2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: energie

Masterthesis

Creation of a Wireless Networked Control System using XBee antennae and LabView

PROMOTOR :
dr. ir. Johan BAETEN

PROMOTOR :
dr. ir. Isidro CALVO

COPROMOTOR :
dr. ir. Oscar BARAMBONES

Steven Abrahams

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie, afstudeerrichting automatisering

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt

2017 • 2018

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: energie

Masterthesis

Creation of a Wireless Networked Control System using XBee antennae and LabView

PROMOTOR :

dr. ir. Johan BAETEN

PROMOTOR :

dr. ir. Isidro CALVO

COPROMOTOR :

dr. ir. Oscar BARAMBONES

Steven Abrahams

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie, afstudeerrichting automatisering

UHASSELT KU LEUVEN

Preface

The work you are about to read is named *"Comparison of a Wired and Wireless Networked Control System using XBee antennae and LabView".* The research for this paper was done at UPV/EHU. This work is a part of my master's thesis in industrial engineering at UHasselt and KU Leuven in the academical year 2017-2018.

For this project I sat down many times with my external promotor, prof. dr. ir. Isidro Calvo, to discuss the goal of my research and create a possible solution. For this I would like to thank him for the support he offered regarding the project and, next to that, the help he offered to get to know the new city I was living in, namely Vitoria-Gasteiz.

Next to my external promotor I would like to thank my internal promotor and external co-promotor as well. Prof. dr. ir. Johan Baeten and prof. dr. ir. Oscar Barambones were always ready to offer any help when needed. Thanks to several feedback moments they helped me to get back on track and complete my project successfully.

Lastly, I would like to thank my friends and family. If I was stuck or needed some motivation I could always reach them for tips or some motivating words to keep working hard and eventually finishing this paper successfully.

Enjoy!

Steven Abrahams

Vitoria-Gasteiz, May 2018

Table of contents

List of tables

List of figures

Abstract

This paper focuses on wireless controlled network systems and investigates the viable solutions for wireless communication in an industrial setting. The wireless communication offers several challenges such as finding the optimal sampling period, analyzing message delay, reducing message dropout and keeping the power consumption low.

This work will not give a working solution but will draw conclusion about using XBee antennae for the wireless communication compared to wired communication, and LabVIEW for programming the control loop. Since it was impossible to physically close the loop it was chosen to test the characteristics that are important for a control loop. Several experiments were designed to compare the two modes of communication.

A first experiment pointed out that the ADC of the Arduino in combination with the XBee for transmitting was the most optimal solution. The next experiments were done with this configuration to investigate the message delay and message dropout caused by several factors.

In general, it can be concluded that for none critical systems or monitoring, the XBee provides a sufficient solution. For more critical solutions it is recommended to improve several factors of the configuration used in this work such as a different baud-rate, using a real-time operating system, controlling the environment to reduce interference, and using a better ADC.

Abstract (Nederlands)

Dit werk focust zich op draadloze netwerken voor regelsystemen om te zien of er een mogelijke oplossing bestaat voor een industriële toepassing. De draadloze communicatie levert enkele uitdagingen zoals het vinden van een optimale bemonsteringsperiode, analyseren van vertragingen, verminderen van het wegvallen van de berichten en verlagen van het energieverbruik.

Deze paper biedt geen kant en klare oplossing maar zal wel een proof of concept zijn voor het gebruik van XBee antennes voor de draadloze communicatie vergeleken met bedrade communicatie, en voor LabVIEW voor het programmeren van de regellus. Aangezien het onmogelijk was om de regellus fysiek te sluiten werd besloten om de karakteristieken die belangrijk zijn voor een regellus te testen. Hiervoor werden vergelijkende experimenten tussen beide communicaties ontworpen.

Een eerste experiment wees uit dat de ADC van de Arduino in combinatie met een XBee voor de transmissie de meest optimale opstelling was. De volgende experimenten werden gedaan met deze opstelling om de vertraging en uitval te analyseren, veroorzaakt door enkele factoren.

Uit de analyse kan geconcludeerd worden dat deze methode voor niet-kritische systemen een goed genoeg alternatief is. Voor meer kritische systemen is het aangeraden om bepaalde aanpassingen toe te passen zoals een hogere baud-rate gebruiken, een real-time operating system gebruiken, de omgeving controleren opdat er minder interferentie is, en een betere ADC gebruiken.

1. Introduction

1.1 Situation

Almost every industrial process must be monitored and controlled automatically. For this there is a need for different sensors and actuators in a network, this is called a Networked Control System or NCS. Nowadays typical control systems are interconnected with wires.

In the future the industry is expected to work more with wireless communication since this increases the flexibility of the installation and it avoids long wires that are hard to place or are an obstruction. If wireless communication is applied to a NCS's it becomes a wireless NCS or in short WNCS.

1.2 Problem statement

There are several problems with WNCS's. The first one is selecting the adequate sampling period, since before transmitting the data needs to be sampled. In [1] and [2] is spoken of a guideline for picking a sampling rate, namely: $\omega * h$ *equals [0.1 , 0.6]* where ω equals the desired frequency [Hz] of the control loop and h the sample period [s]. This guideline is based on previous experiences and simulations. Normally the sampling period should be as low as possible however for a wireless network this means there is more network traffic and this can overload the network. So it is best to choose $\omega * h$ equal to ±0.6 [2].

The next problem is message delay. The wireless communication will cause a delay between the sensor and controller and/or the controller and actuator. This introduces phase shifts limiting the control bandwidth and therefore the closed-loop stability [2].

A third problem is message dropout. This can be caused by either the control algorithm or the wireless communication itself (due to loss of connection). Obviously, these dropouts cause problems for the reliability of the system. For example, if an actuator has to stop at a certain value for the position, but it doesn't receive the command, it will not stop at the right position and cause errors [2].

The final problem is Network Energy Consumption. In order to realize the requested flexibility, a WCNS needs a non-wired power source. If the energy consumption of the sensors, actuators and/or wireless communication is too high it may cause problems to maintain a reliable application with a long lifespan [2].

1.3 Goals

Since a normal wired NCS already is a reliable method for creating a control loop it is interesting to make a comparison between a wired and a wireless NCS. This is why this study will look into the different characteristics and the effects of wired and wireless communication on these characteristics.

Several experiments will be designed and executed at different sample periods and distances to see the effect on the results of these experiments. Both a wired and a wireless connection will be tested simultaneously or separately, depending on the experiment, to see the difference in the characteristics.

The results of the study should point out if the concept of working with a WNCS is a reliable alternative or not.

1.4 Proposed solution

To try and solve these problems there is decided to work in LabVIEW. LabVIEW offers you great control over the control algorithm and the communication. You can determine the sampling period by using timers and you can guard message delays and/or dropouts and implement safety precautions in case of occurrence.

For reducing the energy consumption XBee antennae will be used. These antennae are based on the IEEE 802.15.4 protocol, which is known for its low power consumption as can be seen in Figure 1 found in source [3].

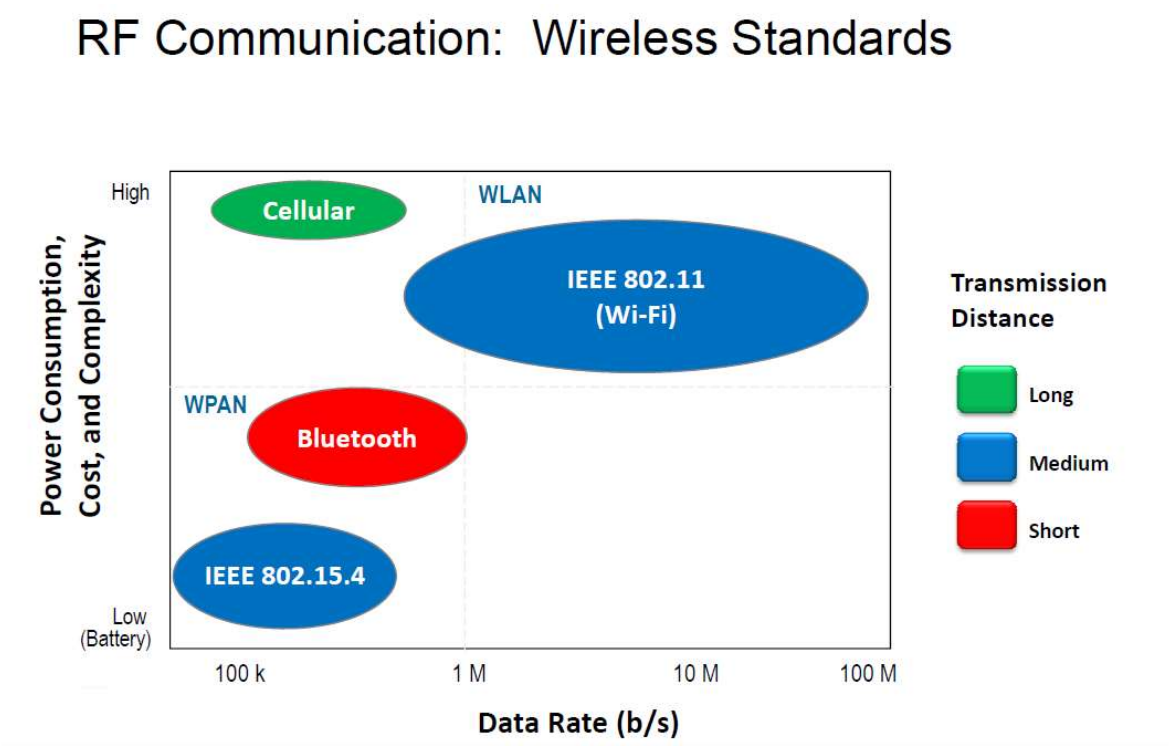


Figure 1: RF Communication [3]

For this project there will be three XBeeS1 antennae as seen in figure 2, one is connected to the computer that runs the LabVIEW program. The XBee is connected using an explorer like the one in figure 3. The next one will be attached to a shield shown in figure 4 that is connected to an Arduino which is, in his turn, connected to any kind of sensor. The last one will also be connected to an Arduino with a shield and is connected to an actuator. This is the minimum number of modules needed to simulate a control loop.

Figure 2: XBee antenna [11]

Figure 3: XBee USB explorer [9]

Figure 4: XBee shield for Arduino [10]

2. An overview

As already stated, the goal of this project is to examine if a WNCS is a viable replacement for wired NCS's. The possible solution that is proposed is tested in LabVIEW using wireless communication through XBee modules. In Figure 5 is an overview of what the control loop and its components would look like if the loop is fully closed.

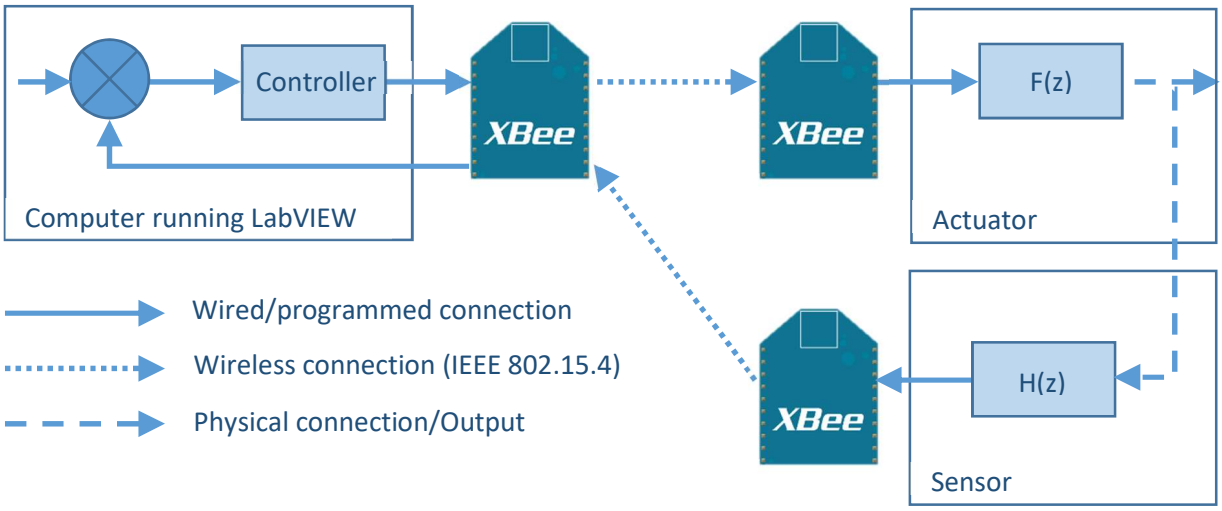


Figure 5: Overview control loop

The first part is the computer that contains the logical part of the control loop in LabVIEW (this is the comparison between the set value and the measured value plus the controller). Connected to the computer is the first of three XBees. This one sends data to the actuator and receives data from the sensor.

