eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Escuela Universitaria de Ingeniería Vitoria-Gasteiz

Ingeniaritzako Unibertsitate Eskola Vitoria-Gasteiz

UHASSELT

KU LEUVEN

Final career project

Integration of a PI controller in a wireless control network.

Name student(s): Geert Beckers & Dieter Slechten

Name promoters (and company/research centre): Johan Baeten (UHasselt/KU Leuven), Isidro Calvo & Oscar Barambones (Universidad del País Vasco)

specialisation: Energy – automation

Preface

The paper you are about to read is named: “Integration of a PI controller in a wireless control network”. The research for the paper was done at the university of the Basque country, UPV/EHU. This paper serves as the final thesis for both our Master’s degrees in industrial engineering at KU Leuven and UHasselt in the academic year of 2018-2019.

First and foremost we would like to thank our external promotor Prof. Dr. Ir. Isidro Calvo. Throughout this project he sat down with us many times and helped us get back on track when we were stuck.

Next we would to thank our internal promotor Prof. Dr. Ir. Johan Baeten as well as our external co-promotor Prof. Dr. Ir. Oscar Barambones. They were a great help with the controlling aspect of this paper.

Next we would also like to thank Ander Chouza who we could always ask for help or a different perspective.

Finally we would like to thank our friends and family for their help and support.

And last but not least we would like to thank you, the reader. We hope you find this paper interesting and informative.

Geert Beckers & Dieter Slechten

Vitoria-Gasteiz, January 2019

Table of contents

List of tables

List of figures

Abstract (English)

The university of the Basque country is doing research about implementing wireless networks into industrial processes. This paper focuses on the integration of a specific control structure in which wireless communications are used. Research will be done in order to determine the validity and reliability of using wireless communications in an industrial setting.

In order to test this an experimental setup was created to mimic an industrial process. A piezoelectric actuator, an Arduino Uno microcontroller and position sensor were used. The first experiments were done in order to determine the right control parameters. There were also a few experiments to analyse the package losses that were encountered. And finally different range tests were conducted to see how this effected package loss and latency.

The conclusion of this paper is that a PI controller can be implemented in a wireless control network and that it is a valid solution. However this will not be the case in every industrial process. In processes that require for instance real-time control, the latency will be too big to use wireless communication, also the package loss is something to take into account.

Abstract (Nederlands)

De Universiteit van het Baskenland doet onderzoek naar de mogelijkheid om draadloze netwerken te implementeren bij industriële processen. Deze paper focust op de integratie van een specifieke regelstructuur waarbij draadloze communicatie wordt gebruikt. Dit onderzoek moet de graad van validiteit en betrouwbaarheid van deze draadloze communicatie bewijzen in een industriële toepassing.

De gebruikte experimentele opstelling bootst een industrieel proces na. De opstelling bevat een piëzo-elektrische actuator, een Arduino Uno microcontroller en een positie sensor. De eerste experimenten werden gedaan om juiste controleparameters te achterhalen. Er werden ook een aantal experimenten gedaan om het data verlies te analyseren. De laatste reeks experimenten testen het effect van de afstand op dataverlies en looptijd.

De conclusie van deze paper is dat een PI controller weldegelijk kan geïmplementeerd worden in een draadloos regelnetwerk en dat het een goede oplossing is. Dit is echter niet geval voor elke industriële toepassing. Bij processen die bijvoorbeeld een real-time regeling nodig hebben, zal de looptijd ten gevolgen van de draadloze communicatie te groot zijn om een goede regeling te realiseren. Ook het dataverlies is iets waar rekening mee gehouden moet worden.

1 Introduction

1.1 Situational sketch

The university of the Basque country is doing research about implementing wireless networks into industrial processes. We are continuing the work of Steven Abrahams, he compared wired and wireless control applications. He researched if it was possible to use wireless technologies without losing too much accuracy and flexibility.

This project focuses more on the integration of a specific control structure in which wireless communications are used and researches if using wireless technology can be used in an industrial setting. To test this we are using a piezoelectric actuator and a position sensor.

Several tests will be carried out to test the viability of the technology. With these tools we can create a closed-loop control system. To accomplish this wireless network XBee antennas are used, these transmit the data from the sensor to the computer, and from the computer back to the actuator. The PI controller will be configured within LabVIEW and will eventually be uploaded onto a myRio device, with which it can run autonomously.

1.2 Domains of application

Wireless network can be implemented in many different domains. A couple of examples are vehicles, airplanes, industrial automation and building automation.

According to a study published in “IEEE communications Surveys & Tutorials” (Volume 20, Issue 2). The wiring harnesses used for the transmission of data and power in a vehicle can weigh 40 kg and contain sometimes up to 4 km of wiring. By using a wireless network there is room for improvement in fuel efficiency, reducing manufacturing and maintenance cost and easy way.

Through the high demands of safety and efficiency, most of the airplanes rely on large wired sensor and actuator networks. The wiring harnesses of airplanes weigh 2-5% of the total weight of an airplane. By reducing this weight, the performance and safety will improve while the manufacturing and maintenance cost will decrease.

Modern building control systems require a variety of sensing capabilities to control temperature, pressure, humidity and flow rates. By controlling their temperature, pressure, humidity and flow rates the energy consumption would be less. The implementation of these sensing capabilities would be much easier and cheaper if the sensors and actuators are wireless.

This paper focusses on the implementation in industrial automation. The estimation of using a wireless network can save up to 90% compared to the deployment cost of wired field devices. Since the discrete product of the factory automation requires sophisticated operations of robot and belt conveyors at high speed, the sampling rates and real-time requirements are often stricter than those of process automation. Furthermore, many industrial automation applications might in the future require battery-operated networks of hundreds of sensors and actuators communicating with access points. (Park, Ergen, Fischione, Lu, & Johansson, 2017)

1.3 Stating the problems

This paper consists of two major subjects: communications and control. These subjects inherently have a few problems that need to be held in account.

1.3.1 Communications

Latency or message delay is a factor that needs to be measured and researched. If the latency is too big, the controller cannot function properly and will limit the responsiveness. This is something that is inherent to wireless communications and cannot be solved completely. But it can be measured and a conclusion can be drawn to determine if wireless communication is a valid option for certain industrial processes.

Package loss is another important factor that needs to held in account. Valuable data used to control the process will get lost and will cause the controller to be an unreliable part of the industrial process. This loss of data can be caused by interference with other wireless communications or due to loss of connection.

1.3.2 Control

Defining and finetuning the control parameters is a challenge because the delay in communications needs to be taken into account. For the experiments conducted in this report we intentionally chose an easy process to control but in reality the industrial processes that this technology can be used in, is much more complex.

Communication also plays a big part in controlling a system. For every problem that is being solved in the communication it must be validated that this solution does not make the control aspect of the system worse or slower.

1.4 Project goals

The main goal of this project is to set up a complete wireless control loop with a PI controller, sensor and end effector.

The secondary goals are :

- Implement a “myRio” device, to allow the system to function independently from the computer. This makes the set-up representative for a real industrial system.
- Research the characteristics of the wireless network and optimize them as far as possible. Reduce the loss of data and increase responsiveness.
- Simplify the system by only using one antenna for sending and receiving instead of two antennas.

2 Description of the experimental setup

In order to test the wireless communication and wireless control loop an experimental setup is needed. Three XBee antennas, an Arduino microcontroller, a computer and various electronic components are used in this setup. In Figure 1 the layout is shown in which these components are configured. The individual components are explained in the next chapters.

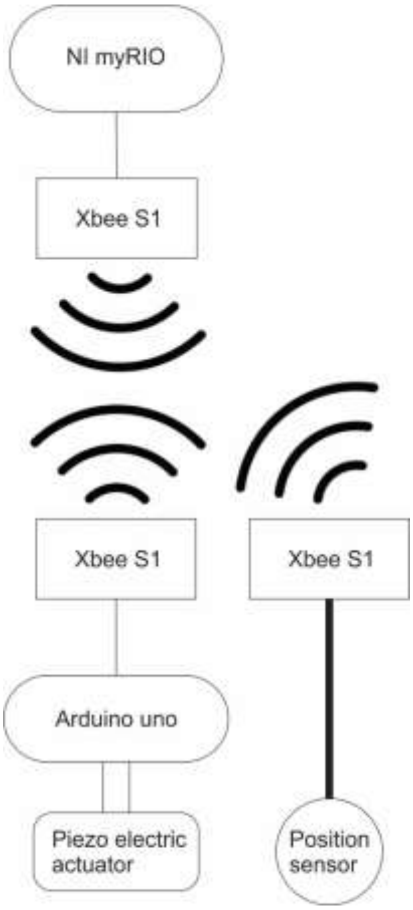


Figure 1: Specific layout of the experimental setup

2.1 Components

2.1.1 Arduino microcontroller

An Arduino Uno microcontroller is used to interact with the physical world. Its primary use is to generate a PWM signal in order to control the end effector. This kind of microcontroller is chosen because it is easy to program and cheap to buy. There also is a lot of support for different accessories, for instance the XBee antenna shield explained in the next chapter.

Figure 2: Arduino Uno microcontroller

2.1.2 XBee shield

This XBee shield is placed onto the Arduino Uno so that it is able to send and receive wireless data through an XBee antenna. The analog and digital pins are transferred through the XBee shield so that these can still be used. It is also equipped with a switch so that you can easily switch to receive data through the serial port on the Arduino (to upload your program) or through the XBee antenna.

Figure 3: XBee shield for the Arduino

2.1.3 Controller

This is simply a computer that runs a LabVIEW program. In this program the PI controller will be configured and a few tests will be able to run on here. It also includes all the necessary code to save data, the code to send and receive data to and from the XBee antennas. In Figure 4 an example LabVIEW program is shown, this is a program that is used in this project to send data through the XBee antenna.

