function [f1, f2, f3, n] = simplify(file1, file2, file3)

% Simplify checks wheter file1, file2 and file3 are empty and reorders them
% so the filenames will be in the front of the vector. n returns the number
% of non-empty filenames.

if isempty(file1)| (contains(file1, 'Browse ->>'))
    if isempty(file2)| (contains(file2, 'Browse ->>'))
        if isempty(file3)| (contains(file3, 'Browse ->>'))
            pause(0.1);
        else
            n = 1;
            file1 = file3;
```

```
        end
    else
        file1 = file2;
        if isempty(file3)| (contains(file3, 'Browse ->>'))
            n = 1;
        else
            n = 2;
            file2 = file3;
        end
    end
else
    if isempty(file2)| (contains(file2, 'Browse ->>'))
        if isempty(file3)| (contains(file3, 'Browse ->>'))
            n = 1;
        else
            n = 2;
            file2 = file3;
        end
    else
        if isempty(file3)| (contains(file3, 'Browse ->>'))
            n = 2;
        else
            n = 3;
        end
    end
end
drawnow;
f1 = file1;
f2 = file2;
f3 = file3;
```

turnImage.m

```
function New_Image = turnImage(Old_Image,factor)

% This function will turn a RGB or grayscale image Old_Image 90xfactor
% degrees. If factor is not specified, the image will be turned 90 degrees.
% Old_Image has to be given in matrix form, so it has to be read by imread.

if nargin < 2
    factor = 1;
end

 [imL imB imD] = size(Old_Image);
 if imD == 3                                    % In case of an RGB image
     Red = rot90(Old_Image(:,:,1),factor);
     Green = rot90(Old_Image(:,:,2),factor);
     Blue = rot90(Old_Image(:,:,3),factor);
     New_Image = zeros(imB,imL,imD,'uint8');
     New_Image(:,:,1) = Red;
     New_Image(:,:,2) = Green;
     New_Image(:,:,3) = Blue;
 elseif imD == 1                            % In case of a black/white image
     New_Image = rot90(Old_Image,3);
 else
     error('Image doesn''t contain RGB or grayscale values');
 end
```

A.2 The receive software

errorCorrection.m

```
function [COR] = errorCorrection(IN)

ERR = IN;
lERR = length(ERR);
COR = -1*ones(1,(lERR/4*3));
lCOR = length(COR);


j = 0;
for i = 1:4:(lERR-3)
    address = uint32(ERR(i));                       %Get the address and
    k = address + j*256;                            %rearrange the received
    if COR(k*3+1) == -1                             %bytes in a way only the
        COR(((k*3)+1):((k*3)+3)) = ERR((i+1):(i+3));      %RGB values are stored on
    end                                             %the right place of the
    if address == 255                               %picture.
        j = j+1
    end
end


for i = 1:3:lCOR-2
    if COR(i) == -1
        if i<(lCOR-2)
            step = 3;
            count = 0;
            while ((i+step)<(lCOR-2)) && (COR(i+step) == -1)    %Look for unreceived pixels.
                step = step+3;
                count = count+1;
            end
            if count == 0
                COR(i:(i+2)) = COR((i+step):(i+step+2));        %If only one pixel is missing
            else                                                %copy the pixel next to it.
                if i ~= 1
                    left = floor(count/2);
                    l = left;
                    right = count;
                else
                    left = 0;
                    right = count;
                end
                while left >= 0                                 %If there is more than one
                    COR((i+left*3):(i+left*3+2)) = COR((i-3):(i-1));    %pixel missing the
                    left = left-1;                              %pixel will be equal to the
                end                                             %nearest pixel.
                while right > l
                    COR((i+right*3):(i+right*3+2)) = COR((i+step):(i+step+2));
                    right = right-1;
                end
            end
        else
            COR(i:i+2) = COR(i-3:i-1);
        end
    end
end

COR = uint8(COR);


% Designed code for a Repetition Code with bitwise multiplicity.
%
%  if nargin < 2
%      ErrorControl = 0;
%  else
%      ErrorControl = EC;
%  end
% lERR = length(ERR);
% RX = zeros(1,3600);
% RX(1:lERR) = ERR;
```

```
%
%if ErrorControl == 0
%     for i = 1:4:(length(RX)-3)
%         v = (i-1)/4;
%         if mod(v,2) == 0
%             if RX(i) == 0
%             elseif i == 1
%                 if RX(1) ~= 0 && RX(1) ~= 255
%                     if RX(2) == 0
%                         RX(1:end-1) = RX(2:end);
%                     elseif RX(2) == 255
%                         RX(5:end) = RX(2:end-3);
%                     elseif RX(3) == 0
%                         RX(1:end-2) = RX(3:end);
%                     elseif RX(3) == 255
%                         RX(5:end) = RX(3:end-2);
%                     elseif RX(4) == 0
%                         RX(1:end-3) = RX(4:end);
%                     elseif RX(4) == 255
%                         RX(5:end) = RX(4:end-1);
%                     elseif RX(5) == 0
%                         RX(1:end-4) = RX(5:end);
%                     elseif RX(5) == 255
%                         'First pixel not received'
%                          elseif RX(6) == 0
%                          RX(1:end-5) = RX(5:end);
%                      elseif RX(5) == 255
%                          'First pixel not received'
%                          elseif RX(5) == 0
%                          RX(1:end-4) = RX(5:end);
%                      elseif RX(5) == 255
%                          'First pixel not received'
%                          elseif RX(5) == 0
%                          RX(1:end-4) = RX(5:end);
%                      elseif RX(5) == 255
%                          'First pixel not received'
%                     end
%                 end
%             else
%                 if RX(i) == 255
%                     RX(i+4:end) = RX(i:end-4);
%                 elseif RX(i -1) == 0
%                     RX(i:end) = RX(i-1:end-1);
%                 elseif RX(i-1) == 255
%                     RX(i+4:end) = RX(i-1:end-5);
%                 elseif RX(i-2) == 0
%                     RX(i:end) = RX(i-2:end-2);
%                 elseif RX(i-2) == 255
%                     RX(i+4:end) = RX(i-2:end-6);
%                 elseif RX(i-3) == 0
%                     RX(i:end) = RX(i-3:end-3);
%                 elseif RX(i-3) == 255
%                     RX(i+4:end) = RX(i-3:end-7);
%                 end
%             end
%         else
%             if RX(i) == 255
%             else
%                 if RX(i) == 0
%                     RX(i+4:end) = RX(i:end-4);
%                 elseif RX(i -1) == 255
%                     RX(i:end) = RX(i-1:end-1);
%                 elseif RX(i-1) == 0
%                     RX(i+4:end) = RX(i-1:end-5);
%                 elseif RX(i-2) == 255
%                     RX(i:end) = RX(i-2:end-2);
%                 elseif RX(i-2) == 0
%                     RX(i+4:end) = RX(i-2:end-6);
%                 elseif RX(i-3) == 255
%                     RX(i:end) = RX(i-3:end-3);
%                 elseif RX(i-3) == 0
%                     RX(i+4:end) = RX(i-3:end-7);
%                 end
%             end
%         end
%     end
```