The second part is the actuator. It exists out of an Arduino, a motor and the second XBee. The XBee receives the data sent from the computer and is connected to the Arduino's RX and TX port. At his turn the Arduino translates the data and actuates the actuator.

The last part is the sensor. As the name already states, this part contains a sensor and the final XBee to measure the change in the state of the actuator.

Initially the goal was to create a motor control in which the actuator was a DC-motor and the sensor was an encoder. Due to some issues during the development of the project it was not possible to use an encoder but instead a potentiometer was used. With this potentiometer the physical connection with the motor could not be made but it was still possible to test several characteristics of the communication.

In the next chapters each part of the control loop and all the connections will be explained. Firstly, the wireless communication through XBee will be explained since this is the centrepiece of the project. Next chapter will dig deeper into the LabVIEW code used to communicate with the modules and the logical part of the control loop. Chapter 5 discusses the used hardware more in detail and the different measurements that were done. These measurements give several results and will be summarized and discussed at the end of the chapter. And finally, a conclusion can be made in the last chapter using these results.

3. Introduction to XBee

3.1 The XBee antennae

In this project it was opted to work with XBee antennae. They are designed to meet the IEEE 802.15.4 standard which ensures that the antenna is a low-cost and low-power solution. On top of that the data being transmitted is reliable since there is a checksum to check if the data that was send is still correct [4].

In Figure 1 a small comparison is made between several other communication protocols. The most interesting one is the comparison with WiFi. Both forms of communication are medium range and are designed after an IEEE standard. They both typically work at 2.4 GHz although both can also work at other frequencies [4], [5]. This overlap in frequency can cause interferences. Later in this paper the influence is tested.

The range of the XBee antenna goes up to 30 meters indoors or in an urban setting and even 100 meters outdoors in line of sight. The XBee-PRO is even better and goes up to 100 meters indoors and 1500 meters outdoors [4].

The low-power functionality is due to the XBee's low TX and RX currents that are respectively 45 and 50 milliamps at 3,3 volts. The XBee also has a sleep or power-down mode in which the current is lower than 10 microamps. For the XBee-PRO these currents are 215 milliamps for the TX current and 55 milliamps for the RX current. The TX current is significantly more since more power is needed to transmit the signal over a longer distance. The power-down current is the same is the normal XBee [4].

Depending on the application it must be decided which XBee fits best for the application's needs. Since this project was created to investigate several characteristics it was chosen to work with the regular XBees.

3.2 XCTU Software

By default, the antennae work as plug and play modules and don't need any adjustments. But to create a personalized network using the XBee antennae it is necessary to reprogram the firmware on the antennae. This can be done by using the XBee Configuration, Test and Utility software or simply the XCTU software.

Every XBee antenna has a set of settings that can be adjusted using XCTU or AT-commands. The AT-commands will be explained in chapter 3.3.

The first thing that must be done to change the settings is connect the XBee to the computer using the XBee USB explorer. Once the XBee is connected the XCTU software can be opened. In the top left corner of the window is an option to search for connected XBee devices.

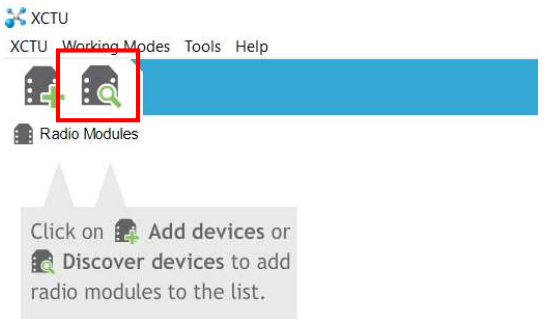


Figure 6: Search devices XCTU

The screen in figure 7 pops up. Here the correct port must be selected. If this port is not known all ports can be selected.

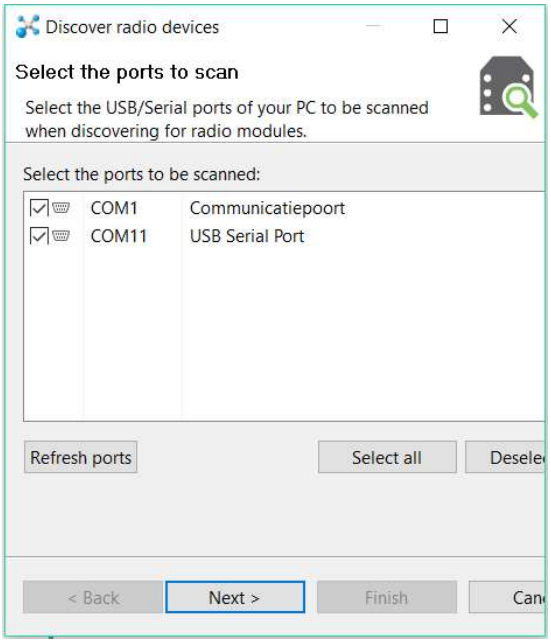


Figure 7: COM-port screen

After pressing next the screen in figure 8 appears. The settings in this screen can stay as they are. Pressing finish causes the software to start looking for the connected device.

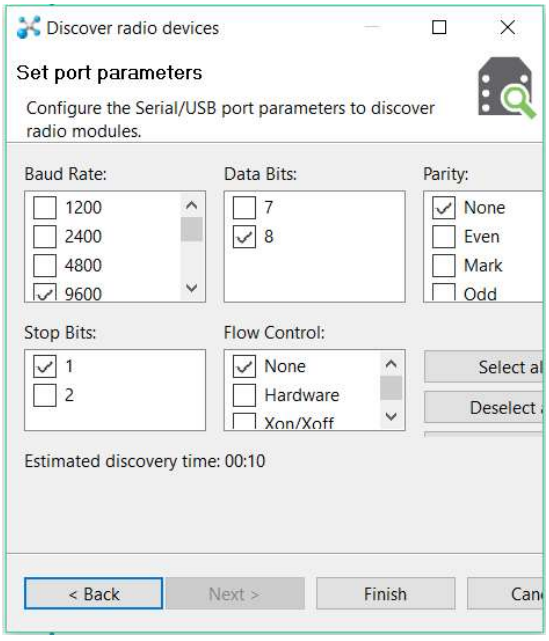


Figure 8: Settings search XCTU

While searching, a new screen like figure 9 appears that shows the devices that are found. On this screen the devices you want to add can be selected. After selecting the right devices click “add selected devices”.

Figure 9: Discovered modules

On the left side of the next screen are all the devices that are added. By clicking one of them the settings can be adjusted on the right side of the screen. Figure 10 shows what this screen looks like.

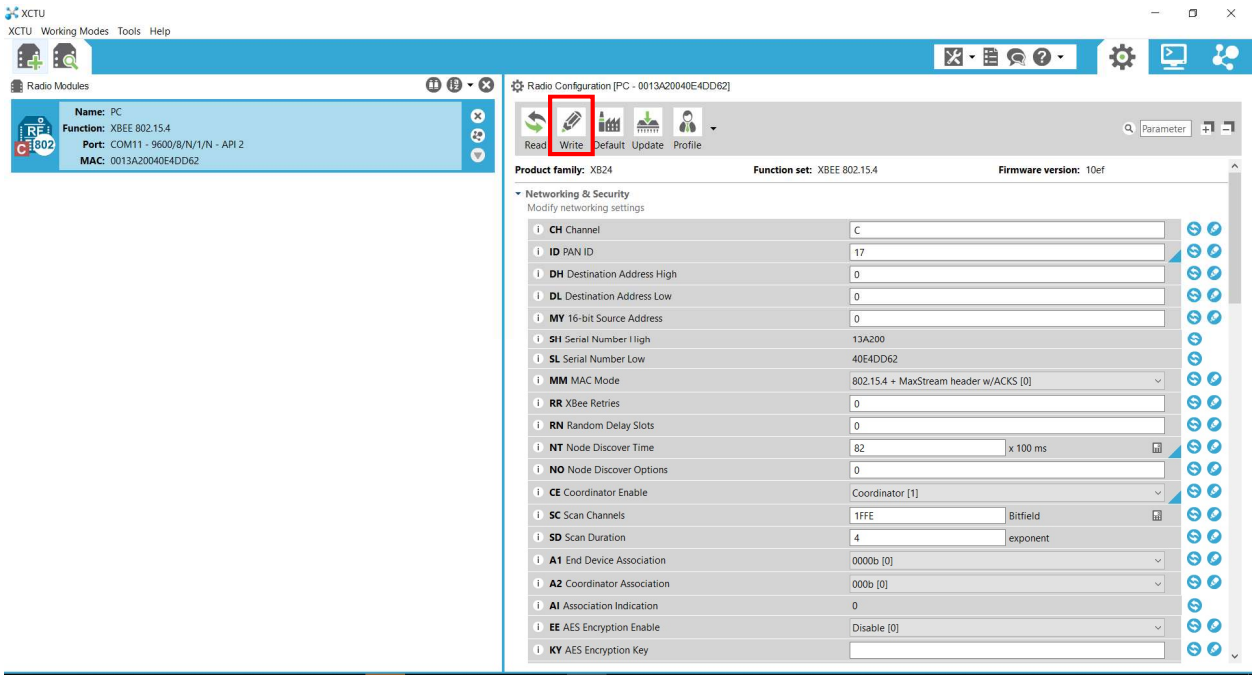


Figure 10: Settings screen XBee

Pushing the write button will write the changes made in the settings on the XBee and save them. The write button is framed in red in figure 10.

There are many different settings that can be adjusted to match the needs of the application. The functionality of each setting can be found in the XCTU software by pressing the exclamation mark corresponding to the setting. In table 1 a short summary is provided with the most important settings and its functionality as described in XCTU.

In this table it can already be noticed that the settings are also called AT-commands. This is because AT-commands can be used to change settings using other serial communication software. This will be further explained in the next chapter.

Table 1: Summary XBee settings

Setting/AT-command	Functionality
ID	Set the PAN (Personal Area Network) ID. Set ID = 0xFFFF to send message to all PANs. Every device in the same network needs to have the same PAN ID.
DH/DL	Set/read the upper/lower 32 bits of the 64 bit destination address. Set the DH register to zero and DL less than 0xFFFF to transmit using a 16 bit address. 0x000000000000FFFF is the broadcast address for the PAN.
MY	Set/read the 16-bit source address for the modem. Set MY = 0xFFFF to disable reception of packets with 16-bit addresses. 64-bit source address is the serial number and is always enabled.
CE	Set/read the coordinator setting: 0-End Device, 1-Coordinator.
NI	Set/Read Node Identifier string
AP	Enables API mode. (To enable API w/PPP select [3])
D0…8	This setting configures the different pins on the XBee antenna. All pins can be used as DI or DO. Some can be configured as ADC, CTS or RTS flow control and Associated LED indicator.
IR	Set/read sample rate. When set this parameter causes the modem to sample all enabled DIO and ADC at a specified interval.

3.3 API-mode vs. AT Commands

In the previous chapter the term AT-commands was mentioned several times. AT-commands are used to change the settings of the firmware on the XBee antennae. To use these commands the XBee must be put into AT-command mode first, this is done by sending “+++” over the serial line connected to the XBee. Once in AT-command mode the setting can be altered or requested by sending AT plus the corresponding code for the wanted alternation or information. After sending this the changes can be written to the XBee using ATWR as command. By sending ATCN the XBee will exit the AT-command mode [4]. In figure 11 is an example.

Sample Configuration: Modify RF Module Destination Address

Example: Utilize the X-CTU "Terminal" tab to change the RF module's DL (Destination Address Low) parameter and save the new address to non-volatile memory.

After establishing a serial connection between the RF module and a PC [refer to the 'Setup' section above], select the "Terminal" tab of the X-CTU Software and enter the following command lines ('CR' stands for carriage return):

Method 1 (One line per command)

Send AT Command	System Response
+++	OK <CR> (Enter into Command Mode)
ATDL <Enter>	{current value} <CR> (Read Destination Address Low)
ATDL1A0D <Enter>	OK <CR> (Modify Destination Address Low)
ATWR <Enter>	OK <CR> (Write to non-volatile memory)
ATCN <Enter>	OK <CR> (Exit Command Mode)

Method 2 (Multiple commands on one line)

Send AT Command	System Response
+++	OK <CR> (Enter into Command Mode)
ATDL <Enter>	{current value} <CR> (Read Destination Address Low)
ATDL1A0D,WR,CN <Enter>	OK<CR> OK<CR> OK<CR>

Figure 11: Example AT-commands[4]

During the duration of the project several ways of communicating between the XBees was tested but many of them were unsuccessful. All these problems caused the proceeding of the project to be delayed.