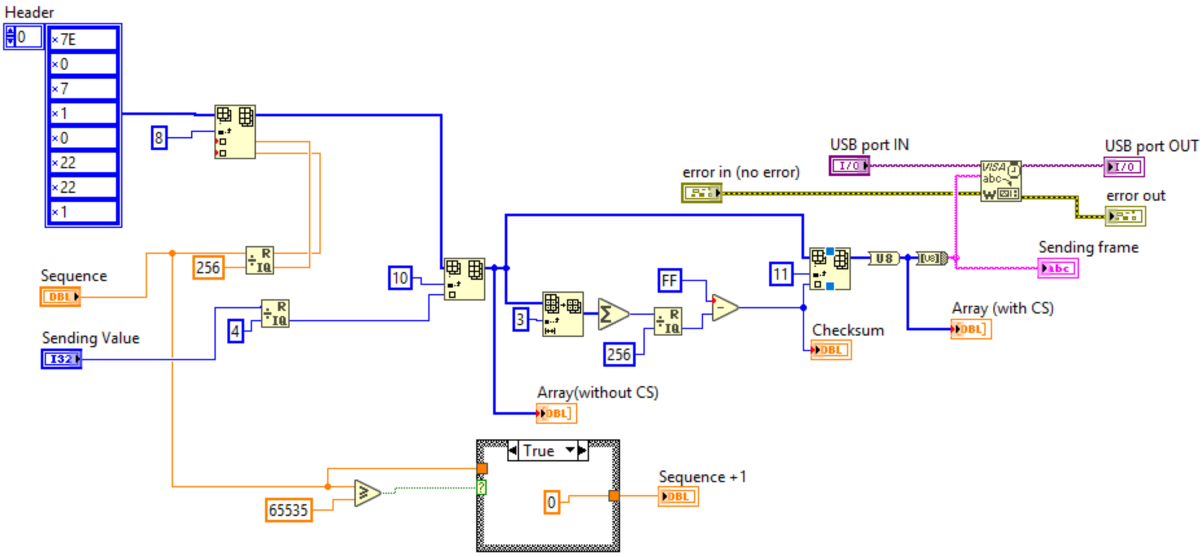


Figure 4: Example of LabVIEW program running in the controller

2.1.4 XBee antennas

These antennas are merely used to transfer data from one point to the next wirelessly. The XBee S1 version were chosen for this project. These antennas have a range of 30 meters indoors and 100 meters outdoors. This range is more than enough for the tests conducted in this paper.

Three XBee antennas are used in the experimental setup to make it easier to configure, later on two XBee antennas could be used. This can be achieved by eliminating the XBee antenna of the sensor and using the antenna on the microcontroller for both sending and receiving data.

Figure 5: XBee S1 antenna

2.1.5 XBee USB explorer

In order to configure all the parameters for the XBee antennas, a serial connection needs to be made to a computer running the XCTU software. The USB explorer is outfitted with a micro USB port which allows easy connection to the computer.

In the XCTU software various parameters can be changed about the XBee. For instance: Destination address, Source address, Baud rate, I/O settings, etc. This program and the configuration of the XBee antennas will be explained in a later chapter.

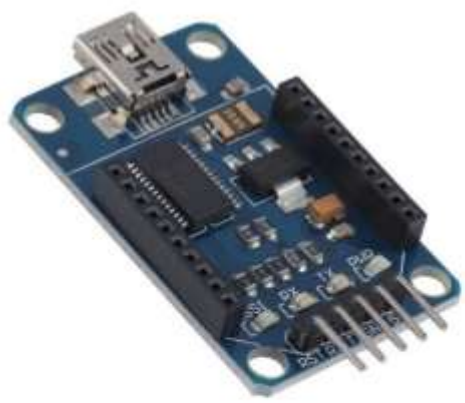
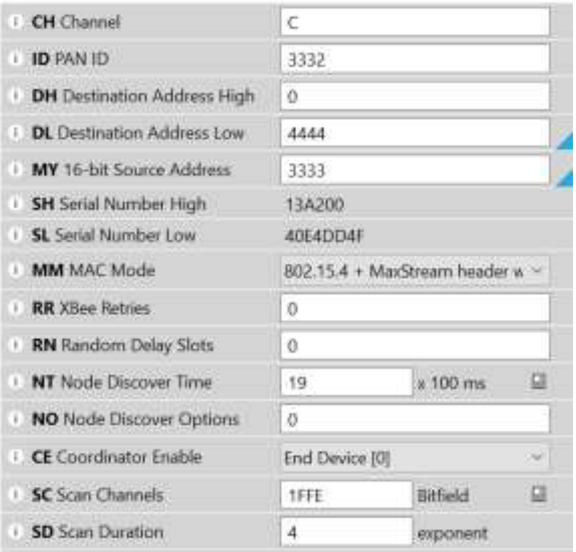


Figure 7: XBee USB explorer



CH Channel	C	
ID PAN ID	3332	
DH Destination Address High	0	
DL Destination Address Low	4444	
MY 16-bit Source Address	3333	
SH Serial Number High	13A200	
SL Serial Number Low	40E4DD4F	
MM MAC Mode	802.15.4 + MaxStream header w	
RR XBee Retries	0	
RN Random Delay Slots	0	
NT Node Discover Time	19	x 100 ms
NO Node Discover Options	0	
CE Coordinator Enable	End Device [0]	
SC Scan Channels	1FFE	Bitfield
SD Scan Duration	4	exponent

Figure 6: Example of parameters within XCTU

2.1.6 End effector

The end effector used in this setup is a piezoelectric actuator outfitted with a position sensor. This actuator is a good representation of an actual end effector used in industrial processes. The built in position sensor is an added bonus as this makes it easier to build a closed control loop. This actuator however has proven to be very sensitive to impulses so care must be taken when working with it.

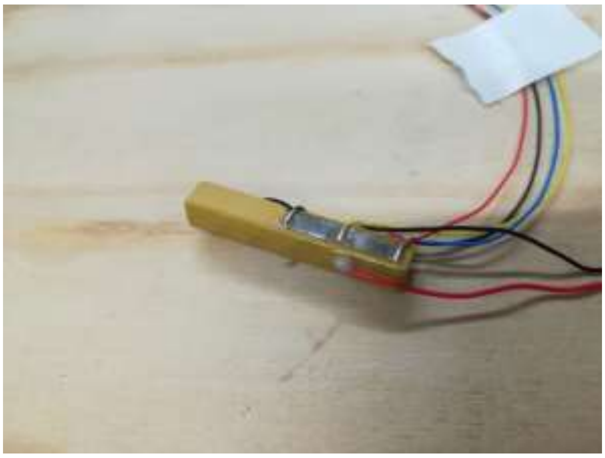


Figure 8: Piezo electric actuator

2.1.6.1 Position sensor

The position sensor discussed earlier has a range from 0 V – 10 V. This range needs to be converted to a range of 0 V – 3,3 V. This has to be done because the analog input of the XBee antenna only allows a range of 0 V – 3,3 V. This conversion is done by a simple voltage divider with three resistors.

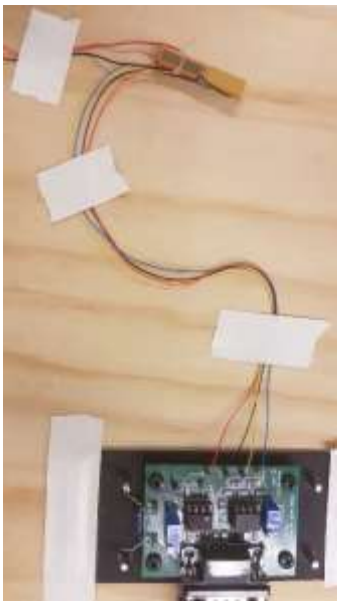


Figure 9: Amplifier and position sensor

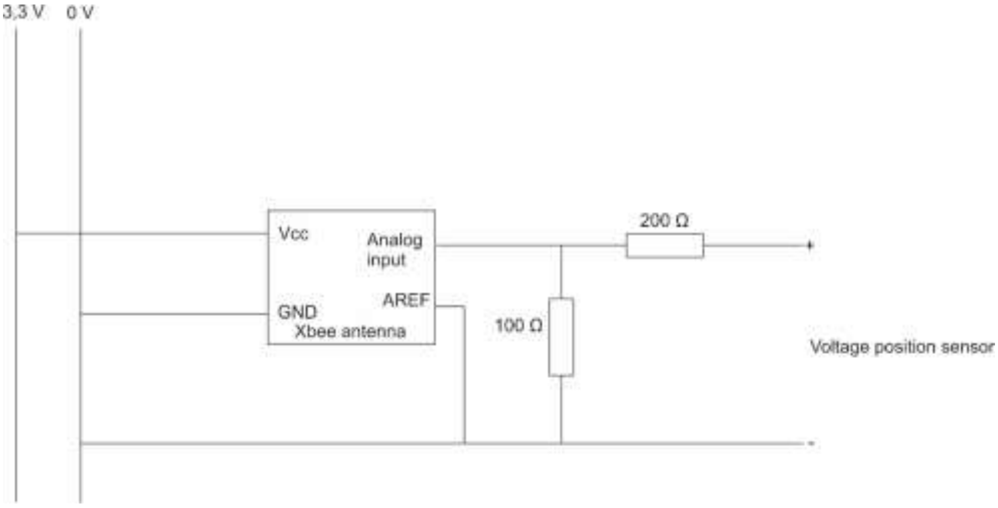


Figure 10: Position sensor circuit

2.1.7 Transistor circuit

In order to convert the PWM signal that is provided by the Arduino microcontroller a transistor circuit is built. This circuit will transform the PWM signal into an analog signal with a range of 0,7 V – 10 V. The range starts at 0,7 V because of the characteristics of the transistor. A transistor needs this voltage over the base and emitter, so that it can allow current to pass from the collector to the emitter.