```
% else
%      for i = 1:12:(length(RX)-11)
%          v = (i-1)/12;
%          if mod(v,2) == 0
%              if RX(i:i+3) == 0
%              else
%                  if RX(i:i+3) == 255
%                      if i>12 && RX(i-12) == 0
%                          RX(i-12:end-12) = RX(i:end);
%                      else
%                          RX(i:end-12) = RX(i+12:end);
%                      end
%                  elseif RX(i-1:i+1) == 0
%                      RX(i-1:end-1) = RX(1,i:end);
%                  elseif RX(i-1:i+1) == 255
%                      RX(i-1:end-1) = RX(1,i:end);
%                  elseif RX(i-2:i) == 0
%                      RX(i-2:end-2) = RX(1,i:end);
%                  elseif RX(i-2:i) == 255
%                      RX(i-2:end-2) = RX(1,i:end);
%                  elseif RX(i-3:i-1) == 0
%                      RX(i-3:end-3) = RX(1,i:end);
%                  elseif RX(i-3:i-1) == 255
%                      RX(i-3:end-3) = RX(1,i:end);
%                  elseif RX(i-4:i-2) == 0
%                      RX(i-4:end-4) = RX(1,i:end);
%                  elseif RX(i-4:i-2) == 255
%                      RX(i-4:end-4) = RX(1,i:end);
%                  elseif RX(i-5:i-3) == 0
%                      RX(i-5:end-5) = RX(1,i:end);
%                  elseif RX(i-5:i-3) == 255
%                      RX(i-5:end-5) = RX(1,i:end);
%                  elseif RX(i-6:i-4) == 0
%                      RX(i-6:end-6) = RX(1,i:end);
%                  elseif RX(i-6:i-4) == 255
%                      RX(i-6:end-6) = RX(1,i:end);
%                  elseif RX(i-7:i-5) == 0
%                      RX(i-7:end-7) = RX(1,i:end);
%                  elseif RX(i-7:i-5) == 255
%                      RX(i-7:end-7) = RX(1,i:end);
%                  elseif RX(i-8:i-6) == 0
%                      RX(i-8:end-8) = RX(1,i:end);
%                  elseif RX(i-8:i-6) == 255
%                      RX(i-8:end-8) = RX(1,i:end);
%                  elseif RX(i-9:i-7) == 0
%                      RX(i-9:end-9) = RX(1,i:end);
%                  elseif RX(i-9:i-7) == 255
%                      RX(i-9:end-9) = RX(1,i:end);
%                  elseif RX(i-10:i-8) == 0
%                      RX(i-10:end-10) = RX(1,i:end);
%                  elseif RX(i-10:i-8) == 255
%                      RX(i-10:end-10) = RX(1,i:end);
%                  elseif RX(i-11:i-9) == 0
%                      RX(i-11:end-11) = RX(1,i:end);
%                  elseif RX(i-11:i-9) == 255
%                      RX(i-11:end-11) = RX(1,i:end);
%                  end
%              end
%          else
%              if RX(i:i+3) == 255
%              else
%                  if RX(i:i+3) == 0
%                      if i>12 && RX(i-12) == 255
%                          RX(i-12:end-12) = RX(i:end);
%                      else
%                          RX(i:end-12) = RX(i+12:end);
%                      end
%                  elseif RX(i-1:i+1) == 255
%                      RX(i-1:end-1) = RX(1,i:end);
%                  elseif RX(i-1:i+1) == 0
%                      RX(i-1:end-1) = RX(1,i:end);
%                  elseif RX(i-2:i) == 255
%                      RX(i-2:end-2) = RX(1,i:end);
%                  elseif RX(i-2:i) == 0
%                      RX(i-2:end-2) = RX(1,i:end);
%                  elseif RX(i-3:i-1) == 255
```