The example in figure 11 shows that it is possible to change the destination address of the XBee. This brought up one of the ideas to use this to change the communication in LabVIEW between the sensor and the actuator in the control loop. The problem here was that after sending "+++" there needs to be a certain amount of time without any messages, called the guard time. By default, this time is one second which is far too long for any crucial control loop. This guard time can be adjusted to be shorter but here there is a chance that the AT-command mode is entered unwillingly. On top of that this way of working used the LIFA-base in LabVIEW. LIFA stands for LabVIEW Interface For Arduino and is a good way to communicate between LabVIEW and an Arduino. But to start this communication an initialization needs to be done which also took more than a second. So, both the AT-command mode and the initialization of the Arduino take too long to be a reliable option for a control loop.

Since AT-command mode was not the best solution another way of addressing had to be found. This other way is the API mode. As shown in table 1 there is an option to put an XBee in API-mode with escaping characters. In this mode all the data that is communicated is fit into a frame of bytes. Figure 12 shows the structure of a typical frame. Since some characters like for example 0x7E have a certain meaning that may confuse the message if they are repeated, it is necessary to escape these characters. They can be escaped by placing 0x7D before the character that, in his turn, is XORed with 0x20.

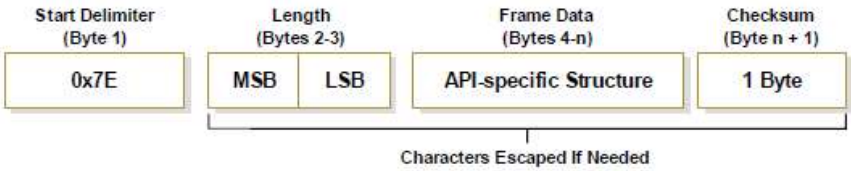


Figure 12: API-frame structure [4]

All the bytes send in a frame are hexadecimal values. The frame always starts with the frame delimiter 0x7E to indicate where the frame starts. After the delimiter follows the number of bytes in the entire frame, without the delimiter. Next is the API-specific structure. This structure always starts with an API identifier to determine what kind of message is being sent or received and is followed by the data corresponding to the message. To end the frame a checksum is calculated. The sum of all the data after the length of the frame including the checksum should equal 0xFF. If not, the message was not delivered correctly [4].

The most interesting structure that was investigated more is the structure to transmit data from one XBee to another specific XBee. Shown in figure 13, the API identifier for this structure is 0x01 followed by the Frame ID which identifies the UART data frame for the host to correlate with a subsequent acknowledgement. If this frame ID is set to 0x00 the response frame is disabled. Next is the address of the receiving XBee. This address is a 16-bit value or two bytes, a MSB and a LSB. This address can be set for every individual XBee using the AT-commands ATDH and ATDL as shown in table 1. Following the address are some possible options to add to the frame. At the end the data is added that must be transmitted, going up to 100 bytes per packet [4].

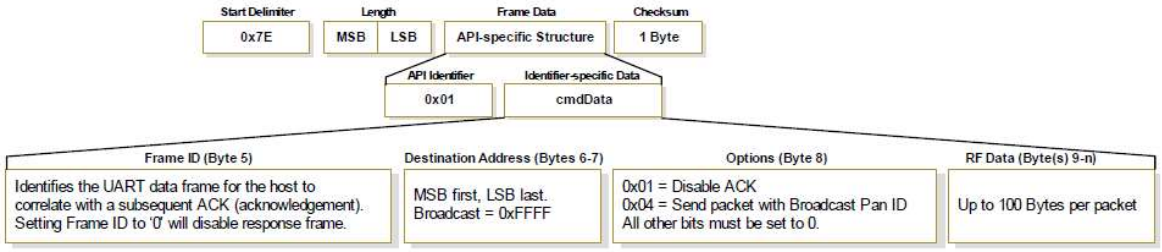


Figure 13: TX (Transmit) Request [4]

It was planned to use this structure to send data from the computer to the actuator. But using 16-bit addressing never worked. After spending several days of looking up information on how to do the addressing still no information could be found. Because this did not work it was opted to use broadcasting to transmit the data and add an address in the transmitted data. This data is then analysed by the Arduino connected to the XBee antenna and checks if the address in the data corresponds with the chosen address of the module. This will be explained in more detail in chapter 4.2 and 5.1.

4. Implementation in LabVIEW

4.1 VISA-resource

Because there are many different companies that make instrumentation there was a need for standardization to ensure interoperability between all these different devices. But even after this standardization some problems remained. This caused National Instruments and several other companies to design NI VISA to make the initialization of communication easier and faster [6].

In LabVIEW it becomes clear why NI VISA is so easy to use.

Figure 14: Typical NI VISA program [6]

Figure 14 shows a typical program in which first a communication line is opened, next something is written and read on this line and at the end the communication is closed. These are the only building blocks needed to setup communication with any instrument on any port of the computer [6].

4.2 Analysing and constructing API-frames

In the application, built in LabVIEW, API-frames are used to communicate between the XBees. To translate the data coming from the sensor a small VI (virtual instrument) was written in LabVIEW based on a YouTube-video [7]. This VI, shown in figure 15, is later used as a SubVI for the larger code.

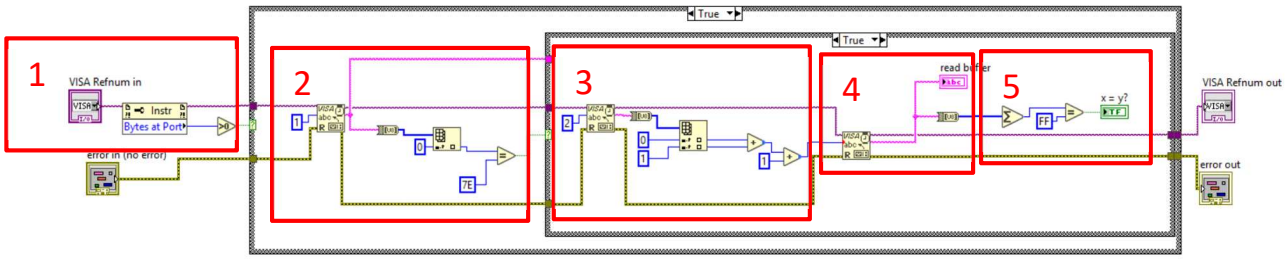


Figure 15: Analysis API-frame [XBee Receive.vi]

There are five parts in the VI. Figure 16 zooms in on the first two parts. The first part checks if there are any bytes at the port. If this is true the program enters a first case structure containing the rest of the program. The second part reads the first byte and checks if this byte matches the frame delimiter 0x7E. If this is true again, a second case structure is entered.

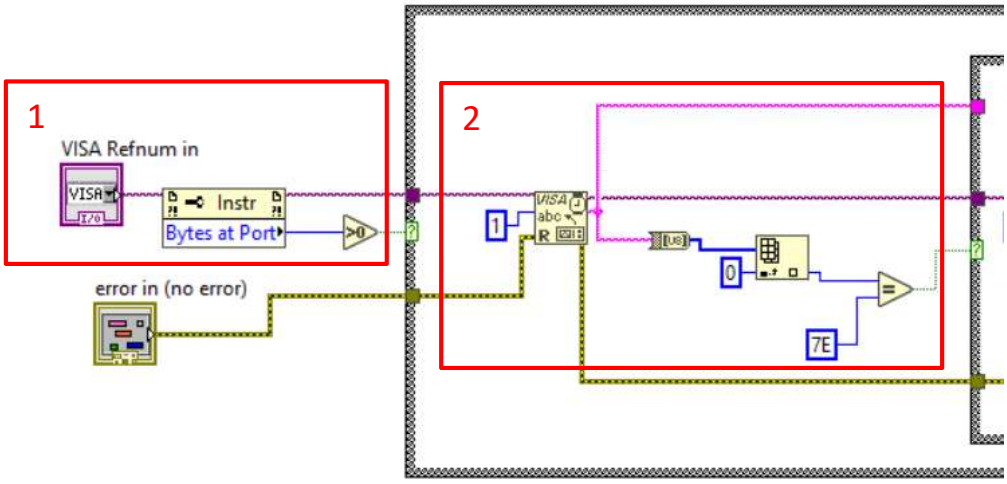


Figure 16: Analysis API-frame (1 and 2)

In this second case structure are the last three parts shown in figure 17. Part number three reads the next two bytes to calculate the length of the rest of the frame. Because the length does not include the checksum, one is added to the length. This length is then used in part four to read all the data. The last part is a calculation to see if the sum of the data with the checksum equals 0xFF.

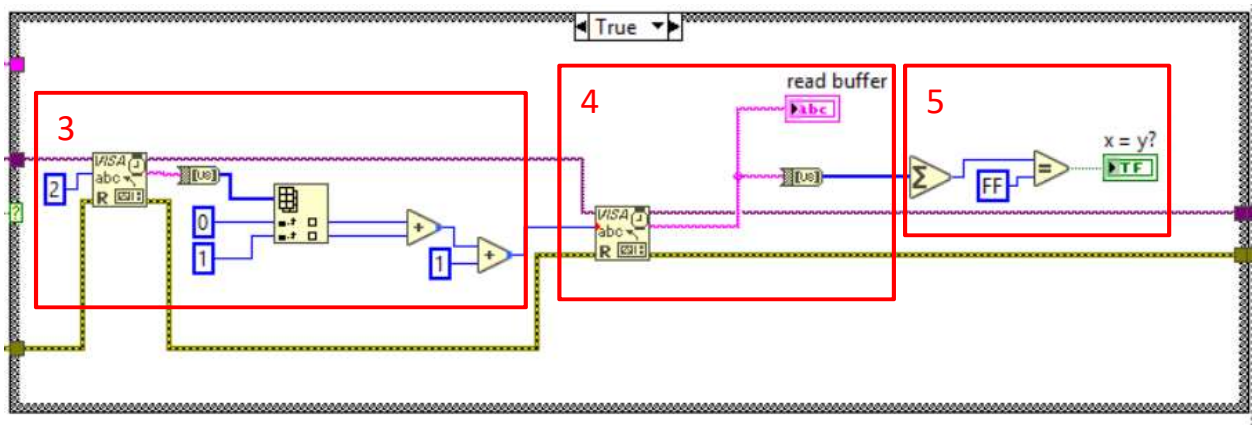


Figure 17:Analysis API-frame (3, 4 and 5)

After reading the data from the sensor, different data must be transmitted in an API- frame. For this the code in figure 18 was written. This code was also based on the YouTube-video [7].

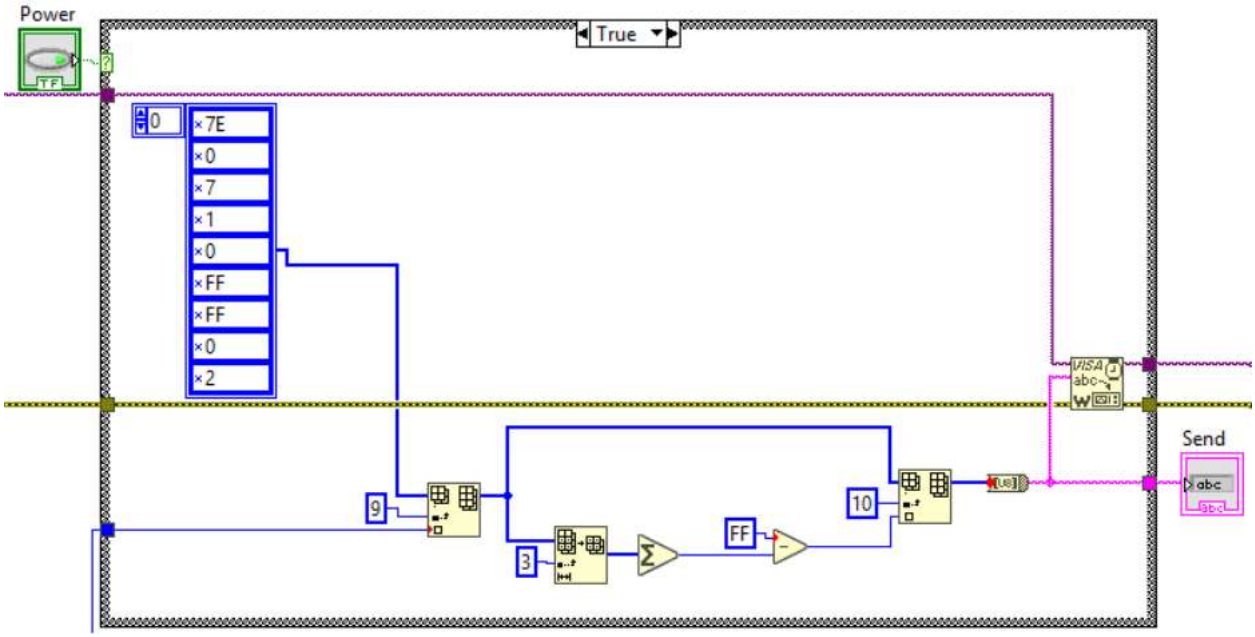


Figure 18: Constructing API-frame [XBee Send.vi]

To build the frame an array of unsigned 8-bit numbers was used. As usual the frame must start with the frame delimiter 0x7E followed by the length specified in two bytes namely a MSB and LSB. The length is seven bytes in this case. To send data to a 16-bit address the API identifier is 0x01. The next byte is set to 0x00 since this option is not necessary for this application. As mentioned before the specific addressing did not work, therefore it was chosen to broadcast the data. To broadcast the data the address bytes must be set to 0xFFFF. After the address follows another option which is set to 0x00 as this option was not necessary either. The bytes that follow this option is the data being transmitted. The first byte of the data is 0x02 and is a chosen address for the receiving actuator. Now the other data must be added to the array. In figure 18 it is shown that there are functions to add elements to an array. This function is used to add the data coming from the previous step of the LabVIEW program. The last element that must be added to the frame is the checksum. To calculate this value the sum is made of all the data in the frame after the length and then the sum is subtracted from 0xFF. This last value is also added to the array using the same function as before. Once the frame is built it should look like the frame in table 2. At the end the array must be converted into a string so that the data can be written via VISA to the XBee connected to the computer.