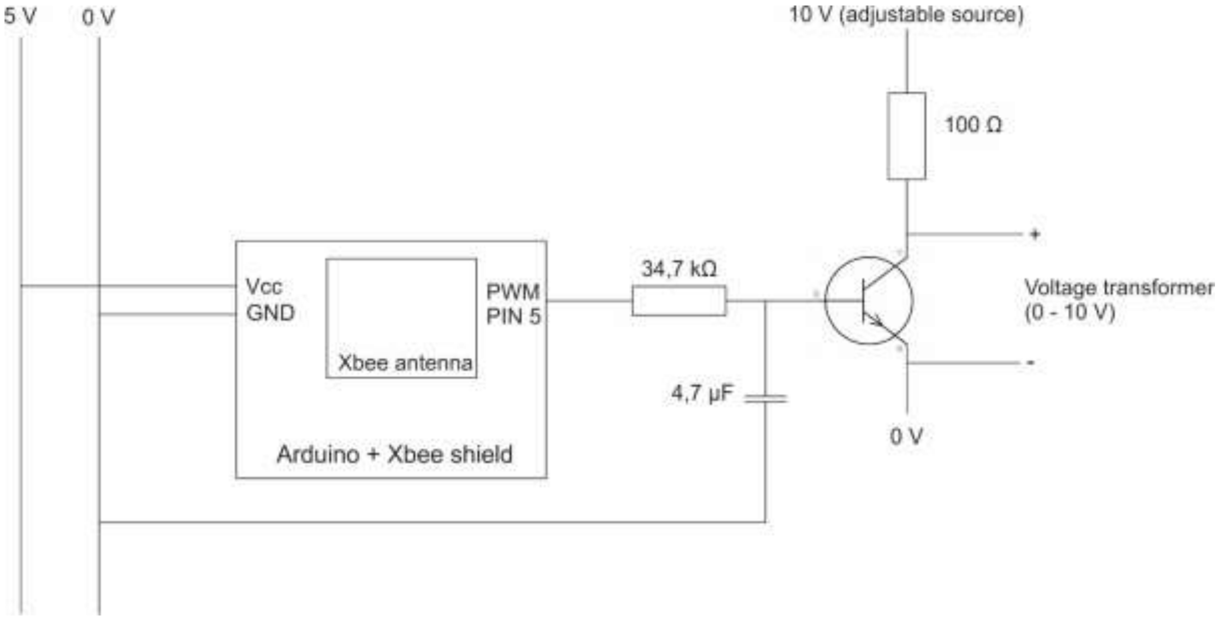


Figure 11: Transistor circuit

The transistor that is used in this circuit is vulnerable to overdrawing current. Care must be taken in order to ensure that not too much current is provided to the collector, emitter or base as this will lead to failure of this part.

2.1.8 Power transformer

The piezo electric actuator uses a range of 0 V – 150 V, therefore a power transformer was needed. This power transformer will change the input range of 0,7 V – 10 V provided by the transistor circuit to the needed 0 V – 150 V.

These transformers can be zeroed and the out channel can be enabled and disabled. On the display the current output voltage can also be read. This allows for greater flexibility and an extra method to check what voltage is being provided to the actuator.

Care must be taken when applying the voltage, the voltage over the actuator should never be greater than 150 V as this can cause damage to it. Therefore it was decided that the voltage was limited to 145 V, just to be safe. The transformers are displayed in Figure 12.

On top is the transformer for the actuator. This has one high voltage output that goes to the actuator and one external input which is connected to the transistor circuit.

On the bottom is a transformer for the sensor. This has one input that is connected to an amplifier which is connected to the sensor. This also has one output which is a 0 V – 10 V scale that goes to the circuit discussed in chapter 2.1.6.1.

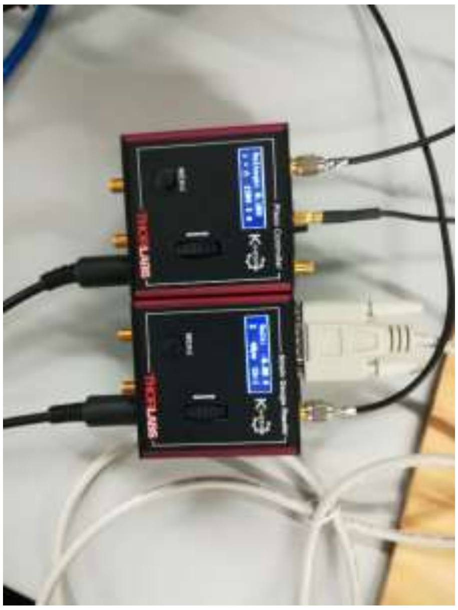


Figure 12: Transformers

2.1.9 myRIO embedded device

In order to make our LabVIEW programs run autonomously a myRIO embedded device was used. Onto this device programs can be uploaded and it has built in memory to save the gathered data. To this device the XBee antenna was connected via the USB explorer and a micro USB cable as displayed in Figure 13. The cable to the left is the power cable and the cable in the middle is the USB cable used to download the program onto the device.

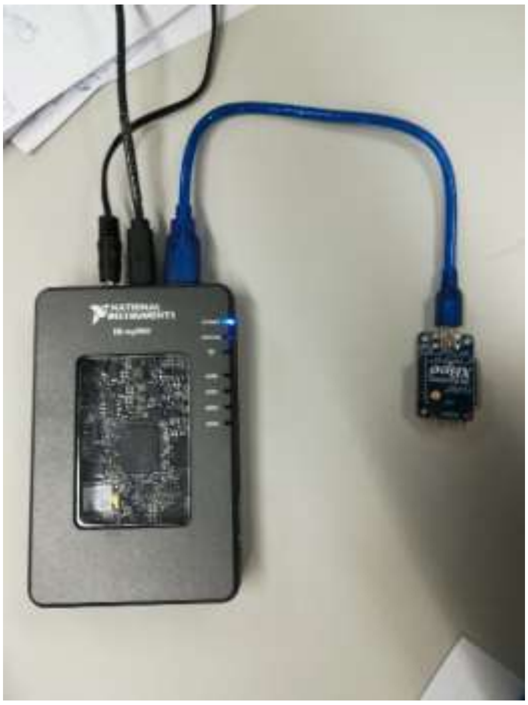
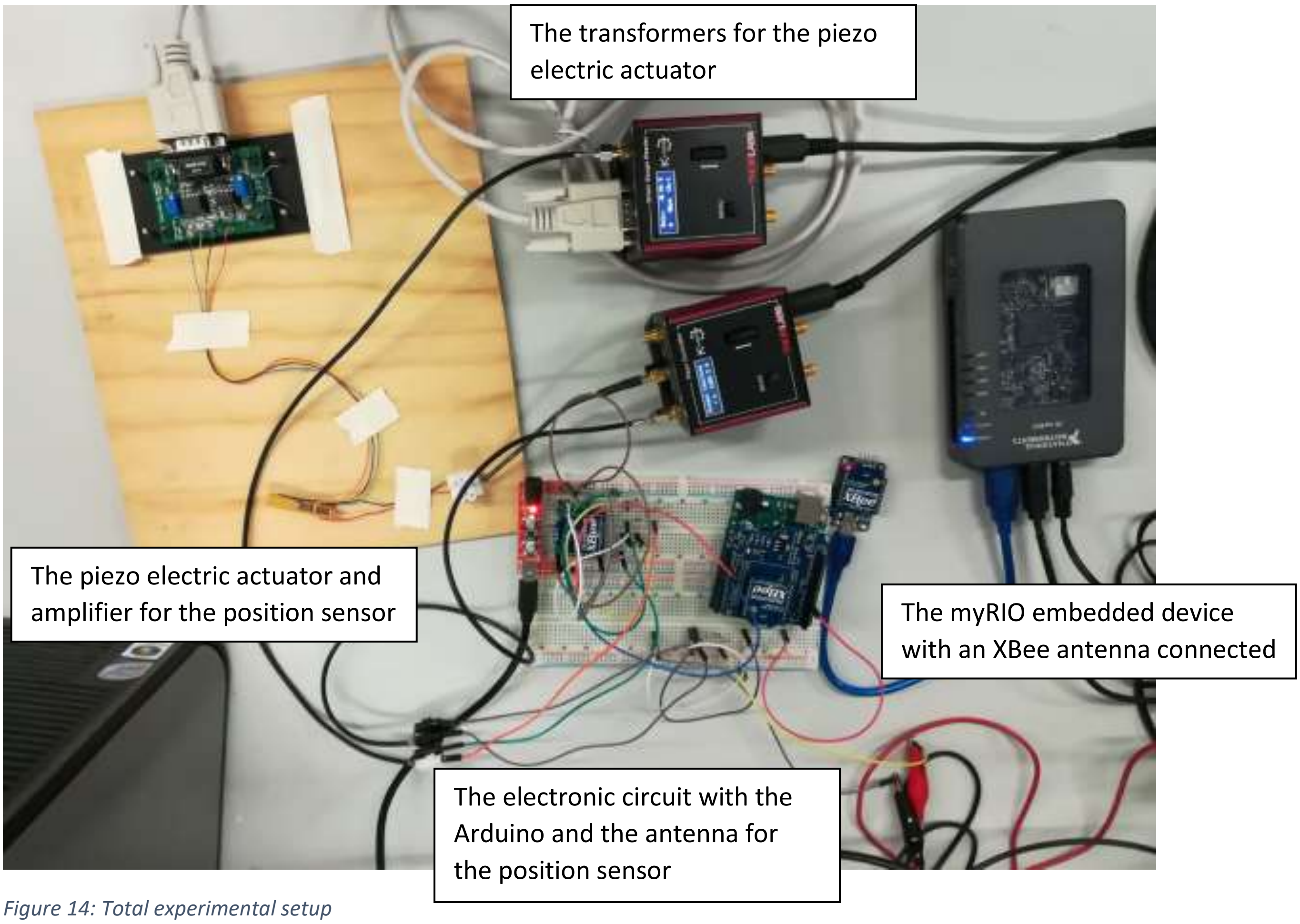


Figure 13: myRIO embedded device

2.2 Total experimental setup

In the figure below the actual experimental setup is shown. All the components discussed in the previous chapter are utilised in this setup. This setup is the most complex version, this could be simplified by only using one XBee antenna to send and receive the information to and from the piezoelectric actuator.