```
%                     RX(i-3:end-3) = RX(1,i:end);
%                 elseif RX(i-3:i-1) == 0
%                     RX(i-3:end-3) = RX(1,i:end);
%                 elseif RX(i-4:i-2) == 255
%                     RX(i-4:end-4) = RX(1,i:end);
%                 elseif RX(i-4:i-2) == 0
%                     RX(i-4:end-4) = RX(1,i:end);
%                 elseif RX(i-5:i-3) == 255
%                     RX(i-5:end-5) = RX(1,i:end);
%                 elseif RX(i-5:i-3) == 0
%                     RX(i-5:end-5) = RX(1,i:end);
%                 elseif RX(i-6:i-4) == 255
%                     RX(i-6:end-6) = RX(1,i:end);
%                 elseif RX(i-6:i-4) == 0
%                     RX(i-6:end-6) = RX(1,i:end);
%                 elseif RX(i-7:i-5) == 255
%                     RX(i-7:end-7) = RX(1,i:end);
%                 elseif RX(i-7:i-5) == 0
%                     RX(i-7:end-7) = RX(1,i:end);
%                 elseif RX(i-8:i-6) == 255
%                     RX(i-8:end-8) = RX(1,i:end);
%                 elseif RX(i-8:i-6) == 0
%                     RX(i-8:end-8) = RX(1,i:end);
%                 elseif RX(i-9:i-7) == 255
%                     RX(i-9:end-9) = RX(1,i:end);
%                 elseif RX(i-9:i-7) == 0
%                     RX(i-9:end-9) = RX(1,i:end);
%                 elseif RX(i-10:i-8) == 255
%                     RX(i-10:end-10) = RX(1,i:end);
%                 elseif RX(i-10:i-8) == 0
%                     RX(i-10:end-10) = RX(1,i:end);
%                 elseif RX(i-11:i-9) == 255
%                     RX(i-11:end-11) = RX(1,i:end);
%                 elseif RX(i-11:i-9) == 0
%                     RX(i-11:end-11) = RX(1,i:end);
%                 end
%             end
%         end
%     end
% end
%
% COR = uint8(RX);
```

getRGB.m

```
function getRGB(COL,IN)

a = 1;
c = 1;
d = 1;
for k = 1:3
    for i = k:3:(length(IN)-3+k)
        if k == 1
            r(a) = IN(i);                   %Get the Red values and store in an array.
            a = a+1;
        elseif k == 2
            g(c) = IN(i);                   %Get the Green values and store in an array.
            c = c+1;
        else
            b(d) = IN(i);                   %Get the Blue values and store in an array.
            d = d+1;
        end
    end
end
j = 1;
for i = 1:COL:length(r)-COL+1
    R(j,:) = r(i:i+COL-1);                  %Combine the retrieved R,G and B values into
    G(j,:) = g(i:i+COL-1);                  %a three dimensional RGB matrix.
    B(j,:) = b(i:i+COL-1);
    j = j+1;
end
```

```
imshow(RGB);
```

Receive.m

```
function [Received NumberGood PercentageGood NumberL4Wrong PercentageL4Wrong NumberH4Wrong
PercentageH4Wrong NumberReceived] = receive(COM, Baudrate, Number, RGBImage, Show)


%% Enable serial connection

clc;
s = serial(COM,'Baudrate',Baudrate,'inputBufferSize', 600000);  %Create a serial port with  a
                                                                %buffersize of 600000.
fopen(s);                                                   %Open the serial port.
if nargin<4                                                 %The Show bit is optional, when left out
    show = 0;                                               %the image will not be shown.
else
    show = Show;
end
pause(1);

%% Receiving data

 [Row Col Dep] = size(RGBImage);                            %The dimensions of the original Image.

b = 0;
    RX = -1*ones(1,522240);                                 %The number of bytes that will be
received.
    j = 1;
    k = 0;
    NumberReceived = -1;
    n = 0;
    while n < 522240                                        %As long as there are less bytes received
        n = s.BytesAvailable;                               %as expected, wait for new data.
        if (n == k) && (n ~=0)                              %If the number of available bytes hasn't
            if b < 200                                      %changed for at most 200 seconds the
            b = b+1                                         %program will stop checking for new
            pause(1);                                       %available data and continues with
            else                                            %the rest of the program.
                'Error: Less bytes received'
                break;
            end

        end
        k = n;
%         elseif n > 0
%              r = fread(s,n);
%              r = r';
%              %length(r)
%              %length(RX(i:n+i-1))
%              RX(i:n+i-1) = r;
%              i = i+n;
%              r = [];
%               if RX(1) == -1
%                   rx = fread(s,1);
%                   if rx(1) > 15
%                       while(true)
%                         n = s.BytesAvailable;
%                         if n > 2
%                             fread(s,3);
%                             break;
%                         else
%                             pause(0);
%                         end
%                       end
%                   else
%                       RX(j) = rx(1);
%                       j = j+1;
%                   end
%               else
%                   RX(j) = fread(s,1);
%                   j = j+1
%               end
%        end
```

```
    end
    NumberReceived = j;
    RX = uint8(fread(s,n));                         %Store the available bytes in
    lRX = length(RX);                               %RX.
    if lRX < 522240                                 %If the available bytes is less
        RX(lRX+1:522240) = 0;                       %than expected, RX is extended to
    end                                             %the right size with zeros.

% else
%     RX = -1*ones(1,2088960);
%     i = 1;
%     while length(RX)<2088960
%         n = s.BytesAvailable;
%         if n > 0
%             r = fread(s,n);
%             RX(i:n) = r;
%             i = n;
%            % RX = [RX fread(s,1)];
%         elseif n == 0 && RX(1) ~= -1
%            if a == 0
%            t = timer('ExecutionMode','fixedRate','Period',1,'TimerFcn','a = a+1');
%            elseif a >= 3;
%                stop(t);
%                delete(t);
%                'Error: Less bytes received'
%                break;
%            end
%         end
%     end
%
%     RX = uint32(RX);
%     lRX = length(RX);
%     for i = 1:lRX/4
%         for j = 1:4:(lRX-2)
%             RX3(i) = RX(j)*65536 + RX(j+1)*256 + RX(j+2) + RX(j+3);
%         end
%     end
%     lRX3 = length(RX3);
%     if lRX3 ~= lRGB*4
%         error('Number of received bytes are not equal to the expected number: /nReceived
number of bytes = ' + lRX4 + '/nExpected number of bytes = ' + lRGB);
%     end
% end

fclose(s);                                          %Close the serial port.