Table 2: Summary frame to be sent

Byte-n°	1	2	3	4	5	6	7	8	9	10	11
Byte value	0x7E	0x00	0x07	0x01	0x00	0xFF	0xFF	0x00	0x02	0x??	0x??
Meaning	Frame delimiter	MSB for length	LSB for length	API-identifier	Byte for options	MSB and LSB for address -0xFFFF is for broadcasting		Byte for options	Address chosen	Data to be sent	Calculated checksum

4.3 LabVIEW code

The programs to send and receive data between the XBees were transformed into SubVI's and added into the total code shown in figure 19.

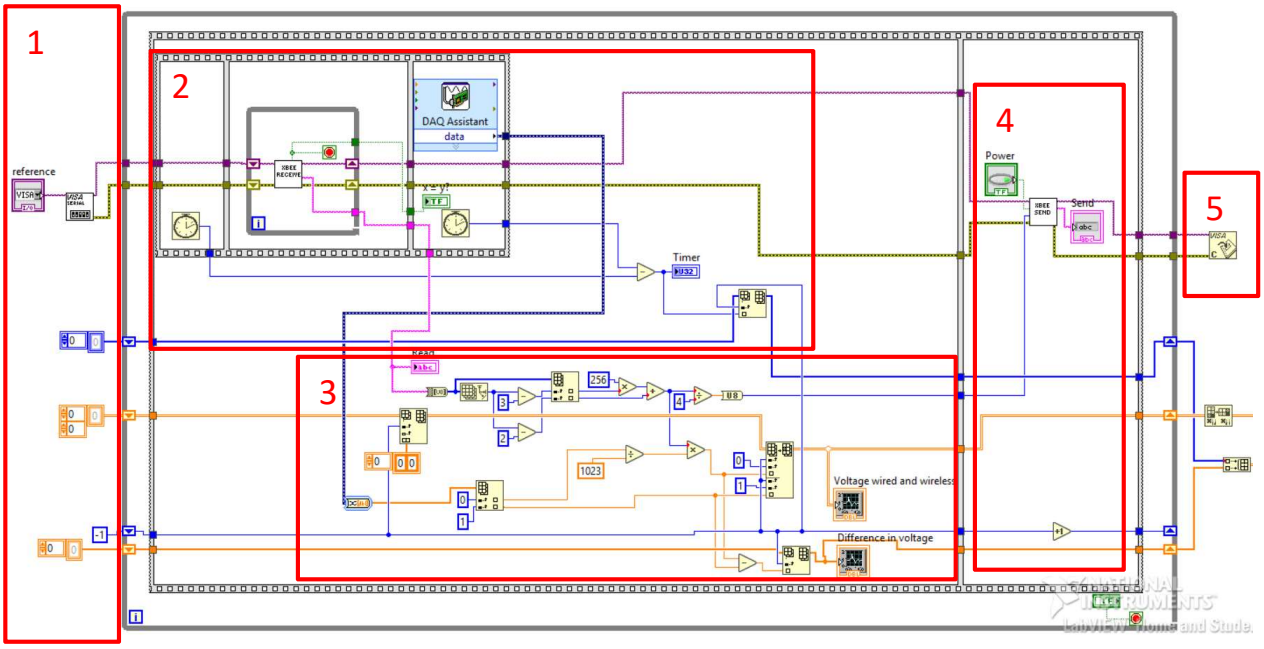


Figure 19: Total Code LabVIEW

The program consists out of five main parts. The first part is the initialization of several parameters or arrays and the opening of a serial communication with the XBee using the VISA resource. The second part is a flat sequence containing timers, the SubVI to receive wireless data and a DAQ assistant VI for the myDAQ to receive the wired data. The timers in this sequence are used to time the duration of the while loop containing the SubVI to read the XBee data. The SubVI is put in a while loop because otherwise the program would continue before receiving the entire frame. If the checksum at the end of the frame is correct, a signal that the message is received causes the while loop to be exited. As soon as the loop is exited the DAQ assistant VI is triggered to read the data via the wired connection. The DAQ assistant is located right behind the SubVI so that the difference in time between the measurements is as low as possible. After the sequence the difference is made between the two timers to check the total time of the while loop and added to an array to preserve the data.

The third part is enlarged in figure 20.

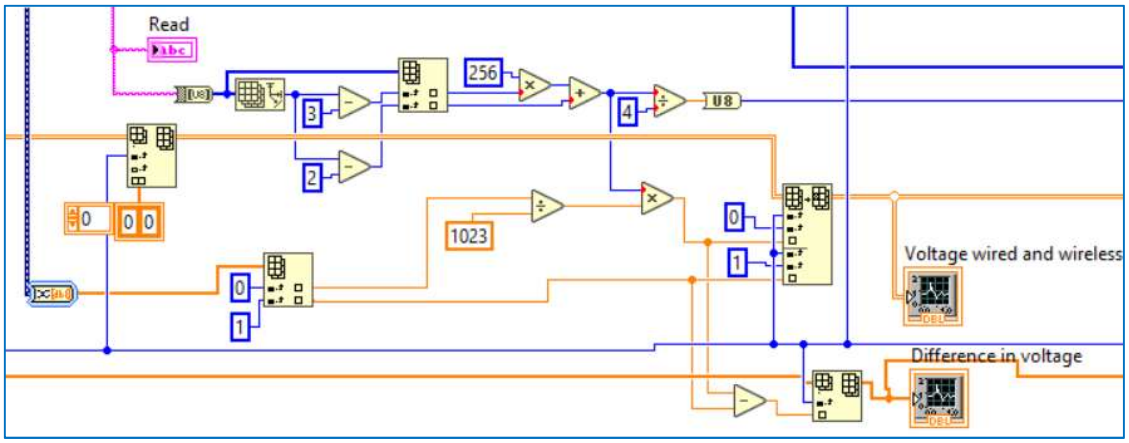


Figure 20: Total Code LabVIEW (part 3)

In the third part the digital data is transformed into the corresponding voltage. The following steps are taken.

The read data is a string and must be converted to an array to access the right data. After transforming the string to an array, the size of this array is queried. Using this size and subtracting three and two the data corresponding with the voltage can be taking out of the array. Two and three are used because the index of the first element in an array is zero and the last element is the checksum. For example:

- 7E 00 0A 83 00 00 22 00 01 00 11 03 FF 48 is the received frame;
- The size is 14 and the data needed is underlined;
- In the array the underlined elements have index 14-3=11 and 14-2=12 while the checksum has index 13, one less than the size.

The data sent by the XBee has a 10-bit resolution and each element in the array is eight bits. Therefore, the most significant byte must be multiplied by 256 and added to the least significant byte. This value corresponds to a 10-bit value ranging from zero to 1023. Because an Arduino is used to write a PWM signal this 10-bit value must be divided by four so the range changes from zero to 255 which corresponds to the range of the PWM signal an Arduino can deliver.

The data read from the myDAQ is also transferred into an array. The first element of the array is the reference voltage of 3,3 volts. This voltage is divided by 1023 to be multiplied by the data received from the XBee. This reference was voltage from the myDAQ corresponds perfectly with the reference voltage connected to the ADC from the Arduino. The second element of the array is the measured voltage. No transformations need to be done.

The measured voltage received through the XBee and the myDAQ are added in an array that was initialized before the outer while loop. The difference is also calculated and added into a different array. Both these arrays are then plotted into two graphs to show the trend in time.

Part four is in the second part of the sequence. Here the PWM data is sent to the XBee of the actuator. On the bottom a counter is increased by one to count the loops to put the data in the right position of the array.

When the stop-button is pressed part five will close the serial communication, and save the data from the arrays into an excel file. The part for saving the data is not entirely shown in figure 19 but is in figure 21.

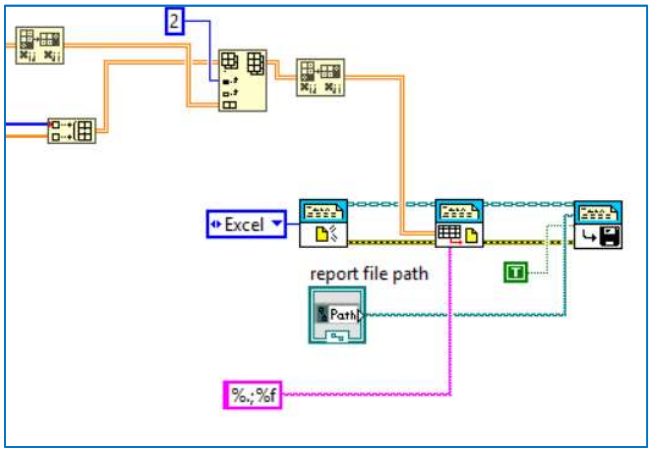


Figure 21: Saving data to excel

All the data is put in one large 2D-array. This array is then saved into an excel-file to analyse the data.

4.4 Front panel

In LabVIEW a front panel is created to monitor and control the program. The front panel of this program contains two graphs, one with the voltages and another one with the difference between these voltages. It also contains some indicators to show the received and sent string, the value of the timer, and a Boolean LED to indicate if the checksum was correct. Before starting the application, the correct port must be selected under 'reference' to connect to the XBee on the USB-port, and the right path to the excel-file to save the data. Finally, there are two buttons, one is the stop-button to stop the program correctly and one to indicate if the actuator should be powered or not. All this is shown in figure 22.