The piezo electric actuator and amplifier are attached to a wooden board because these are very fragile. The cables are of a small diameter and are prone to breaking. On the next page a schematic overview is displayed that shows the individual parts more clearly.



The transformers for the piezo electric actuator

The piezo electric actuator and amplifier for the position sensor

The myRIO embedded device with an XBee antenna connected

The electronic circuit with the Arduino and the antenna for the position sensor

Figure 14: Total experimental setup

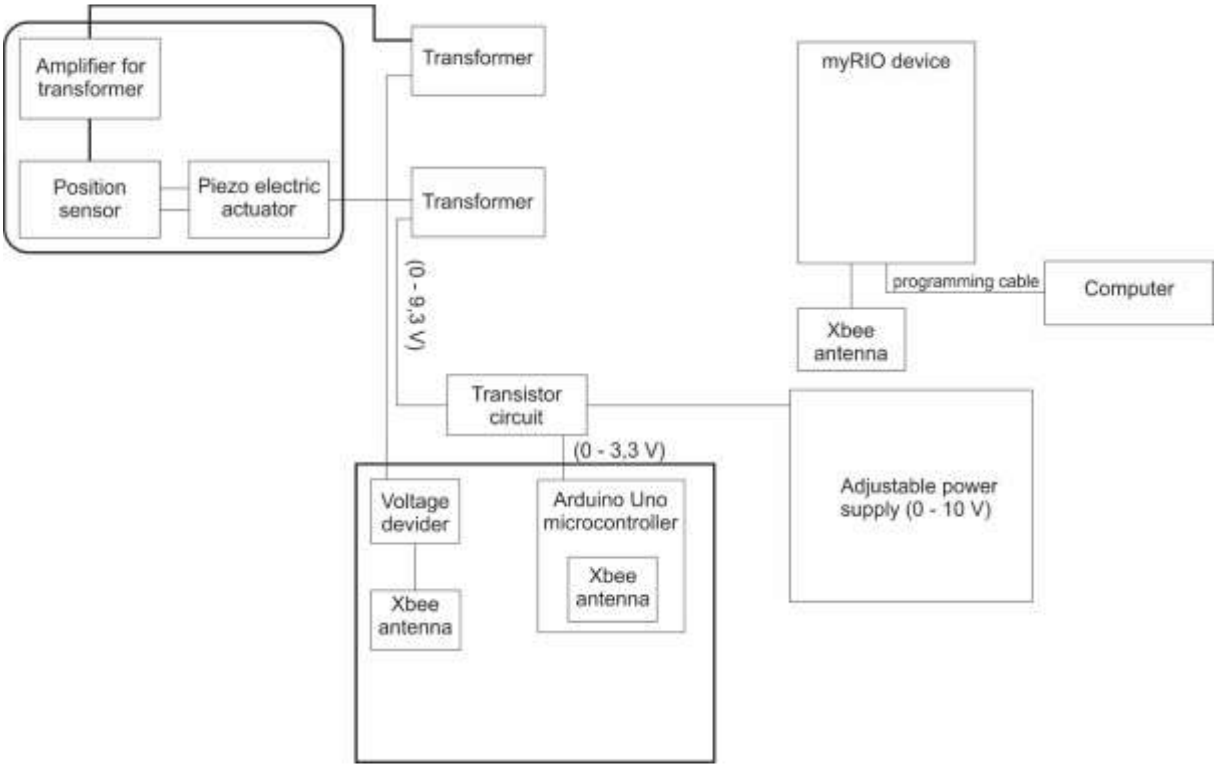


Figure 15: Schematic overview of the total setup

2.3 First tests

In the first test the end effector and sensor were replaced by simpler components. This was done to simply test the communication and to understand how XBee sends and receives data. The simplified setup can be seen in Figure 16. This made it possible to check if the XBee antennas were properly configured for future tests.

Here a button was used to send data to the controller and an LED was used to receive data from the controller. By using the XCTU software the receiving data could be monitored, so it was possible to see what the data included and if this made sense. Via the same software we could send a command to the XBee antenna to set an output to “HIGH”. This then lighted up the LED, and now it was certain that the XBee antenna was configured correctly to both send and receive data.

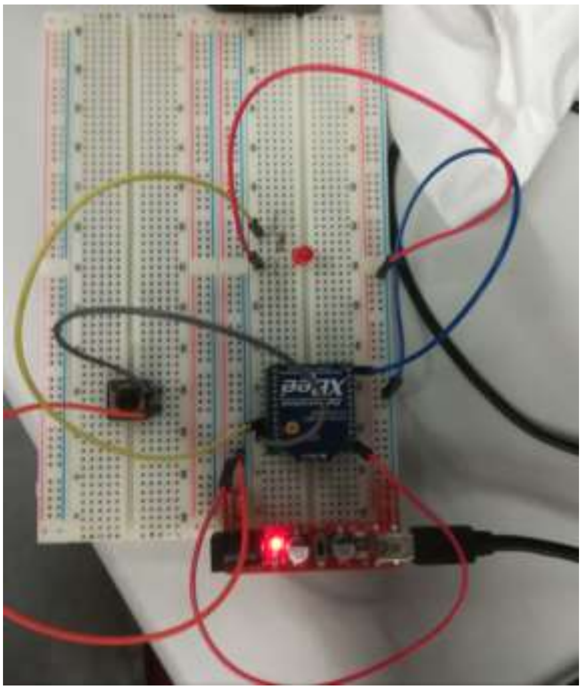


Figure 16: Simplified setup

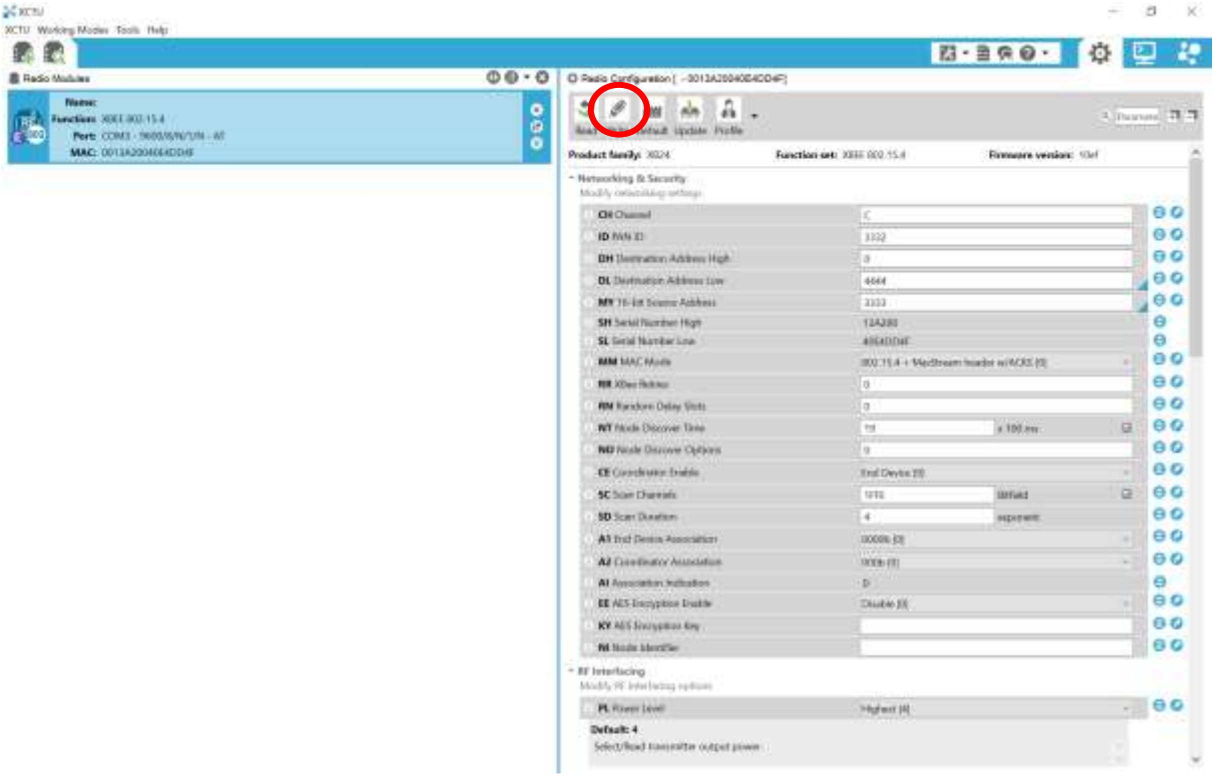
3 Configuring XBee antennas with XCTU

3.1 General explanation of how the software works

XCTU is a program that can be used to change the parameters that define the XBee antennas. With this you can program which XBee antennas connect to each other, which ones send and which ones receive data. In this program you can change a large list of parameters, too large to list them all, but here are some of the most important ones:

- Channel: Choose which RF channel to use.
- PAN ID: Personal Area Network ID, Antennas can be grouped into different networks. This allows for more flexibility.
- Destination address: The address to which data will be sent.
- Source address: The address of the XBee antenna. In order for data to be received by this Xbee, it must be sent to this address.
- Power level: the transmitter output power can be set.
- I/O settings: here various digital and analog I/O can be set, as well as PWM configurations
- Baud rate: the baud rate can be configured to various standard values
- Etc…

When the program is first opened the XBee antennas that are connected to the computer need to be discovered. After this is done the found antennas are displayed to the left of the screen, on the right you will find the parameters currently downloaded into the antenna. The parameters can be directly altered in this window. Once the changes are made press the write button (highlighted with a red circle on Figure 17) and the new parameters are downloaded to the antenna.