% R = uint8(ones(1,100));
% G = uint8(ones(1,100));
% G(1:2:99) = 0;
% B = uint8(ones(1,100));
% B(2:2:100) = 0;
%
% RGB(:,:,1) = R;
% RGB(:,:,2) = G;
% RGB(:,:,3) = B;

% RGBSend(1,1:3:(300-2)) = R;
% RGBSend(1,2:3:(300-1)) = G;
% RGBSend(1,3:3:(300)) = B;
%
%
% RGBSErrControl = [];
% for i = 1:300
%     [byte1 byte2 byte3] = get3bytes32(RGBSend(i));
%     RGBSErrControl = [RGBSErrControl byte3 byte2 byte1];
% end
%
% if ErrorControl == 0
% RX = RGBSend;
% lRX = length(RGBSend);
% else
%     RX = RGBSErrControl;
%     lRX = length(RGBSErrControl);
%     for i = 1:3:(lRX-2)
%         n = ceil(i/3);
%         RX3(n) = RX(i)*65536 + RX(i+1)*256 + RX(i+2);
```

```
%     end
%     lRX3 = length(RX3);
% end

% RX = RECEIVED;
% lRX = length(RX);
%
% R = uint8(zeros(1,length(lRX/4*3)));
% j = 1;


try
    'Checking for errors...'
    R = errorCorrection(RX,Number);                 %Check for any errors and correct
    'Successfully checked'                          %if needed, also deletes the addresses.
catch
    Received = RX;
    NumberGood = -1;
    PercentageGood = -1;
    NumberL4Wrong = -1;
    PercentageL4Wrong = -1;
    NumberH4Wrong = -1;
    PercentageH4Wrong = -1;
    msgbox('errors in try/catch');
    return
end


% if CorrectError == 1;
%     'Correcting the errors'
%     COR = errorCorrection(RX);
%     for i = 1:lRX
%     v = mod(i-1,4);
%         if v ~= 0
%             R(j) = COR(i);
%             j = j + 1;
%         end
%     end
% else
%     for i = 1:lRX
%         v = mod(i-1,4);
%         if v ~= 0
%             R(j) = RX(i);
%             j = j + 1;
%         end
%     end
% end

% else
%     RX = errorCorrection(RX,1);
% end

lR = length(R)

%% Check the received bytes
j = 1;
RGB = [];
for i = 1:Col:((Row*Col)-Col+1)
    RGB(1,i:(i+Col-1),1) = RGBImage(j,:,1);         %Reorganise the RGB values
    RGB(1,i:(i+Col-1),2) = RGBImage(j,:,2);         %into a single row array.
    RGB(1,i:(i+Col-1),3) = RGBImage(j,:,3);
    j = j+1;
end

good = zeros(1,lR);
wrong = zeros(1,lR);
for k = 1:3
    for i = k:3:lR-(3-k)
        n = ceil(i/3);
        if R(i) == RGB(1,n,k)
            good(i) = good(i) + 8;
        else
            for j = 1:8
                NEControl = bitget(R(i),j);
                count = NEControl + bitget(RGB(1,n,k),j);
                if (count ~= 1)
```

```
                    good(i) = good(i) + 1;          %Counts the number of well
                else                                %received bits.
                    if j < 5
                        wrong(i) = wrong(i) + 1;     %Counts the number of wrongly
                    end                             %received bits.
                end
            end
        end
    end
end



% else
%     result = uint8(zeros(1,lRX3));
%     good = zeros(1,lRX3);
%     wrong = zeros(1,lRX3);
%     for i = 1:lRX3
%         for j = 1:3:22
%             EControl1 = bitget(RX3(i),j);
%             EControl2 = bitget(RX3(i),j+1);
%             EControl3 = bitget(RX3(i),j+2);
%             n = ceil(j/3);
%             v = round((EControl1 + EControl2 + EControl3)/3);
%             result(i) = bitset(result(i),n,v);
%         end
%     end
%     for k = 1:3
%         for i = k:3:lRX3-(3-k)
%             n = ceil(i/3);
%             if result(i) == RGB(1,n,k)
%                 good(i) = good(i) + 8;
%             else
%                 for j = 1:8
%                     count = bitget(RGB(1,n,k),j) + bitget(result(i),j);
%                     if count ~= 1
%                         good(i) = good(i) + 1;
%                     else
%                         if j < 5
%                         wrong(i) = wrong(i) + 1;
%                         end
%                     end
%                 end
%             end
%         end
%     end
% end

%% Result

Good = 0;
L4Wrong = 0;

for i = 1:lR
    Good = good(i) + Good;
    L4Wrong = wrong(i) + L4Wrong;
end
PercentageGood = Good / (lR*8) * 100;               %Calculate the percentage of
if L4Wrong == 0                                     %good received bits.
    PercentageL4Wrong = 0;
else
    PercentageL4Wrong = L4Wrong / (lR*8 - Good) * 100;    %Calculate the percentage of
end                                                       %wrongly received bits.

NumberH4Wrong = (lR*8) - Good - L4Wrong;            %Number and percentage of
PercentageH4Wrong = NumberH4Wrong / (lR*8) * 100;   %errors in the higer four bits.
NumberL4Wrong = L4Wrong;                            %Number of errors in the lower
NumberGood = Good;                                  %four bits and the number of
                                                    %well received bits.
Received = RX;                                      %The original received data.