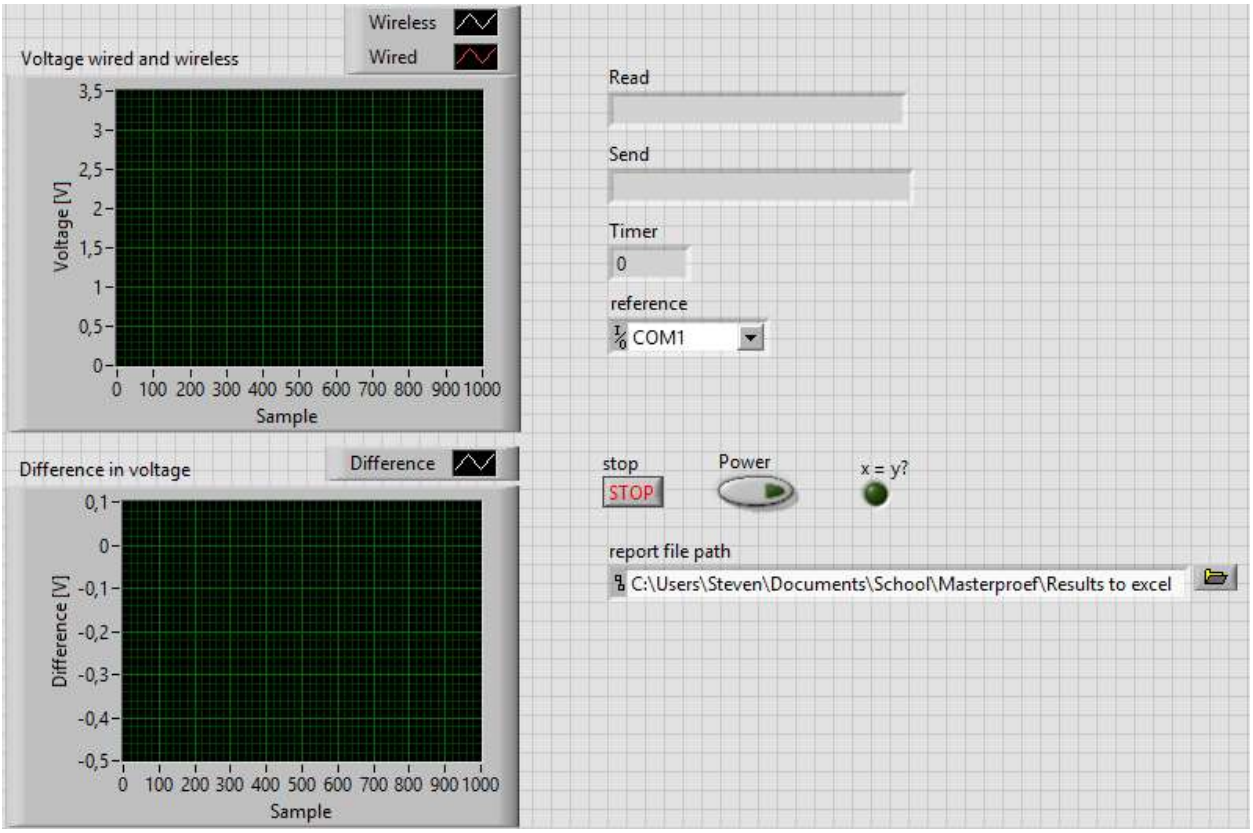


Figure 22:Front Panel

5. Testing of the characteristics

5.1 Hardware

The hardware parts are divided into three modules. The first module is the computer with XBee explorer and XBee antenna.

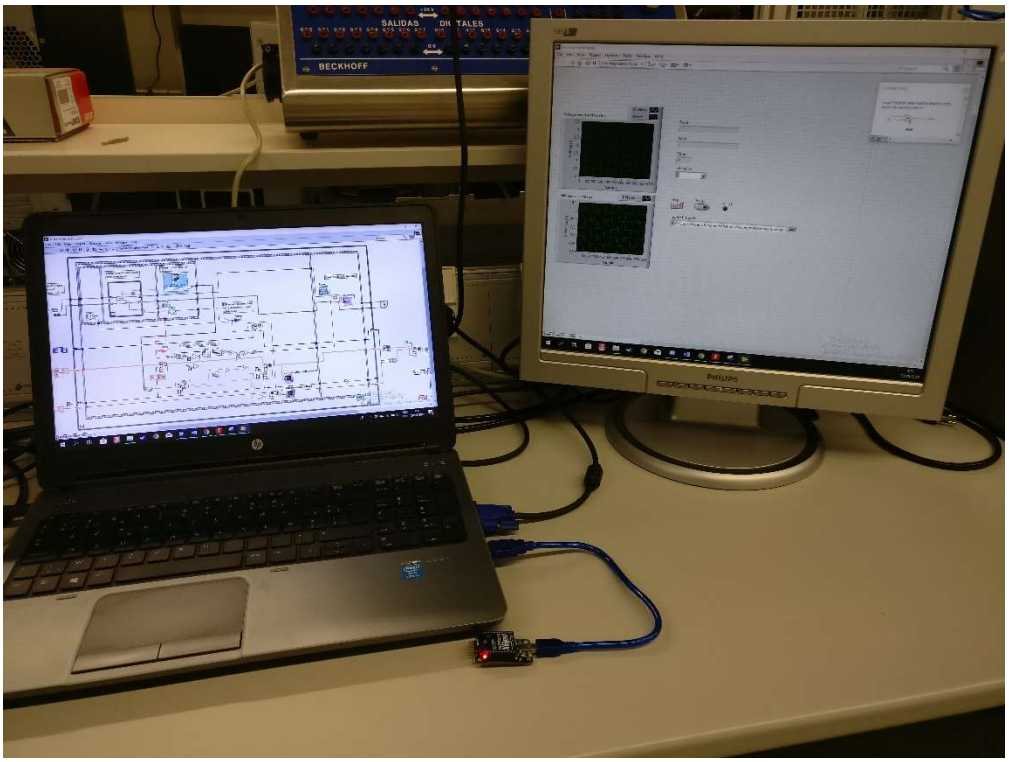


Figure 23: Computer setup

As seen in the photo in figure 23, the computer runs the LabVIEW program and to one of the USB-ports the first XBee is connected using the explorer.

The second module is the sensor-module. Using a power adapter and a 9V battery two different voltages are provided, one of 3.3V and one of 5V. This adapter can be seen in figure 24.

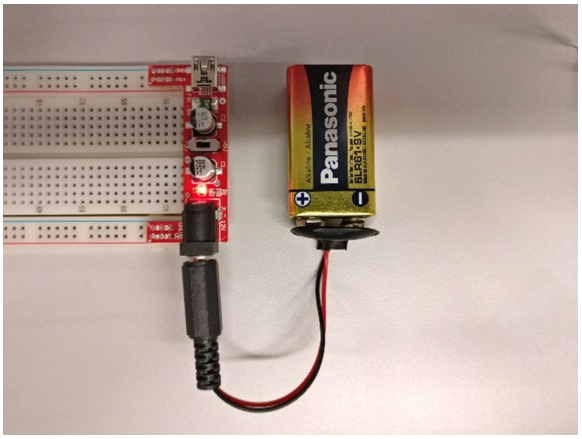


Figure 24: Power adapter

The 5V is used to power an Arduino with XBee shield and XBee while the 3.3V is connected to the reference voltage for the ADC on the Arduino, and to the first pin of a potentiometer. The voltage coming from the middle pin of the potentiometer is connected to analog input A0. The connections are illustrated in figure 25.

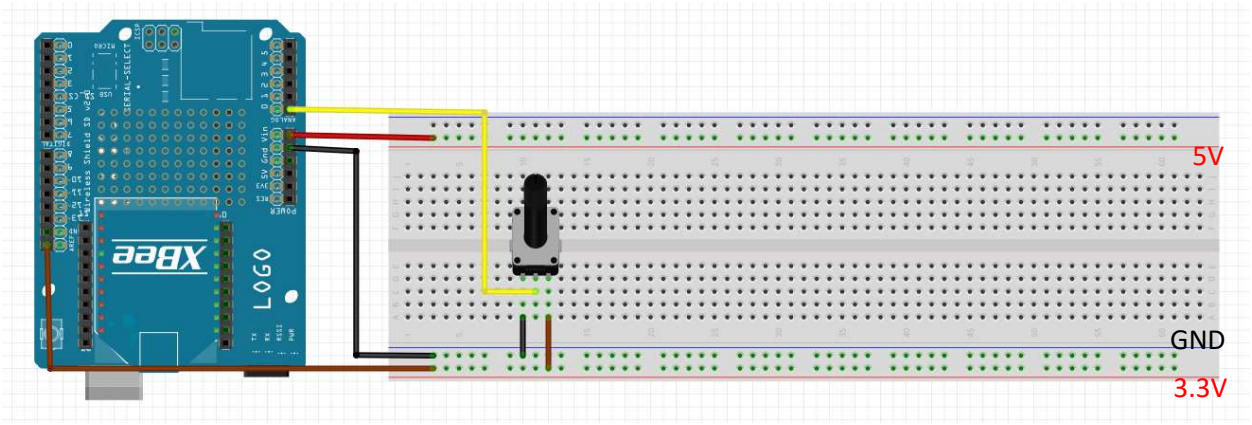


Figure 25: Connection schematic sensor-module

The last module is the actuator module, it is very similar to the sensor module. This module also contains an Arduino with XBee shield and XBee. Next to these components it contains the actuator, which in this case is a DC-motor. The Arduino will send out a PWM signal to the transistor which in his turn controls the speed of the motor. A 9V battery provides power to the motor. These conections are illustrated in figure 26.

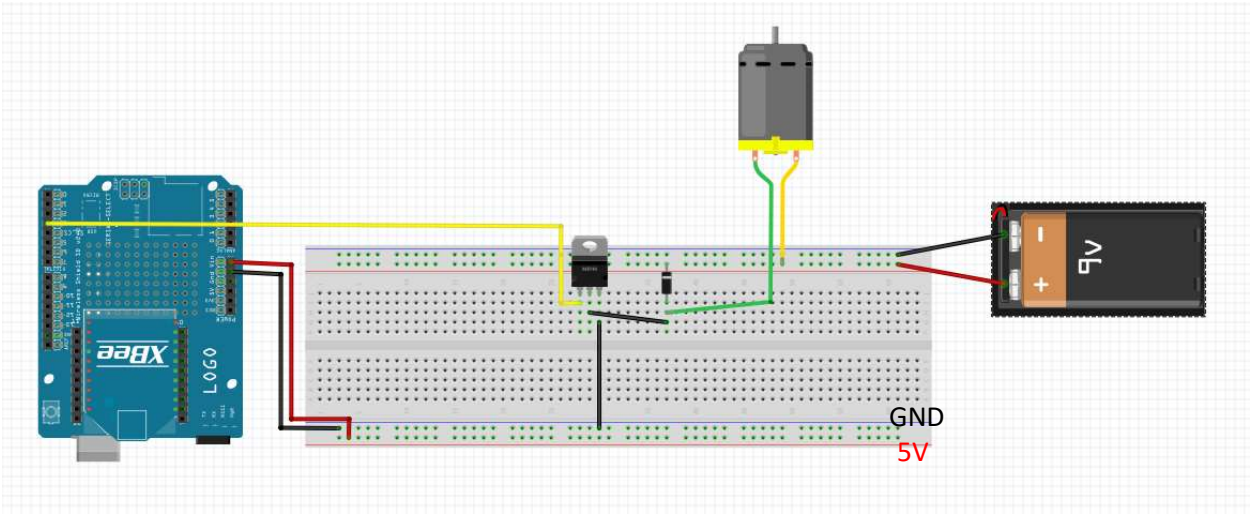


Figure 26: Connection schematic actuator-module

Both Arduinos have a small code written on them. On one side this code was necessary to be able to send the data read by the ADC of the sensor-Arduino, and on the other side to translate the received data into a PWM-signal for the motor. Figure 27 and 28 respectively show the code for the sensor-Arduino and the actuator-Arduino.

```
#define sensorPin A0
word sensorValue=0;

void setup() {
  Serial.begin(9600);
  analogReference(EXTERNAL);
}

void loop() {
  sensorValue=analogRead(sensorPin);



  Serial.write(sensorValue/256);
  Serial.write(sensorValue%256);

  delay(50);
}
```

Figure 27: Code Sensor

```
int PWMPin = 3;
void setup(){
  Serial.begin(9600);
}
void loop(){
  if (Serial.available()>1){
    if(Serial.read()==0x02){
      analogWrite(PWMPin, Serial.read());
    }
  }
  delay(5);
}
```

Figure 27: Code Actuator

The code for the sensor first defines the sensor pin and initializes a parameter to store the value of the sensor. In the setup the serial communication is started at a baud-rate of 9600 and changes the reference voltage from the usual 5V to the voltage that is connected externally to the ARef pin of the Arduino. In the loop the voltage of the sensor is read first and stored into the parameter sensorValue. Since the ADC of an Arduino is a 10-bit value and the data being send over the Serial communication can only be an 8-bit value, the value must be divided by 256 to calculate the MSB and next the remainder of a division by 256 must be calculated to determine the LSB. These values are send one by one using serial.write(). The delay in the loop determines what the sample rate is of the sensor.

For the code of the actuator the PWM pin is chosen and initialized. In the setup the serial communication is started at a baud-rate of 9600. The loop checks if there are two bytes available on the buffer. These two bytes are the address decided in LabVIEW and the PWM value. If the two bytes are available the second if-statement checks if the address matches the address chosen for the Arduino. In this case the Arduino has address 0x02 as explained in chapter 4.2. If this is true, the next byte is written to the PWM-pin to control the speed of the motor. A small delay is added to save power but is not necessary.