Name:
Function: XBEE 802.15.4
Port: COM3 - 9600/8/N/1/N - AT
MAC: 0013A20040E4DD4F

Radio Configuration [- 0013A20040E4DD4F]

Product family: XB24 Function set: XBEE 802.15.4 Firmware version: 10ef

Networking & Security
Modify networking settings

CH Channel	C
ID PAN ID	3332
DH Destination Address High	0
DL Destination Address Low	4444
MY 16-bit Source Address	3333
SH Serial Number High	13A200
SL Serial Number Low	40E4DD4F
MM MAC Mode	802.15.4 + MaxStream header w/ACKS [0]
RR XBee Retries	0
RN Random Delay Slots	0
NT Node Discover Time	19 x 100 ms
NO Node Discover Options	0
CE Coordinator Enable	End Device [0]
SC Scan Channels	1FFE Bitfield
SD Scan Duration	4 exponent
A1 End Device Association	0000b [0]
A2 Coordinator Association	000b [0]
AI Association Indication	0
EE AES Encryption Enable	Disable [0]
KY AES Encryption Key	
NI Node Identifier	

RF Interfacing
Modify RF interfacing options

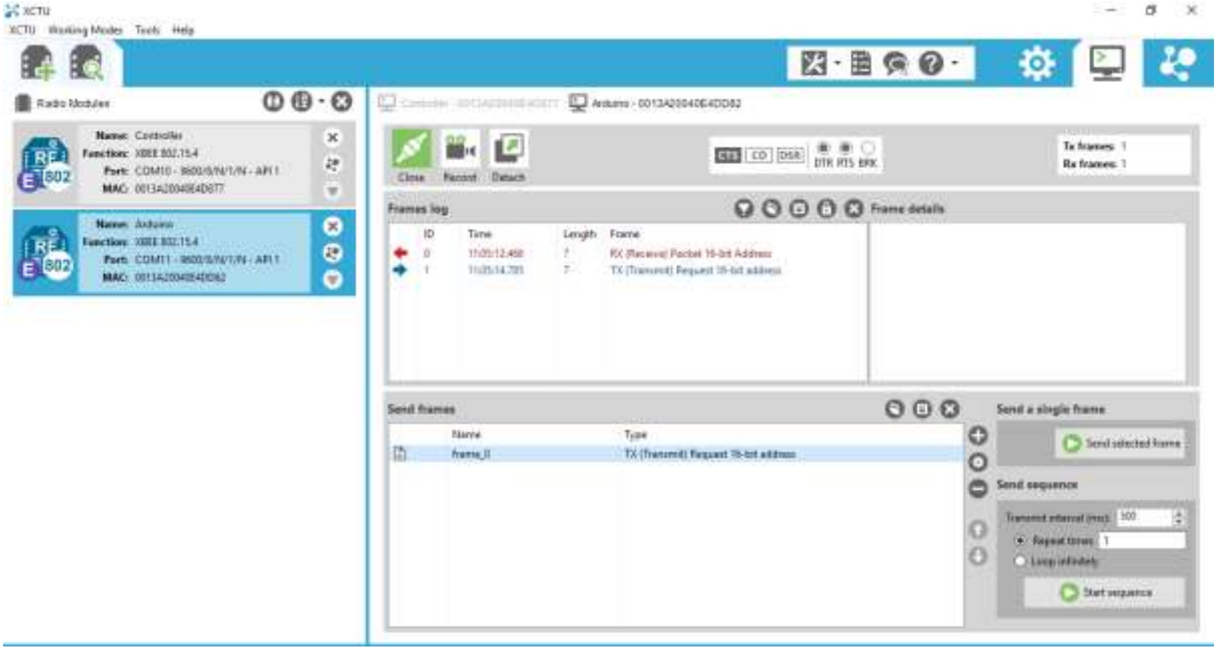
PL Power Level	Highest [4]

Default: 4
Select/Read transmitter output power.

Figure 17: XCTU configuration

This program also features a console in which you can track all the serial communications that happen wirelessly, which is located on the top right of the screen, it is the icon in the middle. Data that is being sent is presented in blue and received data is presented in red. At the bottom of the screen custom data packages or “frames” can be created to send from one XBee to another. How these frames are formed will be discussed in a later chapter.

A quick way to test if two XBee antennas are properly connected to each other, is to connect them both to a separate USB port. Discovering them both with the XCTU software and then utilising the console to send one package from Xbee A to XBee B. This way one can see if XBee A sends the data properly and Xbee B receives it properly. Repeat the other way around and one can be assured that both antennas are connected properly.



Name: Controller
Function: XBEE 802.15.4
Port: COM10 - 9600/8/N/1/N - API 1
MAC: 0013A20040E4D877

Name: Arduino
Function: XBEE 802.15.4
Port: COM11 - 9600/8/N/1/N - API 1
MAC: 0013A20040E4DD82

Controller - 0013A20040E4D877 · Arduino - 0013A20040E4DD82

Tx frames: 1
Rx frames: 1

ID	Time	Length	Frame
0	11:05:12.458	7	RX (Receive) Packet 16-bit Address
1	11:05:14.781	7	TX (Transmit) Request 16-bit address

Name	Type
frame_0	TX (Transmit) Request 16-bit address

Transmit interval (ms): 500
● Repeat times: 1
○ Loop infinitely

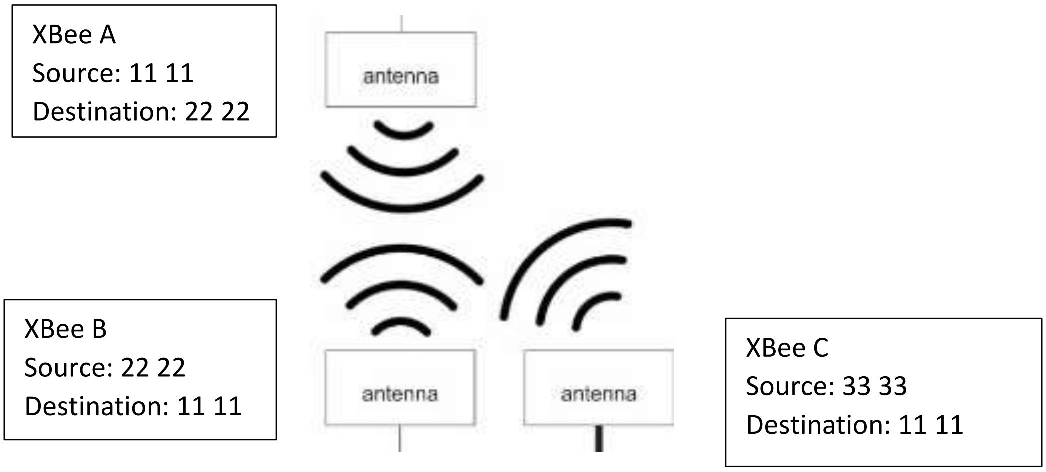
Figure 18: XCTU console

3.2 Configuration for the experimental setup

In our experimental setup, as explained in chapter 2.1.4, three XBee antennas are used. These antennas need to be configured correctly in order to ensure communication between them.

Firstly all three need to be part of the same Personal Area Network, in our case “3332” is chosen which is the default value. After this we need to make sure they all have a different source address. These addresses need to be unique otherwise the wrong XBee will receive the wrong data. Next we need to define the destination addresses, 16-bit addresses are chosen because in our experimental setup not a lot of XBee antennas are used, so 16-bit suffices.

This requires some planning, the right sending antenna needs to be coupled with the right receiving antenna and vice versa. To explain this easier Figure 19 shows the three antennas that are being used in the experimental setup and how they communicate.


XBee A
Source: 11 11
Destination: 22 22

XBee B
Source: 22 22
Destination: 11 11

XBee C
Source: 33 33
Destination: 11 11

Figure 19: Explanation communication with experimental setup

XBee A is connected to the controller, B is connected to the Arduino microcontroller and C is connected to the sensor.

As you can see XBee A sends data to XBee B. B only needs to receive data so the destination address doesn’t matter, but 11 11 is chosen just in case B ever needs to send any data back to the controller. C sends back to A. Therefore the destination address of C needs to be 11 11, the source address of A. The destination address of A needs to be 22 22, the source address of B.

After the destination and source addresses have been correctly configured, the baud rate of all XBee antennas must be configured to be the same. This is important because if two devices are communicating at the same speed, data can be misinterpreted or deemed completely wrong and dismissed (Jimblom, sd). For this setup the standard baud rate is chosen, which is 9600.