% else
%     for i = 1:lRX3
%         Good = good(i) + Good;
```

```
%         L4Wrong = wrong(i) + L4Wrong;
%     end
%     PercentageGood = Good / (lRX3*8) * 100;
%     if L4Wrong == 0
%         PercentageL4Wrong = 0;
%     else
%         PercentageL4Wrong = L4Wrong / (lRX3*8 - Good) * 100;
%     end
% end


if show == 1
    getRGB(Col,R);                                  %Create a RGB matrix and show
end
```

Appendix B: *Tutorial programming an Arduino*

B.1 How to program an Arduino Board

This paragraph is about how to set up a connection with an Arduino Board in order to upload the designed programs which establish a USB to SPI conversion and vice versa.

B.1.1 What is needed to be able to program an Arduino Board?

Next to a programming cable and a power supply not much is needed to get started with an Arduino Board. Arduino has a well organised internet site where everything extensively is explained [31]. The only thing that has to be done is download the free software and you are able to upload any program into the ATMEL168 on the Arduino Board.

The downloaded program is an executable which means there is no need to install anything onto your computer. Another convenience is that the software is not limited to Windows but can also be run on Macintosh OSX and Linux operating systems.

B.1.2 The programming language of an Arduino Board

The language that is used is an implementation of Wiring which is based on the Processing Multimedia Programming Environment. Nevertheless, the programmer is not restricted to that specific language. Even though the Arduino language is not hard to learn when you're already familiar with programming languages like C/C++, VHDL or JAVA, the more experienced programmers can also add libraries written in C++ or use the AVR-C language.

B.1.3 How to upload a program to an Arduino Board

If the program is complete and it's ready to be uploaded a couple of settings have to be made in order to do so. Naturally, the programming cable should be attached to the Arduino and the computer.

Figure B.1: By clicking on the Verify button the program will be checked for any programming flaws.

First, the program has to be Verified or Compiled which can be done by clicking on the Verify button indicated in figure B.1. With this function all errors in the program will be detected. The errors will be shown in the black box at the bottom of the program and will be highlighted in the program itself. See for example the detected error in figure B.2.

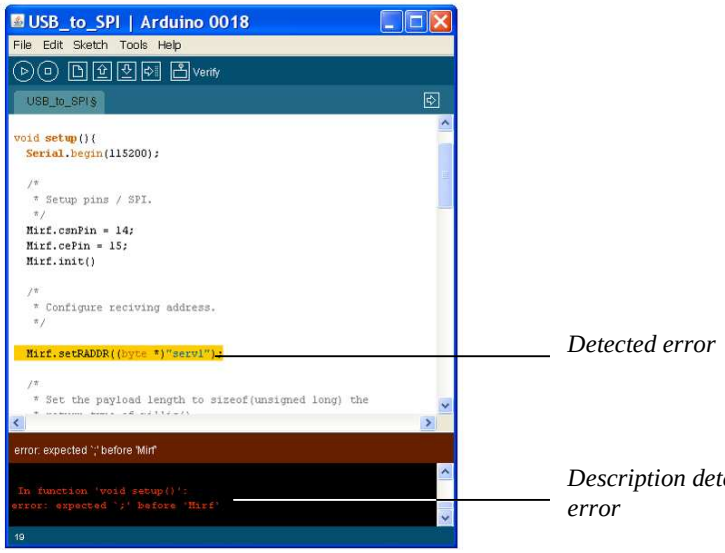


Figure B.2: The detected error is described in red at the bottom of the picture and is also highlighted within the code in yellow.

Secondly, after successfully compiling the developed program the designated type of Arduino Board has to be selected. This is done by clicking on the pull down menu *Tools* where after the appropriate board can be selected in the *Board* menu. This is shown in figure B.3.

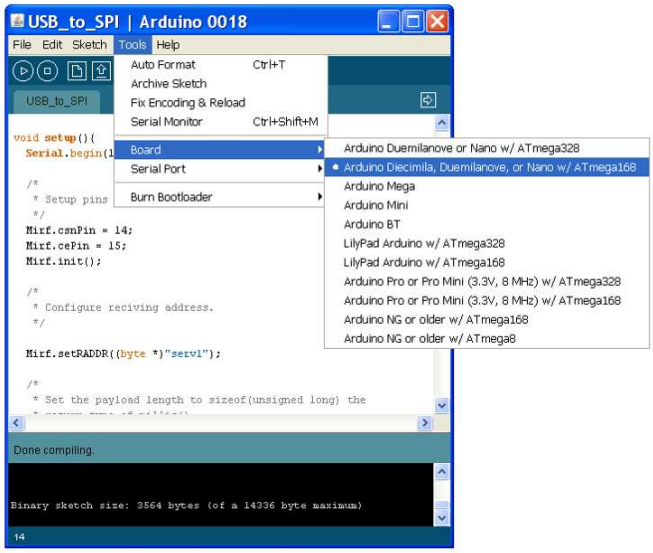


Figure B.3: Selecting the right type Arduino Board.

By checking the name on the Arduino Board itself it can be determined which type of Arduino Board has to be selected. The Arduino Board we used was a Duemilanove and is displayed in figure B.4.

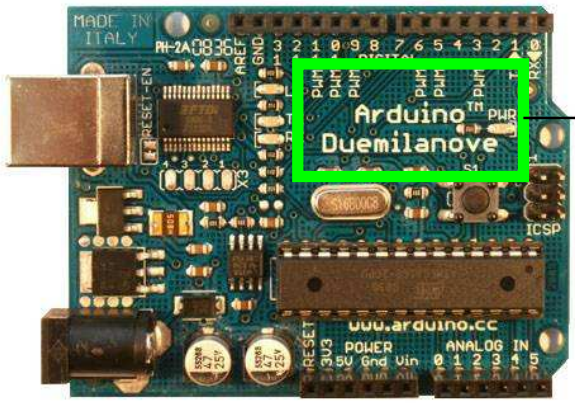


Figure B.4: The Arduino Board we used and where to check for the identification name of it. [32]

Thirdly, the COM port has to be selected on which the Arduino Board is connected. Finding out which one is used can be a bit more difficult. This is because the used COM port isn't the same on every computer. Nevertheless, go to the *Control Panel*, open the *System* menu, click on the *Device Manager* button which is located in the *Hardware* tab and the used COM port can be found under the *Ports (COM & LPT)* topic. The used port is a *USB Serial Port*. An example is shown in figure B.5.

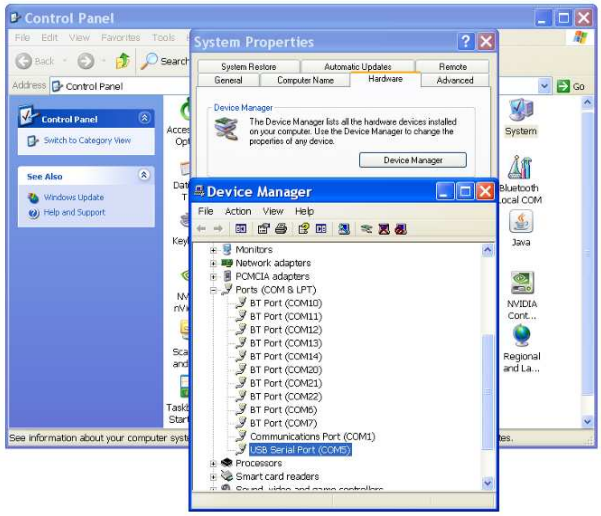


Figure B.5: Determining the COM port on which the Arduino is attached.

When the COM port is known, it can be selected in the *Serial Port* menu which is located in the pull down menu *Tools* (See figure B.6).

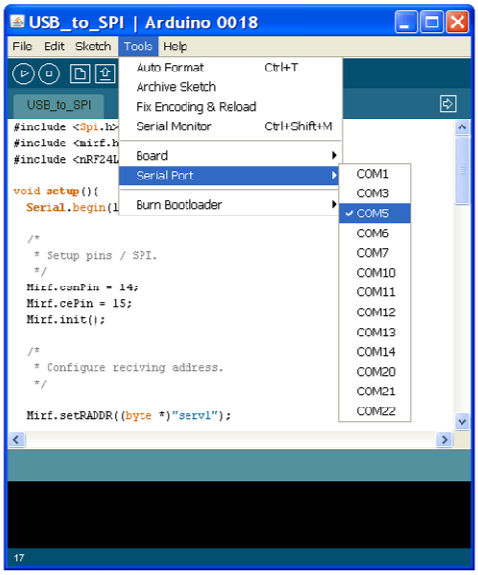


Figure B.6: Selecting the COM port on which the Arduino is attached.

Lastly, when all these settings are made and the program is successfully compiled it is ready to upload to the Arduino Board. This done by simple clicking on the *Upload* button indicated in the image below. When this button is pushed the program will be compiled once more and checked for any errors. Therefore the first compilation suggested above isn't necessary but only recommended so no errors occur during uploading.

Figure B.7: By clicking on the Upload button the program will be compiled and send to the Arduino Board.

Off course, the order in which the settings are made or compilation is done does not matter and can be done arbitrary.

B.2 Communication program 1: Computer to Transmitter