5.2 Experiments

Several experiments were designed to test the different characteristics of the test configuration. The first test was designed to investigate the precision of the ADC in an XBee and compare it to other ADCs. According to the datasheet the ADC transforms the analog voltage into a 10-bit value. This value was compared to a value delivered by a NI myDAQ USB-6008. The myDAQ uses a 12-bit value to transform the voltage. Because of this the myDAQ has a higher resolution. Since it is also designed by National Instruments the compatibility with LabVIEW is very high and the data is very trustworthy. After comparing these two ADC's a third ADC is added to the comparison, namely the one built in the Arduino. This ADC also converts the voltage into a 10-bit value. The data measured by the Arduino is then transmitted to LabVIEW using a wired connection first, and a wireless connection using XBee next. All these tests were done at 0.5 meters and a sample period of 100 milliseconds. By doing these comparisons the most accurate wireless solution is chosen.

Once the best solution is chosen three extra experiments are done. The first of these three experiments was to investigate the effect on several characteristics by changing the sample period, and the distance between the computer and the receiving modules. These tests were done with a sample period of 30, 50 and 100 milliseconds and at a distance of 0, 0.5, 2 and 5 meters.

The second experiment was to measure the delay between sending a signal to the computer and receiving it back at the actuator. For this the code in the Arduino's was slightly altered to send/receive a single bit instead of the two bytes or a PWM value, and the sample period was eliminated. The time difference between sending and receiving was then measured using an oscilloscope as shown in figure 29. These tests were only done at a distance of 0, 0.5, 2 and 5 meters. Next to the wireless connection the wired connection was tested in the same manner.

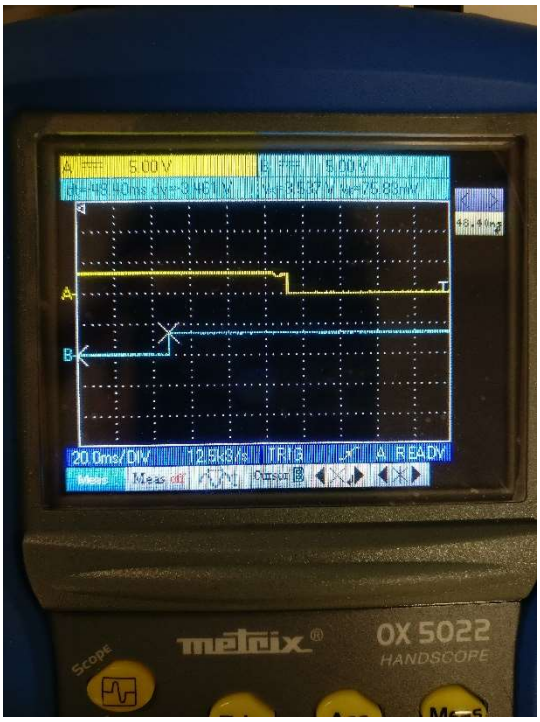


Figure 28: Oscilloscope to measure delay

The last experiment investigated the effect of Wi-Fi on the XBee signals. Since both XBee and Wi-Fi work at 2.4GHz there is a chance of interferences. To test the effects a video of 4K quality was downloaded while the application using XBees was working. The results are then analysed to see if there was any dropout due to the interference.

5.3 Results

Experiment 1: Comparison of the ADCs

In figure 31 are all the comparisons gathered in one figure from the first experiment. For every comparison two graphs are drawn, one that shows the actual voltages measured by the compared ADCs and another that shows the difference between these two voltages.

The graphs in A compare the ADC from the XBee with the myDAQ. It shows that the difference between the two signals is significant for low voltages but less for higher voltages. The error seems to be linear to the voltage. Figure 30 shows the conversion of the voltage to the corresponding voltage according to both ADCs. The myDAQ has an offset that is close to zero but the XBee has an offset around 0.4V at 0V. The higher the voltage becomes the lower de difference between the true voltage and the voltage measured by the XBee.

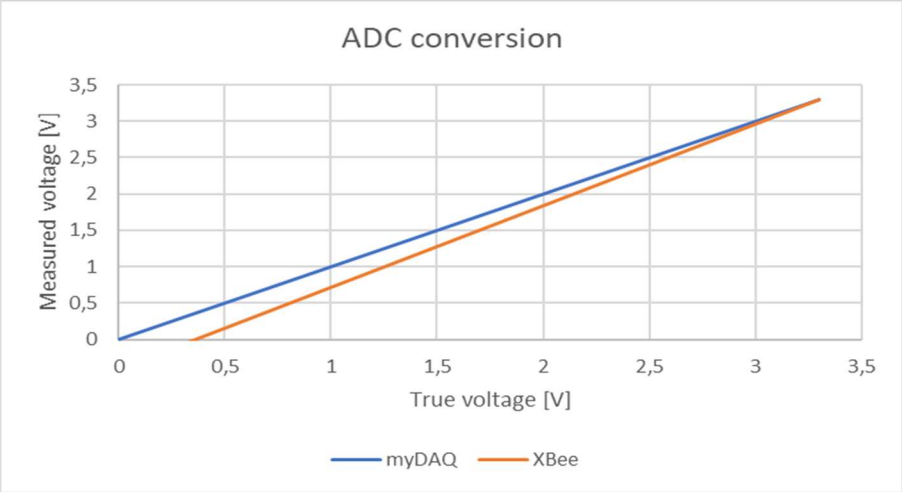
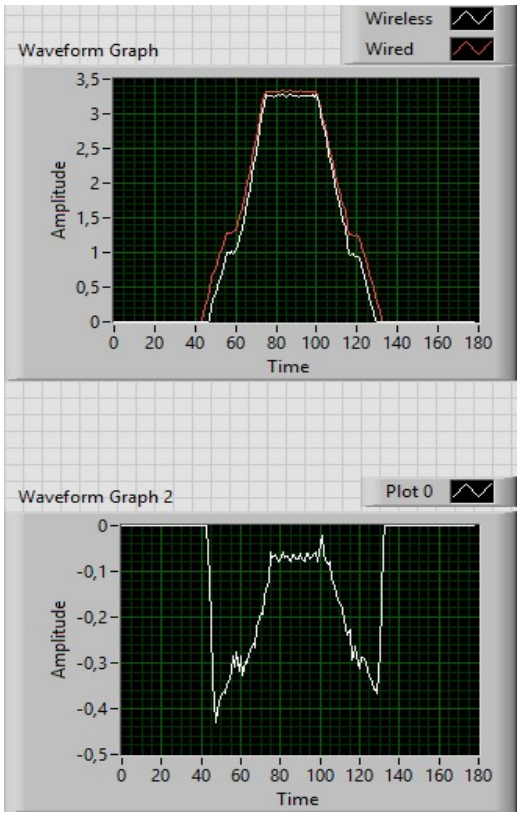


Figure 29: ADC conversion

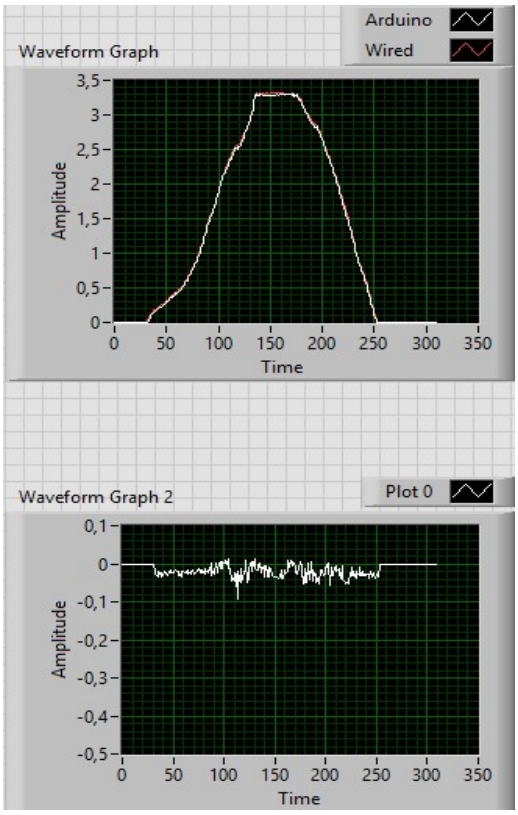
B compares the myDAQ with the Arduino. Here the signals are immediately a lot closer as the maximum difference between the two signals is only 0.1V and average around 0.03V.

C is a comparison between the XBee and the Arduino. Again, the signal has a certain offset at a low voltage and a lower difference at a higher voltage. The offset in the beginning seems to be a bit smaller. This can be caused by the imprecision of the XBee or because the Arduino is a bit less precise than the myDAQ.

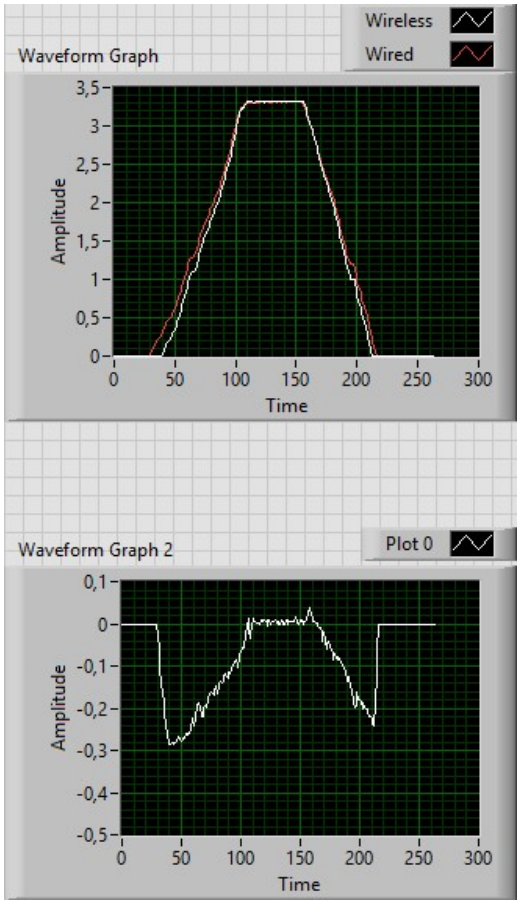
Finally, D shows the comparison of the Arduino transmitting the data through XBee and the myDAQ. This characteristic is very close to B which is normal since the measurements are done by the same ADCs. Since this last graph is the most precise result of a wireless connection, this configuration is chosen as the standard configuration for the other experiments.



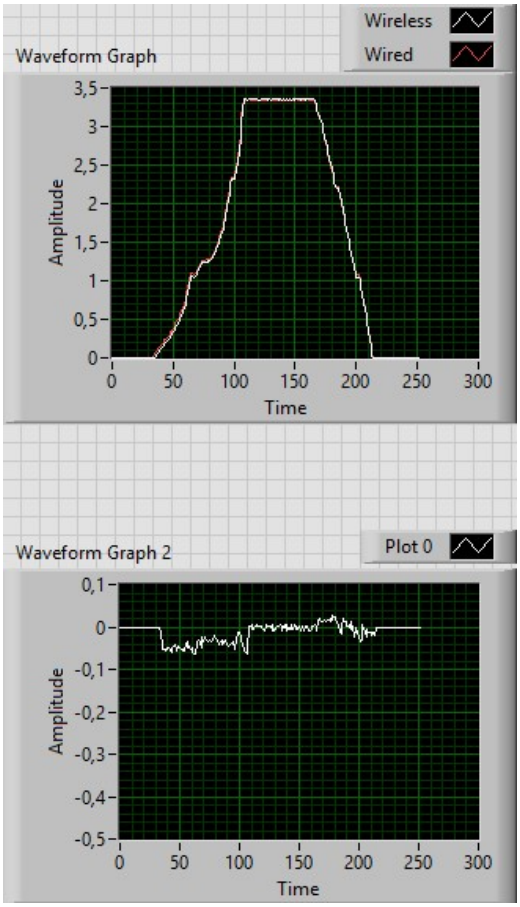
(A)



(B)



(C)



(D)

Figure 30: XBee vs. myDAQ (A) + Arduino vs. myDAQ (B) + XBee vs. Arduino (C) + Arduino with XBee vs. myDAQ (D)

Experiment 2: Comparison of offset and message dropout at different distances and sample periods

The second experiment concluded a lot of data. To summarize this data the average and the standard deviation of several measurements, and of all the measurements is taken. The selected measurements are at the voltages of 1V, 2V and 3.3V. The message dropout is also determined next to the averages and the standard deviation. This summary is given in table 2. One graph is shown in figure 32 as an example, the other graphs corresponding to the other measurements look similar and are added in appendix A.