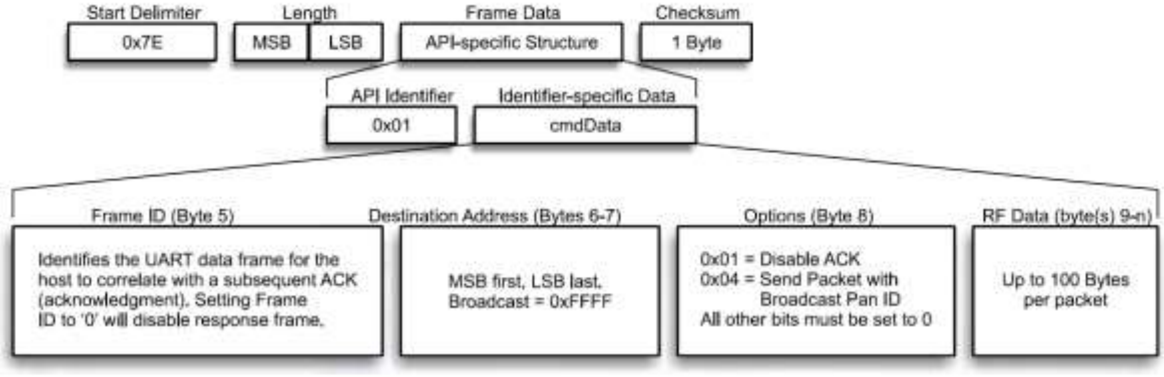
Lastly the I/O needs to be configured according to purpose of each antenna. XBee A does not have to control any I/O so these can remain disabled. Because XBee B is connected to the Arduino, no I/O configurations need to be made. These will be done in the Arduino code. XBee C needs an analog input in order to read the sensor value, so this is configured.

4 Frame

The data we send and receive via XBee is formatted into a "frame". This is a standard format in which every byte has a meaning. This makes it easier to understand where the data starts and ends. In a frame all the bytes are a hexadecimal value.

In Figure 20 the frame structure is displayed. It consists of a few different elements and each has a meaning.

- Start delimiter: this byte is always "0x7E" so that it is clear when the data frame starts.
- Length: these two bytes indicate the length from the next byte up until the second to last byte.
- Frame data: a few different things are indicated here, the API identifier, frame ID, Destination address, extra options and RF data.
    - API identifier: This indicates the type of frame, for instance: sending frame, receiving frame,…
    - Frame ID: This is only used when an acknowledgment is required, for these experiments we do not require this so this byte is set at "0x00".
    - Destination address: this indicates to which address or from which address the data is being send/received.
    - Options: Different options can be chosen with this byte. For these experiments acknowledgements need to be disabled so this byte is set to"0x01".
    - RF data: This is the actual data that is being transferred from one XBee to the next. This can be anywhere from 0 to 100 bytes, and also includes a checksum at the end to check if the data is correct.

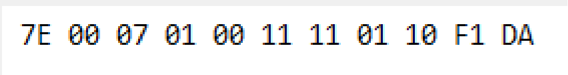


Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x01
Identifier-specific Data
cmdData
Frame ID (Byte 5)
Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgment). Setting Frame ID to '0' will disable response frame.
Destination Address (Bytes 6-7)
MSB first, LSB last. Broadcast = 0xFFFF
Options (Byte 8)
0x01 = Disable ACK
0x04 = Send Packet with Broadcast Pan ID
All other bits must be set to 0
RF Data (byte(s) 9-n)
Up to 100 Bytes per packet

Figure 20: General frame structure

4.1 Sending frame

As previously discussed frames have different API identifiers, these indicate which kind of frame is created. For a frame that is designed to send information the identifier is set to “0x01”. This indicates a TX request for a 16-bit address. Figure 21 below shows what a complete frame looks like in XCTU.



```
7E 00 07 01 00 11 11 01 10 F1 DA
```

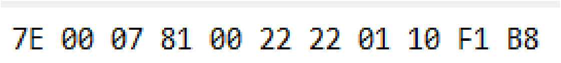
Figure 21: Sending frame

In this frame we can differentiate all the previously discussed elements.

- 7E : start delimiter
- 00 07: the frame is 7 bytes long from the next byte until the second to last byte
- 01: significates a send request for a 16-bit address
- 00: no response frame needed, no acknowledgement
- 11 11: destination address
- 01: the option for disabling acknowledgements is selected
- 10 F1: the data that is being send
- DA: the calculated checksum

4.2 Received frame

A frame that is being received will have the identifier set to “0x81”. This indicates a RX request from a 16-bit address.



```
7E 00 07 81 00 22 22 01 10 F1 B8
```

Figure 22: Received frame

- 7E : start delimiter
- 00 07: the frame is 7 bytes long from the next byte until the second to last byte
- 81: significates a receiving request for a 16-bit address
- 00: no response frame needed, no acknowledgement
- 22 22: source address
- 01: the option for disabling acknowledgements is selected
- 10 F1: the data that is being send
- B8: the calculated checksum

The data that is being sent is not limited to just the control / measured value. For some tests a sequence number is included in these data bytes. This made it easier to see when which frames get sent / received.

There are various different types of frames you can send via XBee antennas but these are too many to list here. The frames discussed previously are the ones used in these experiments.

Also for the experiments a sequence number was added to the data before the checksum “F1” in the examples given previously. This one byte of data is import to know when the frame was sent / received.

5 Explanation of the LabVIEW program

To explain the program easier, it is separated in five parts: creating signal (1), Xbee sending (2), PID (3), Xbee receiving (4) and saving (5). In Figure 23 the total LabVIEW program and the part described below are displayed. For a global overview of this program please take a look at the appendices at the end of this paper.

Every part will be explained in a separate paragraph. The code will be shown and a short text will explain how it works. For the communication tests (Chapter 8 Communication) the code was changed in some aspects. These changes will be explained in the appropriate chapter.

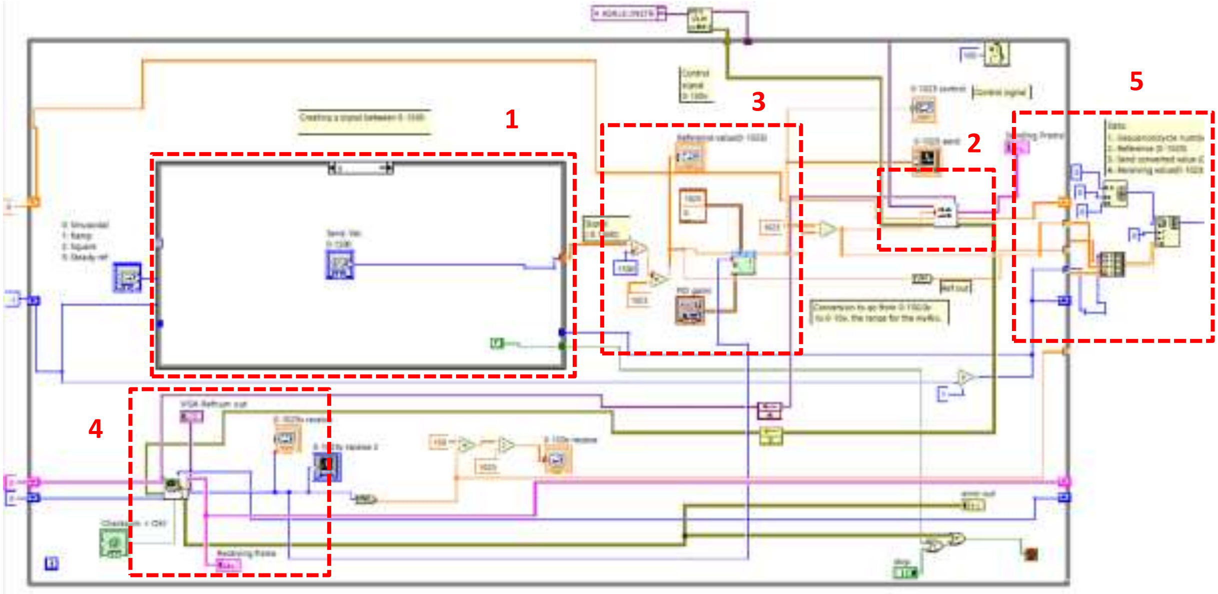


Figure 23: Total LabVIEW program

5.1 Creating signal

The user can choose from four different input signals: sinusoidal, ramp, square and a steady reference. Each of these signals will explained in the following chapters. The user can change these signals by using a case selector block, displayed in Figure 24.

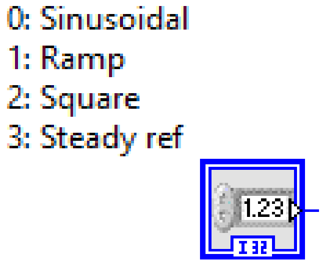


Figure 24: Case selection of input signals

5.1.1 Sinusoidal input signal

For the first method three different sinewaves will be formed, where the maximum will be decreased every sinewave.

So the signal will be a sinus from 0 to 1023 and back to 0 after that the signal will go to 680 and back to 0 and the last sinus will go to 340 and also back to 0. Every sinus will be 1000 samples long, so when the loops reaches 3000 sinus the test will stop.