```
/**
 * Pins:
 * Hardware SPI:
 * MISO -> 12
 * MOSI -> 11
 * SCK -> 13
 *
 * Configurable:
 * CE -> 8
 * CSN -> 7
 */

#include <Spi.h>
#include <mirf.h>
#include <nRF24L01.h>

  byte handshake;
  boolean shaked;
  byte data[4];
  boolean sending = false;
  byte adr[5];

void setup(){
  Serial.begin(28800);
  Serial.flush();
  handshake = 0;
  shaked = false;

   Mirf.csnPin = 7;
   Mirf.cePin = 8;
   digitalWrite(Mirf.cePin,LOW);
   Mirf.init();                        // Initializes pins to communicate with the MiRF module

   Mirf.ceLow();
   Mirf.csnLow();
   Mirf.powerUpTx();
   Mirf.csnHi();

   Mirf.csnLow();                      // Pull down chip select
   Spi.transfer( FLUSH_TX );           // Write cmd to flush tx fifo
   Mirf.csnHi();                       // Pull up chip select

   Mirf.channel = 80;
   Mirf.payload = 4;                   // Define payload size
   Mirf.config();                      // Set payload size and channel

   Mirf.configRegister(EN_AA,0x00);    // Disable auto ack
   Mirf.configRegister(EN_RXADDR,0x01);       // Enable only 1 pipe
   Mirf.configRegister(SETUP_RETR,0x00);      // Disable retransmission
//   Mirf.setTADDR((byte *)"serv1");          // Set target address, done with serial nowadays
   Mirf.ceLow();                       // Set CE Low and let the transmitter wait for data

   Mirf.powerUpTx();                   // Set to transmitter mode and Power up

   Mirf.csnLow();                      // Pull down chip select
   Spi.transfer( FLUSH_TX );           // Write cmd to flush tx fifo
   Mirf.csnHi();                       // Pull up chip select