Table 3: Summary experiment with varying distance and sample period

Test	Distance [m]	5			2			0,5			0		
	Sampling period [ms]	30	50	100	30	50	100	30	50	100	30	50	100
Message dropout [%]		1,3	1,2	1,3	1,8	2,1	2,2	0,9	2,3	1,7	1,6	2,2	2,3
Av. Diff. at 1V [mV]		-19	-60	-85	-29	-29	-45	-24	-23	-22	-22	-32	-21
Standard deviation ↑ [mV]		15	12	20	12	9,4	16	4,6	5,4	5,4	7,9	10	7,6
Av. Diff. at 2V [mV]		-8,5	-25	-39	-16	-12	-15	-14	-11	-12	-12	-13	-5,8
Standard deviation ↑ [mV]		12	16	16	11	23	12	5,0	2,9	4,4	8,3	9,4	14
Av. Diff. at 3.3V [mV]		0,9	2,2	0,8	-0,3	1,1	1,2	-1,7	-2,3	-2,4	-1,0	1,3	-2,0
Standard deviation ↑ [mV]		6,2	5,7	8,2	19	7,9	7,2	5,2	4,9	6,2	6,7	5,6	13
Av. Diff. total [mV]		-9,9	-32	-40	-14	-13	-18	-14	-13	-13	-14	-16	-12
Standard deviation ↑ [mV]		16	31	40	19	19	25	16	14	14	17	19	16

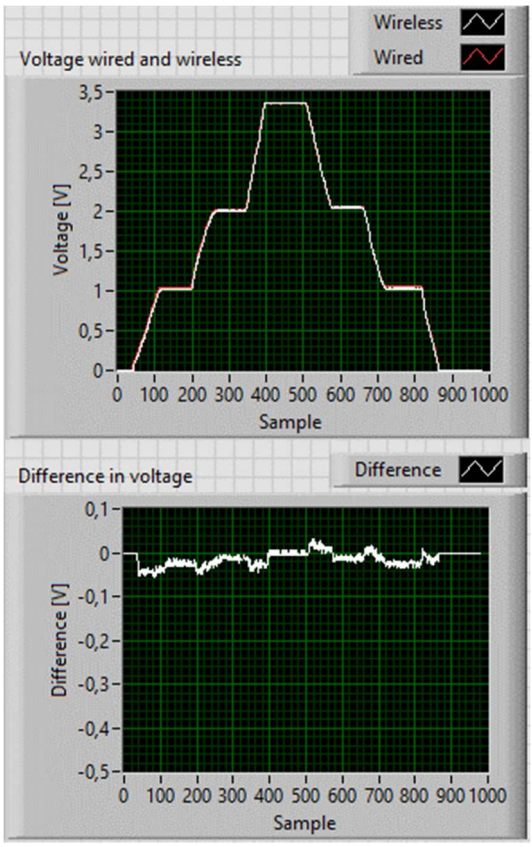


Figure 31: Graph example (30ms/0.5m)

The first thing to notice in table 2 is that the message dropout is low, averaging below 2%. You can also notice that there is not really a trend or correlation between the message dropout, and the distance and/or sampling period. The irregularity is most likely due to interference which could have been higher or lower during testing, thus throwing off the data. To eliminate this issue the test should be repeated in a controlled environment.

In the other data it is also hard to find a correlation or trend. The only clear trend is that the offset becomes smaller when the voltage gets higher. This is because of the same reason as the ADC from the XBee. This was not noticed before because the difference is less clear on the graphs. This can be seen in figure 32 why it was not clear. Whenever there is a rising edge the difference in voltage increases in the negative direction and for a dropping edge it increases in the positive direction. This is probably due to a small delay between the wired measurement and the wireless measurement. These little peaks seem to flatten the graph showing the difference.

A last remark is that at each different voltage the average stays mostly the same with several outliers. The most remarkable outliers occurred at five meters. The reason for this is still not clear.

Experiment 3: Message delay at different distances and sample periods

For the third experiment the message delay was measured. For this the potentiometer was replaced with a pushbutton and the motor by a LED. The sensor and actuator were put at several distances from the computer. The mean and standard deviation of ten measurements are shown in table 4.

Table 4: Message delay measurements

Test	Distance	Wired	0 m	0,5 m	2 m	5 m
Mean 36[ms]		3	45,9	45,5	46,2	47,2
Std. Dev.		0,60	3,93	5,25	4,92	6,39

Two things can be noticed, firstly the distance doesn't seem to have an influence on the delay, and secondly the delay is significantly lower in the wired connection. This is because the XBees transmit the data in a frame at a baud-rate of 9600 bits per second. These frames can be around 100 bits which means it already takes around 10ms to send one frame. If you consider the several steps that must be taken, it fits that it takes around 46 milliseconds for the wireless connection to send and receive the data back. All these different steps also add to a higher standard deviation. To reduce the time needed to translate the data into a frame a higher baud-rate can be used. A second test was done at two different baud-rates. The results are shown in table 5.

Table 5: Results delay at different baud-rate

Test	Distance	9600	38400	57600
Mean		45,9	24,1	22,6
Std. Dev.		3,93	4,07	3,95

The table shows that the delay is lower for a higher baud-rate. This is because the data is transmitted faster. For example, at a baud-rate of 38400 the time to send one frame will be four times smaller equalling 2.5ms. If the time used by the computer to process the data is considered the same it can be deduced that there are three locations where the data is delayed by translation to a frame. It is not certain where these locations are but a hypothesis is suggested in figure 33.

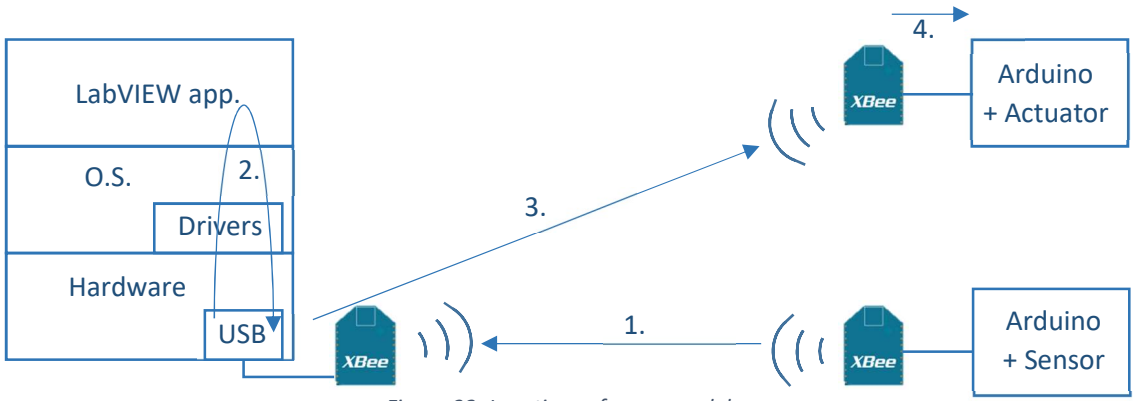


Figure 32: Locations of message delay

1. The first location is when the XBee translates the data into a frame and transmits it through radio waves at the specified baud-rate. For 9600 the time is around 10ms, for 38400 it is 2.5 ms;
2. The message is received and passed through the computer bit by bit. An unknown time is lost here due to the speed of the entire computer process;
3. After everything is processed, the generated frame is communicated serially to the USB port and immediately transmitted by the XBee. This causes another delay of around 10ms for a baud-rate of 9600 or 2.5ms for 38400;
4. After the entire frame is received in the final XBee, the frame is analysed and the received byte is send to the Arduino. Since the XBee must wait for the entire frame to arrive another delay of 10 or 2.5ms is created.

By taking the sum of all these delays, the processing time of the computer can be calculated around 16ms. Since this is just a hypothesis, further research is suggested.

For the wired connection, only a single bit must be communicated which only takes 0.1ms. The wired connection still has a delay of three milliseconds due to the speed of the computer's operating system.

Depending on the application these delays can be tolerated or not. For both the wireless connection and the wired connection it can be interesting to use a real-time operating system. This operating system can reduce and stabilize the delay created by the computer [8].

Experiment 4: Interference with WiFi

The last experiment was a short one. While downloading a high-resolution video the message dropout was checked. The download of the video happened at 4,9Mbps as shown in figure 34. After reviewing the results, a peak moment was chosen where many messages were lost. In that peek moment up to 60% of the messages were not received by the XBee on the computer.

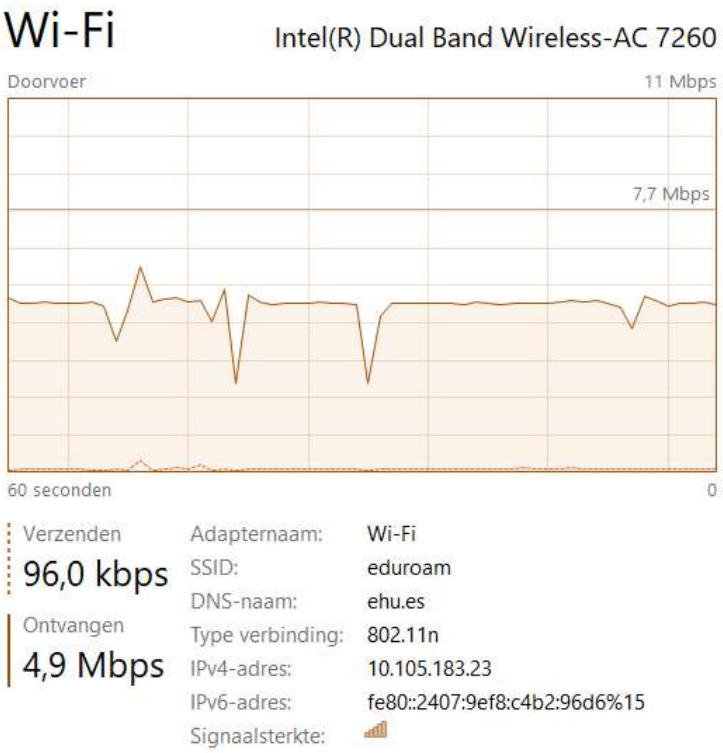


Figure 33: Wi-Fi download speed

This significant dropout is caused by interference because both XBee and Wi-Fi work at 2.4GHz. This interference can be avoided by either using the XBees in a Wi-Fi free zone or using different XBees that work on a different frequency.

6. Conclusion

In this project it was chosen to work with XBee antennae. The choice was made for advantages such as increasing the flexibility and reducing the wiring. But this wireless communication also creates several challenges. As mentioned in the introduction it is challenging to find the optimal sampling period, analyse and anticipate message delay, reduce message dropout and keep the power consumption at a low level.

After doing several experiments, some conclusions could be drawn. Some of these conclusions were positive while others were more negative.

The first positive conclusion of working with XBee antennae and LabVIEW is the reliability of the message being sent and received. Every message that is received contains the correct data in case the checksum is correct. By calculating this every time, it is certain that the received message was correct. Next to that, the message dropout was very low. In normal circumstances off a classroom this dropout was only 2%. During testing it was also noticed that the flexibility of using XBee antennae is very high. It was easy to replace or relocate them at the different distances. For LabVIEW specifically, it was very useful to monitor the data on the front panel offered by LabVIEW.

Nonetheless there were also some negative conclusions. Because the data must be put into a frame, a delay is created. This delay can be reduced by increasing the baud-rate. Not only the frames contributed to the delay, but also the computer. This delay in the computer can be reduced and stabilized by using a real-time operating system. In general, these delays were also very inconsistent which makes it hard to predict the delay and adapt the control loop to the delay. Finally, it was also noticed that the message dropout becomes very high when there is large interference with Wi-Fi.

A more neutral finding was that the ADC used by the Arduino was precise enough but for some applications it may not be. Since the XBee can send more data than the Arduino provides any other ADC can be used as well to improve the resolution and the precision of the data.

In general, it can be concluded that for none critical systems or monitoring the XBee provides a sufficient solution. For more critical solutions it is recommended to improve several factors of the configuration used in this work such as using a better ADC, controlling the environment to reduce interference, increasing the baud-rate and using a real-time operating system to reduce and better anticipate the message delay.

References

[1] B. Wittenmark, K. J. Åström, and K. E. Årzen, “Computer Control: An Overview,” *IFAC Prof. Br.*, pp. 1–82, 2002.

[2] P. Park, S. C. Ergen, C. Fischione, C. Lu, and K. H. Johansson, “Wireless Network Design for Control Systems: A Survey,” *IEEE Commun. Surv. Tutorials*, no. c, pp. 1–36, 2017.

[3] Hans-Petter Halvorsen, “Wireless Data Acquisition in LabVIEW,” 2011.

[4] MaxStream, “XBee ™ / XBee-PRO ™ OEM RF Modules,” *MaxStream*, no. 801, pp. 1–33, 2005.