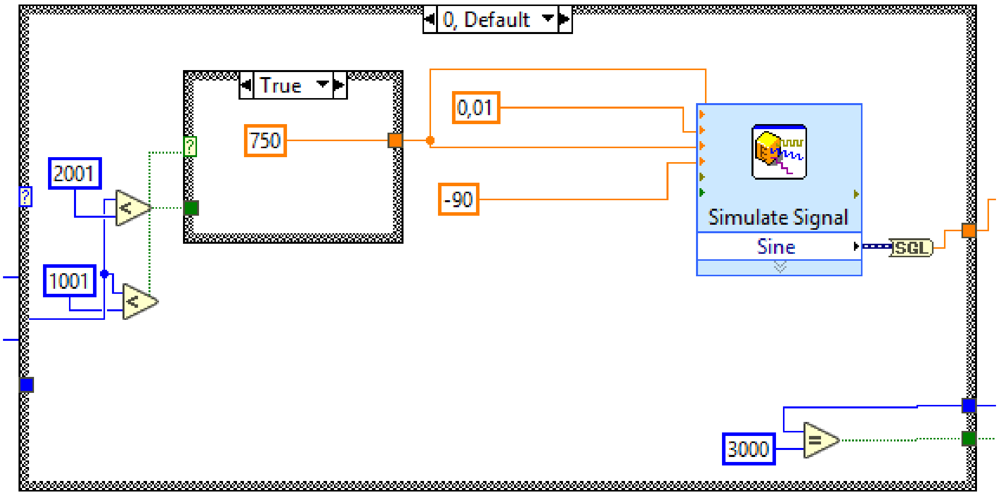


Figure 25: Block diagram of the sinusoidal input signal

Figure 26 shows an example of what the sinusoidal input signal will look like.

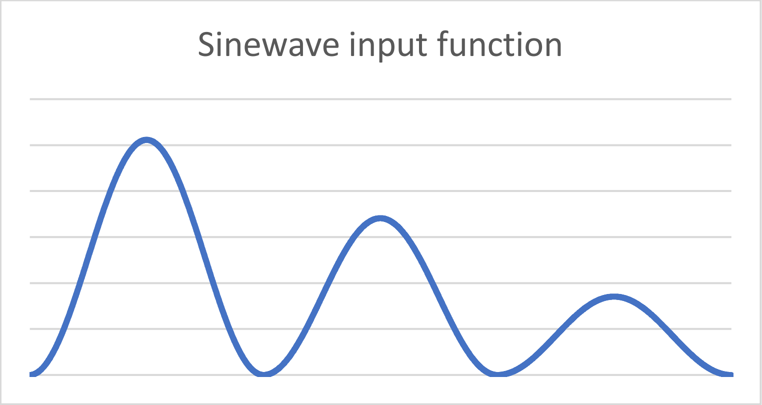


Figure 26: example of the sinewave input signal

5.1.2 Ramp input signal

The next method is a simpler one, a ramp function that increases the value by one every time the main loop of the LabVIEW program is executed. The ramp function is 3000 samples in length. The ramp will reach its peak at 1500 samples and will go back down to zero.

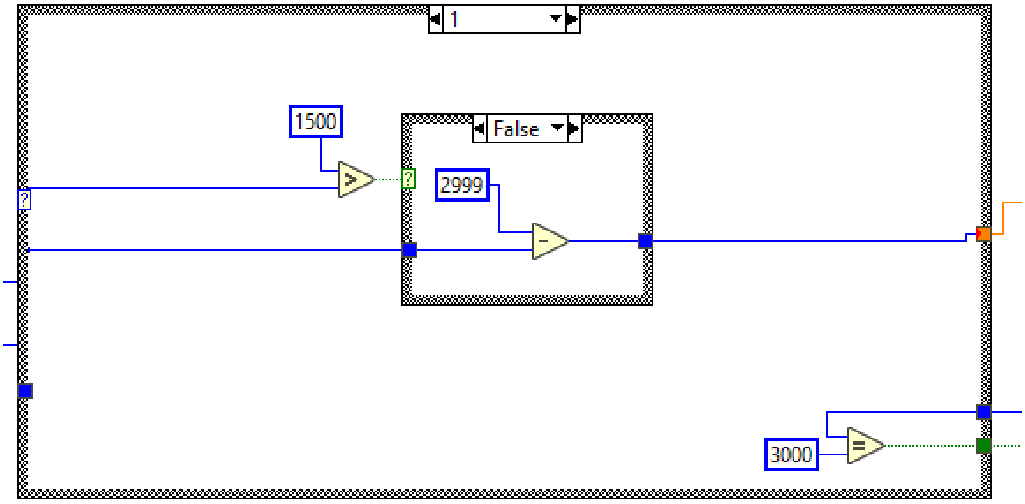


Figure 27: Block diagram of the ramp input signal

Figure 28 shows an example of what the ramp input signal will look like.

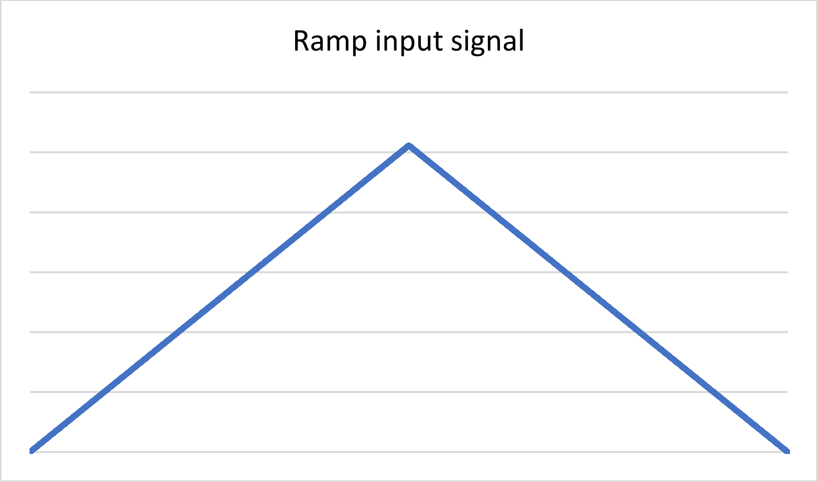


Figure 28: Example of the ramp input signal

5.1.3 Square input signal

This signal will send a maximum value of 1500 for 100 samples and then a minimum value of 0 for that same amount of samples. A start offset of 50 samples is chosen. The total duration of this signal is 500 samples.

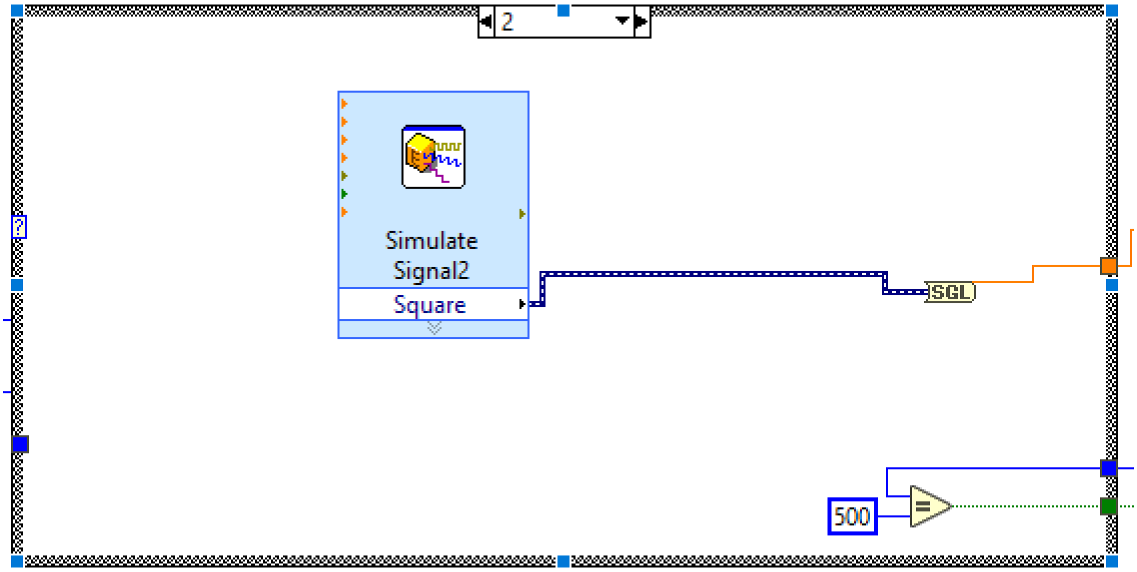


Figure 29: Block diagram of the square input signal

Figure 30Figure 28 shows an example of what the square input signal will look like.

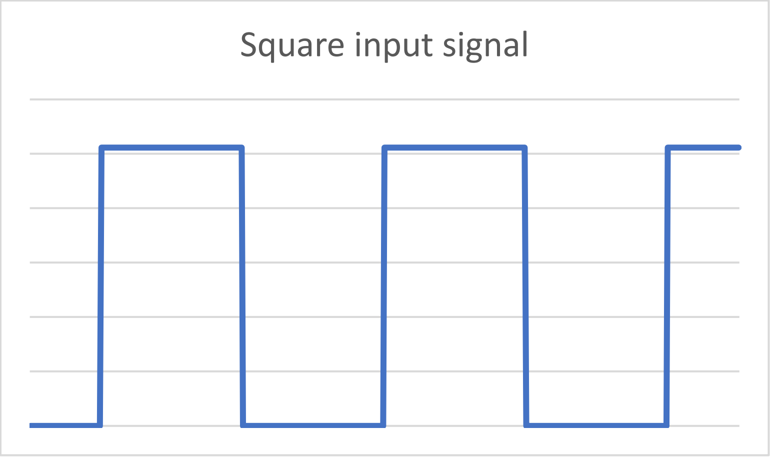


Figure 30: Example of the square input signal

5.1.4 Steady reference input signal

This signal allows the user to send a constant value as an input signal. This signal has no fixed length and the value can be changed as the user pleases.