}

void loop(){
  if (shaked) {
      if (Serial.available() > 3) {    // If 4 bytes have been received
        data[0] = Serial.read();       // Read incoming bytes
        data[1] = Serial.read();       // Read incoming bytes
        data[2] = Serial.read();       // Read incoming bytes
        data[3] = Serial.read();       // Read incoming bytes

        for (int i=0; i <= 4; i++){    // Sent 5 times = redundancy

              while(sending){          // Wait for previous packet to be sent
```

```
                    Mirf.csnLow();
                    uint8_t Status = Spi.transfer(NOP);   // Poll status register
                    Mirf.csnHi();
               if((Status & ((1 << TX_DS)))){   // If TX_DS is high, the packet has been sent
                    sending = false;            // Set flag
                    Mirf.configRegister(STATUS,(1 << TX_DS) | (1 << MAX_RT));
                     // Clear interrupts by writing a '1'
               }
              }

         Mirf.csnLow();               // Pull down chip select
         Spi.transfer( W_TX_PAYLOAD );       // Write cmd to write payload
         Mirf.transmitSync((byte *)&data,Mirf.payload);      // Write payload
         Mirf.csnHi();                // Pull up chip select

         Mirf.ceHi();                 // Start transmission
         delayMicroseconds(5);
         Mirf.ceLow();

         sending = true;

        }
     }
  }
  else {
      if (Serial.available() > 0) {
        handshake = Serial.read();
        if (handshake != 122){
              while(Serial.available() < 4) {
              }

                 adr[0] = (byte)handshake;
                 adr[1] = (byte)Serial.read();        // Read incoming bytes
                 adr[2] = (byte)Serial.read();        // Read incoming bytes
                 adr[3] = (byte)Serial.read();        // Read incoming bytes
                 adr[4] = (byte)Serial.read();        // Read incoming bytes
                 Mirf.writeRegister(TX_ADDR,adr,mirf_ADDR_LEN);
                 shaked = true;

          }
        else {
                Serial.write(85);
                shaked = true;
              }

     }
    }
}
```

B.3 Communication program 2: Receiver to FPGA

```
/**
 * An Mirf example which copies back the data it recives.
 *
 * Pins:
 * Hardware SPI:
 * MISO -> 12
 * MOSI -> 11
 * SCK -> 13
 *
 * Configurable:
 * CE -> 8
 * CSN -> 7
 *
 */

#include <Spi.h>
#include <mirf.h>
#include <nRF24L01.h>

int LED = 8;
uint8_t adres;
```

```
byte oud[] = {-1,-1,-1,-1,-1};

void setup(){

  Serial.begin(115200);
  Serial.flush();
  pinMode(LED,OUTPUT);

  Mirf.csnPin = 14;                              // Define CSN Pin
  Mirf.cePin = 15;                               // Define CE Pin
  Mirf.init();                                   // Initialise CE, CSN and SPI

  Mirf.setRADDR((byte *)"serv1");                // Set address to receive bytes

  Mirf.channel = 80;
  Mirf.payload = 4;    // Define Payload size
  Mirf.config();                                 // Set Payload size and channel

  Mirf.configRegister(EN_AA,0x00);               // Disable auto ack
  Mirf.configRegister(EN_RXADDR,0x02);           // Enable only one pipe
  Mirf.configRegister(SETUP_RETR,0x00);          // Disable retransmission


  adres = 0;                                     // Set first address bit to '0'
}

void loop(){
  digitalWrite(LED,LOW);
  byte data[4];                                  // Buffer to store data
  if(Mirf.dataReady()){                          // Wait until data is ready in the receiver

    do{

     Mirf.getData(data);                         // Get the data

    if ((data[0] != oud[0]) && (data[0] != oud[1]) && (data[0] != oud[2]) && (data[0] !=
oud[3]) && (data[0] != oud[4])){                 // Discard redundant packets
      Serial.print(data[0]);                     // Send the data to the UART
      //Serial.print(" ");
      Serial.print(data[1]);                     // Send the data to the UART
      //Serial.print(" ");
      Serial.print(data[2]);                     // Send the data to the UART
      //Serial.print(" ");
      Serial.print(data[3]);                     // Send the data to the UART
      //Serial.print(" ");

      oud[4] = oud[3];
      oud[3] = oud[2];
      oud[2] = oud[1];
      oud[1] = oud[0];
      oud[0] = data[0];