[5] C. Stobing, “What Does WiFi Stand For and How Does Wifi Work?,” 2016. [Online]. Available: http://www.gadgetreview.com/what-is-wifi-what-does-wifi-stand-for-how-does-it-work. [Accessed: 23-May-2018].

[6] N. N. Instruments, “NI-VISA Overview,” 2018. [Online]. Available: http://www.ni.com/tutorial/3702/en/. [Accessed: 23-May-2018].

[7] C. Calegari, “XBee Aula 2 - Modo API e Labview,” 2016. [Online]. Available: https://www.youtube.com/watch?v=_DMiW87_5nw&t=262s. [Accessed: 15-Apr-2018].

[8] N. N. Instruments, “Building a Real-Time System With NI Hardware and Software,” 2018. [Online]. Available: http://www.ni.com/white-paper/4040/en/. [Accessed: 26-May-2018].

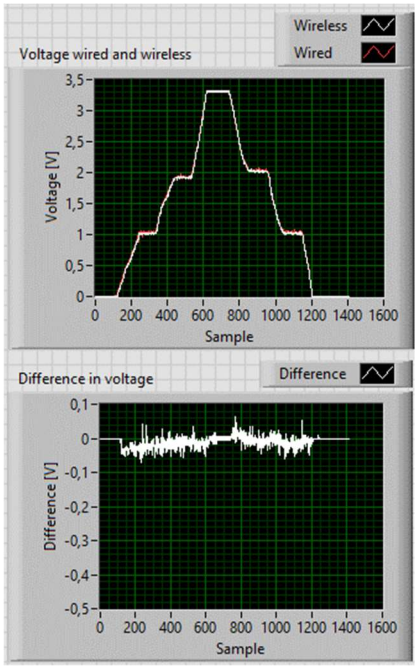
[9] Ardobot, “Xbee Explorer USB,” 2018. [Online]. Available: https://www.ardobot.com/adaptador-xbee-explorer-usb.html. [Accessed: 22-May-2018].

[10] S. Techmake, “XBee Shield para Arduino,” 2017. [Online]. Available: http://www.techmake.com/00422.html. [Accessed: 22-May-2018].

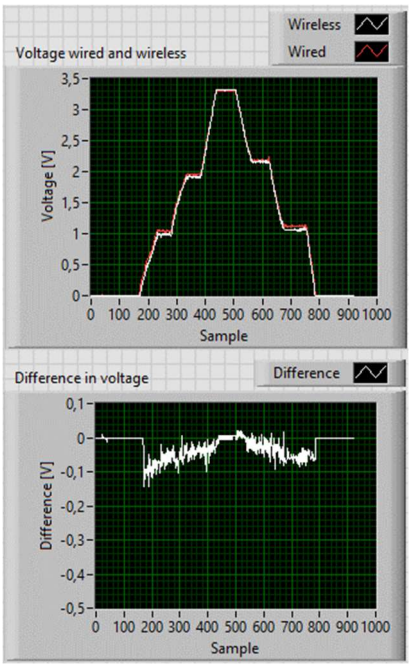
[11] Sparkfun, “XBee 1mW Trace Antenna,” 2018. [Online]. Available: https://www.sparkfun.com/products/11215. [Accessed: 22-May-2018].

Appendices

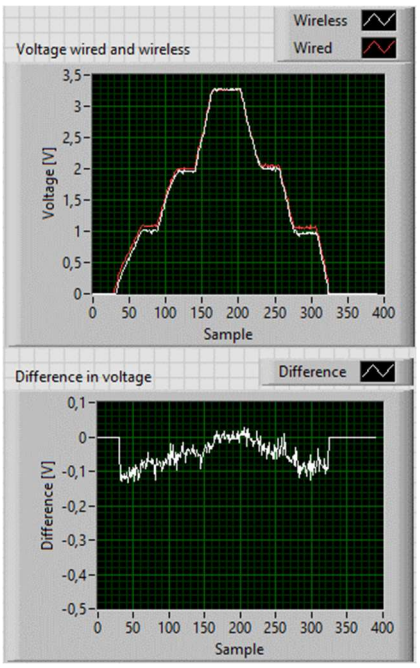
Appendix A:



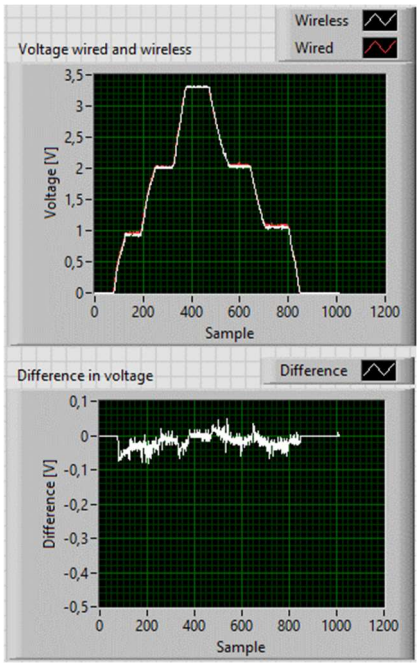
5m/30ms



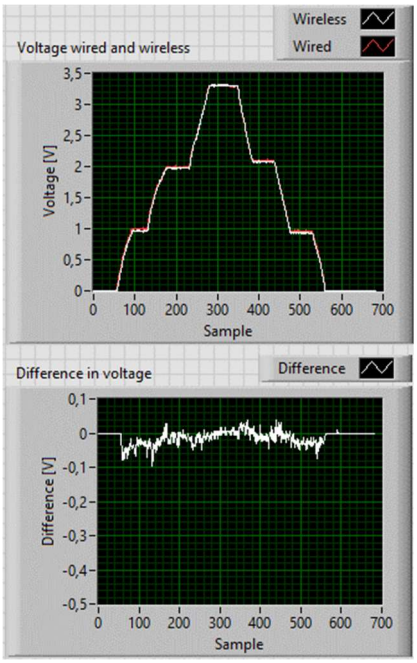
5m/50ms



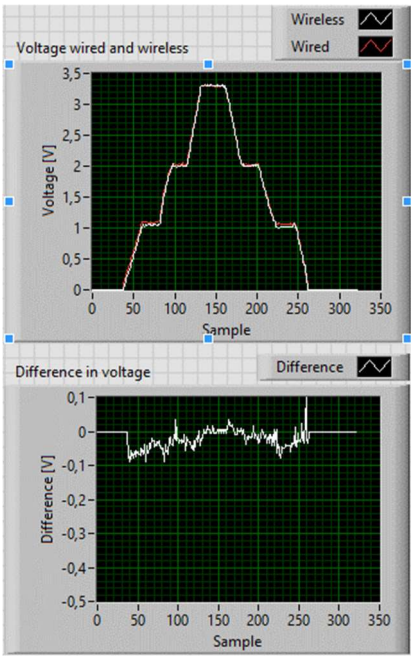
5m/100ms



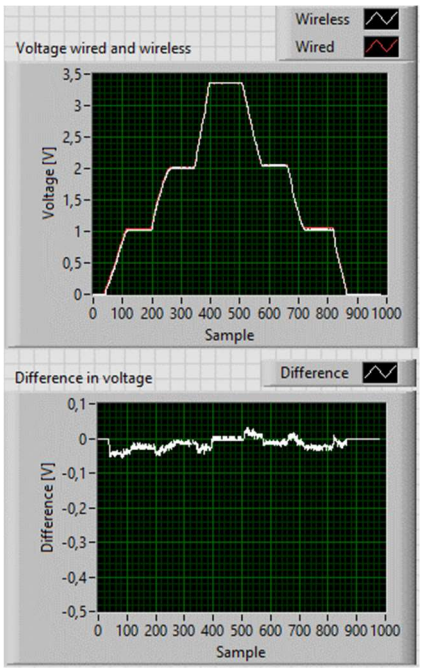
2m/30ms



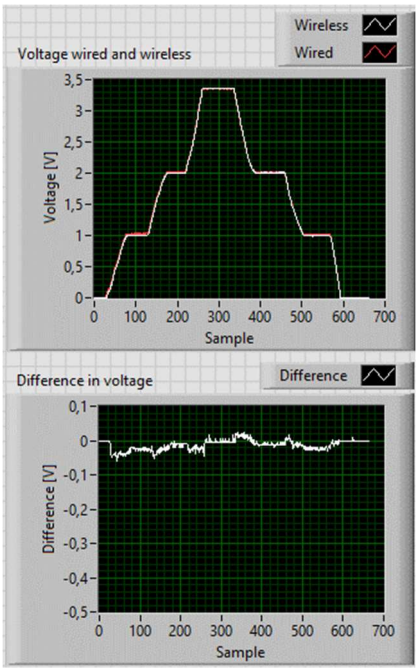
2m/50ms



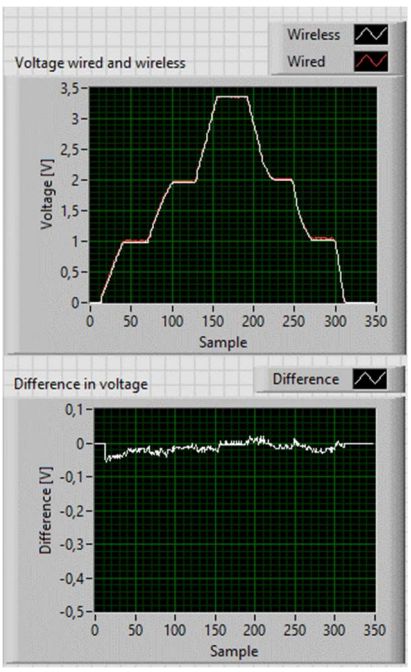
2m/100ms



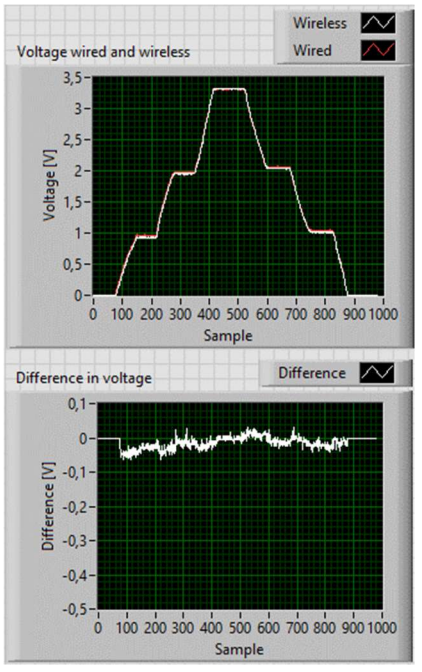
0.5m/30ms



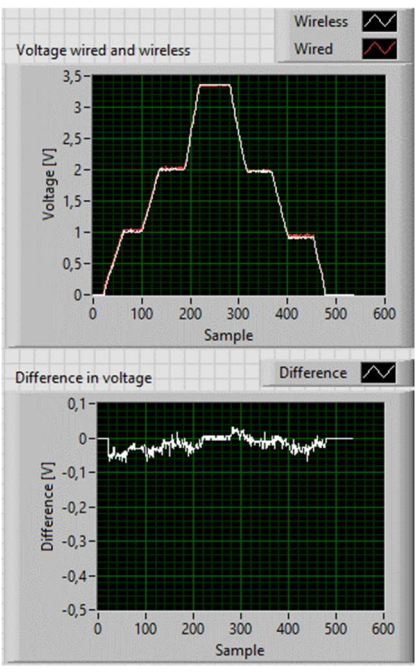
0.5m/50ms



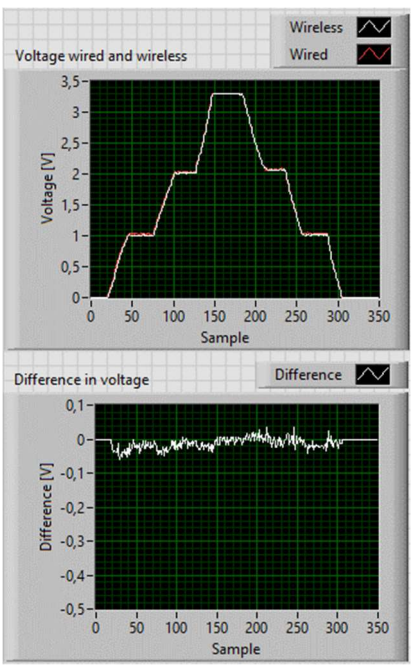
0.5m/100ms



0m/30ms



0m/50ms



0m/100ms

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Creation of a Wireless Networked Control System using XBee antennae and LabView

Richting: **master in de industriële wetenschappen: energie-automatisering**
Jaar: **2018**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Abrahams, Steven

Datum: **4/06/2018**