     }
    }while(!Mirf.rxFifoEmpty());           // Continue reading data until the FIFO is empty
  }
}
```

Appendix C: *Choosing a port*

A computer uses ports to communicate with external devices. The transmission software will have to write data to the port that is connected to the transmitter module. The transmission software will have to be written in a programming language, but not every programming language can write data easily to every port. Before choosing a programming language and start programming the transmission software, we will have to decide which port to use to connect to the transmitter module. The most common ports on modern computers are the serial port, the parallel port and the Universal Serial Bus (USB), which all have their own properties.

The first paragraph of this chapter will determine which properties are required for this research. The second paragraph will discuss the parallel port and compare it to the requirements. The third paragraph will cover the serial port and the USB will be discussed in paragraph four. A fourth option is a compromise between USB and serial ports. This involves a Virtual COM Port (VCP) and is discussed in paragraph five. Finally, paragraph six will conclude which port is best used in this research.

C.1 Requirements

The employees of the advertisement companies will have to be able to connect the transmitter module to their computers. This means the port we will choose needs to be present on their computer. It is therefore of the utmost importance that our module communicates with the computer through a port that is present on every computer (Criterion 1).

Furthermore, employees have their computers running all day and sometimes even all night. Therefore it should be possible to connect the module to a running computer, without needing to restart it. This is called *hot-pluggable* (Criterion 2).

Our final requirement comes from a programming point of view. It should not be too difficult to write data to the port, since complicated transmission protocols and handshake procedures are beyond the scope of this thesis (Criterion 3).

C.2 The parallel port

The parallel port was originally designed to connect printers to a computer. It has 25 pins, from which eight are used to send a byte of data to the external device (e.g. a printer). A few pins are grounds, but the other pins may be used for various tasks. The eight data pins can write eight bits (one byte) of data at the same time (parallel). The external device sets another pin high when the data is received, to let the computer know that it can send the next byte of data [33].

Fig. C.1: A parallel port with 25 pins. Source: Wikipedia [35]

Communication over a parallel port is very easy to understand. There are hardly any protocols involved when writing data over a parallel port, the computer simply sends the data and waits for the acknowledgement. With many programming languages it is easy to write data to it. Hot-plugging is not supported with parallel ports [34]. A parallel port is present on most desktop computers. However, most laptops don't have a parallel port. Using this port would be very inconvenient for users with a laptop.

C.3 The serial port

The serial port (also called COM port) was originally designed to connect modems to a computer. It has nine pins, from which one is used to transmit data from the computer to an external device and one is used for receiving data from the external device. If the computer wants to write a byte to the external device, it will send one bit of the byte at a time. This way, transmission takes eight times longer than with the parallel port, but the serial port is smaller and uses fewer cables [36].

Fig. C.2: A serial port with 9 pins. Source: Ergo Canada [37]

Writing data to a serial port is very easy and supported by almost all programming languages. Almost all desktop computers still have one or more serial ports. Even many laptops have at least one serial port. However, it is expected that the serial port will disappear completely from computers within a few years. Many new peripheral devices use USB, which will be discussed in the next paragraph. If there are no devices to connect with a serial port, it will be completely replaced by USB. On top of that, hot plugging is not supported with serial ports [34].

C.4 The Universal Serial Bus

The Universal Serial Bus was designed to replace the serial and parallel ports. USB creates a method of attaching and accessing peripheral devices that reduces overall cost, simplifies the attachment and configuration from the end-user perspective and solves several technical issues associated with old style peripherals [38].

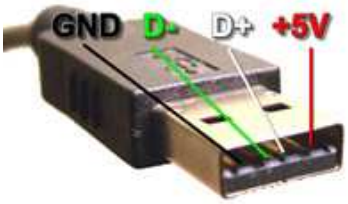


Fig. C.3: A Universal Serial Bus. Source: www.riccibitti.com [40]

The USB uses four wires, from which two are used for data transfer and two are used for power transfer. The two data wires are called D+ and D- (see figure C.3). When a logical ‘1’ is written to the port, the voltage of the D+ wire is made higher than the voltage of the D- wire. A logical ‘0’ causes the opposite.

Even though USB uses only four pins, it is faster and more reliable than the serial and parallel ports. This was achieved with the USB protocol. The protocol includes handshake mechanism, timeout rules and a low control mechanism. Each packet transmitted on the bus includes check bits and CRC protection. [39] With this protocol, a very low bit error rate is attained. Even though this protocol makes USB superior to other interfaces, it also makes it very complex and hard to understand.

Some of the key features that comprise the USB implementation are:

- The *low cost* of the port.
- *Hot pluggable:* Devices can be connected while the computer is running, without requiring user intervention.
- *127 devices* can be connected per USB using a hub.
- *High speed* communication, up to 480 MB/s.
- Peripherals can be *powered* directly from the cable.
- *Error detection* and *recovery* mechanisms.

Almost all modern computers have USB ports. Since one USB port can connect 127 devices to a computer, there is always room for another device. The devices are hot pluggable, but the USB protocol makes writing data to the USB port very complex.

C.5 USB with Virtual COM Port

USB devices are hot pluggable and can be connected to almost every computer. But due to the USB protocol, it is very difficult to write data to this device. A serial port is actually the opposite of the USB: it is very easy to write data over a serial port, but it is not hot pluggable and it is present less and less on modern computers. Combining the best properties of these two ports would be ideal.

Combining these ports is done with software, creating a Virtual COM Port (VCP). A driver needs to be installed on the computer to create a VCP. This driver makes the computer think it has a device connected to a serial port (COM port), while it is actually connected to a USB port. Data can be written to the virtual COM port. The driver will read this data and use the USB protocol to send it to the external device. Data coming from the device can be read trough this virtual COM port as if it was coming from a serial port. So the driver actually combines the best properties of USB and serial ports. All the requirements are met when using a USB with a VCP.

C.6 Comparing the ports

All information from the previous paragraphs is summarized in table C.1. This table shows that the Universal Serial Bus should be used, in combination with drivers that create a Virtual Com Port on the computer. That is the only way all the requirements are met. The following chapters discuss the development of transmission software to write data to the USB port through a VCP and the design of a transmitter module that can be connected to the USB port.

Criterions	Parallel Port	Serial Port	USB	USB with VCP
Criterion 1	yes	yes	yes	yes
Criterion 2	no	no	yes	yes
Criterion 3	no	no	no	yes

Table C.1: Comparing the ports using the criteria from §3.1

Appendix D: *Correspondence about cursor behaviour in edit text*

Dear Mr. de Jong,

I've verified this behavior with our development team.
This is an expected behavior. When modifying a string field of and Edit graphic object the cursor is placed at the beginning of the string:

```
h = uicontrol('Style','Edit');
uicontrol(h)

%Manually write
%%
set(h,'string','MATLAB')
```

This changed from R2007b and has been maintained in all following releases. Unfortunately at this time there's no way to control the cursor position and then there are no workarounds.

From your request I've opened two requests to our development team. A first request is asking to enable a direct control of the cursor position in the edit field. In this way you should be able to control the actual position of the cursor directly from code. A second request is demanding to document the current cursor behavior in product documentation. The development of new features is directly managed from our development team so I'm not able to know how much time his may require.

Please let me know if you need any clarification.

Sincerely,

Davide Ferraro
MathWorks

[THREAD ID: 1-CXRPV2]

Appendix E: *Enquiries*