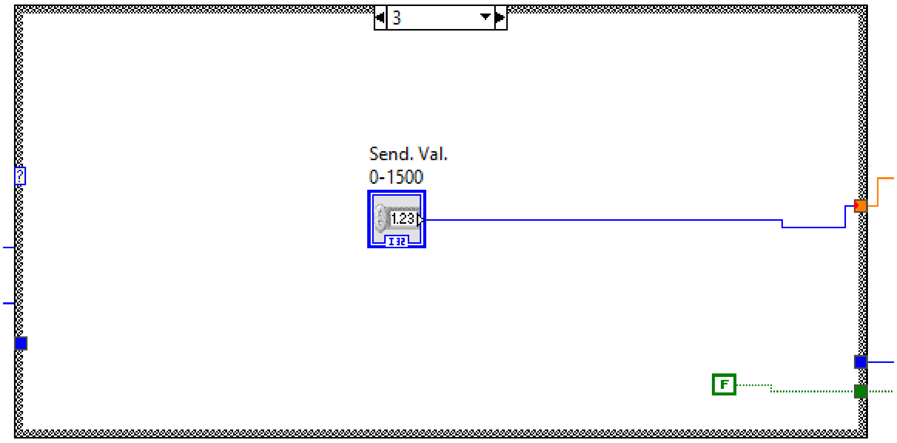
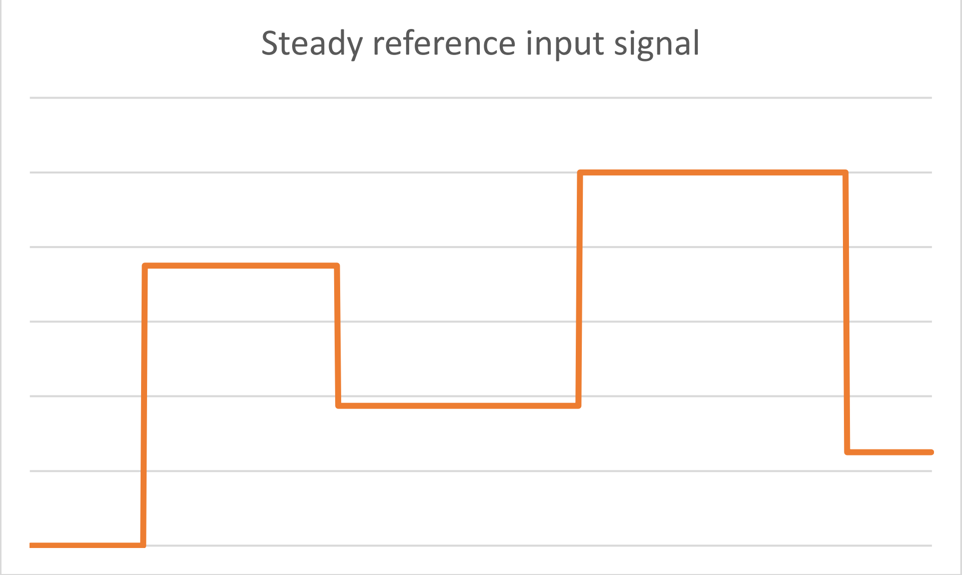


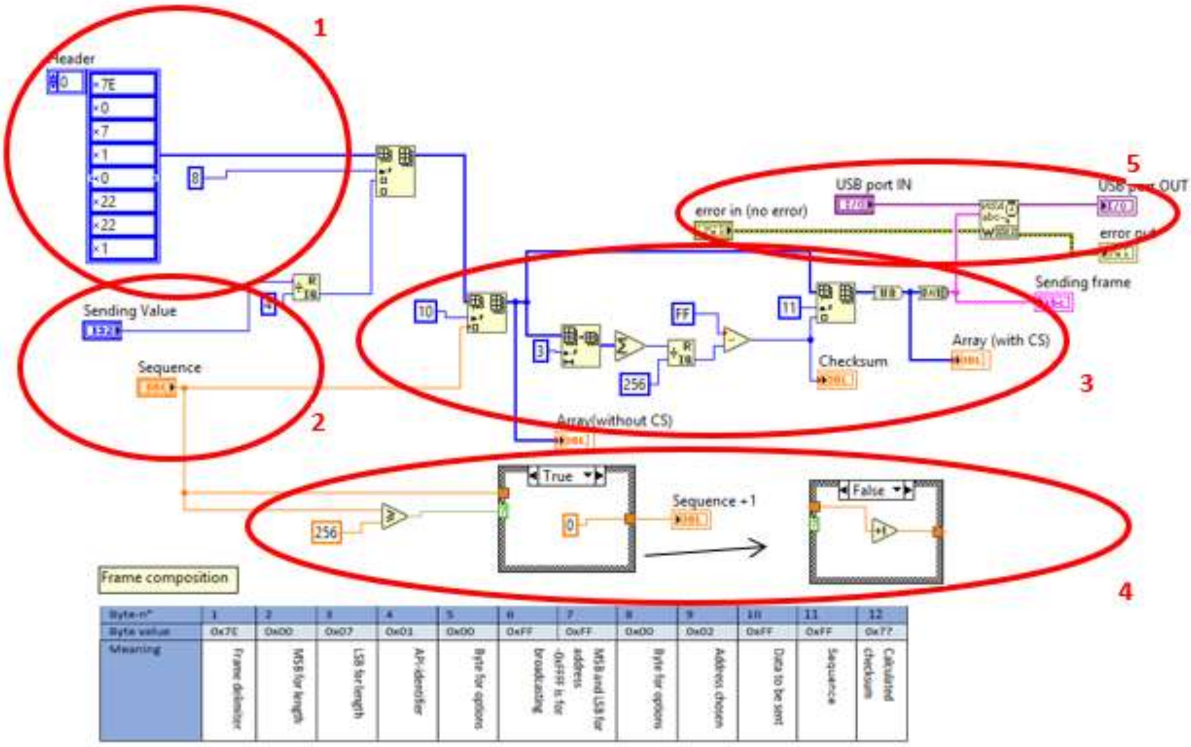
Figure 31: Block diagram of the steady reference input signal

shows an example of what the square input signal will look like.



5.2 Xbee sending

This VI is used to create a frame that store the data that needs to be sent to the actuator. More information on how these frames are configured can be found in chapter 4.



Byte-n°	1	2	3	4	5	6	7	8	9	10	11	12
Byte value	0x7E	0x00	0x07	0x01	0x00	0xFF	0xFF	0x00	0x02	0xFF	0xFF	0x77
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	Sequence	Calculated checksum

Figure 32: The XBee send VI

The VI is consists of 5 parts. In the first part the constant frame parameters are formed (indicated by a red 1 in the top left corner of Figure 32). At the bottom of the figure you can see how a frame is composed. These constant parameters are saved in an array and after these, the data bytes and checksum is added.

The second part is where the data gets transformed to 8 bit or 1 byte of data. The PWM of the Arduino can only read an 8 bit signal, therefore the 10 bit signal needs to be converted to an 8 bit signal. The next byte of data is used for the sequence number and this gets set to the eight position in the frame.

In the third part the checksum is formed. At the end of every frame there needs to be a checksum. This is calculated by the sum of the array without the start delimiter and the frame length. This number is then subtracted from the hexadecimal number “0xFF”. This checksum will set in place 11 of the array. After this the frame is complete and gets formed into the frame so it can be sent through the XBee antenna.

The fourth part consists of making the sequence. Every time the Xbee sends a frame the counter will increase by one as long as the sequence is lower or equal to 65535. Then the sequence will start over.

The fifth and final part consist of a visa write function, where the frame that is formed is sent to the Xbee antenna. If there is an error before the sending block, the sending block will not be executed. The USB port IN variable will determine through which serial port the information will be sent. This port needs to be the port where the Xbee antenna is connected.

5.3 XBee receiving

The XBee receive VI takes the information that is useful out of the received frame. This VI includes three cases: Receiving for the experimental setup, receiving for communication tests and one for receiving with a filter and a check on the checksum.

The part that stays the same for all of these cases is shown in Figure 33. First of the received frame is imported into LabVIEW. The frames that are received after this, will be put in a buffer. The correct USB port is set, so the myRIO device knows where to look. Next the amount of bytes that is being received is checked.

The frame that was imported is checked for the frame delimiter byte "7E" this indicates where the frame starts. If this byte is not found a zero is passed through. If the start delimiter is found the data will be extracted out of the frame. All these action will be done within the case structure explained in the next chapter.

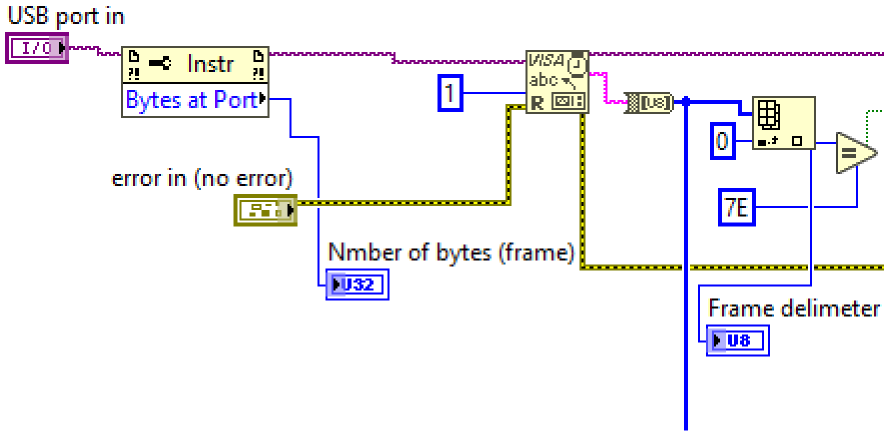


Figure 33: First part of the XBee receive general VI

5.3.1 Case structure of XBee receiving

When the case structure is true, the start delimiter of “7E” was found.

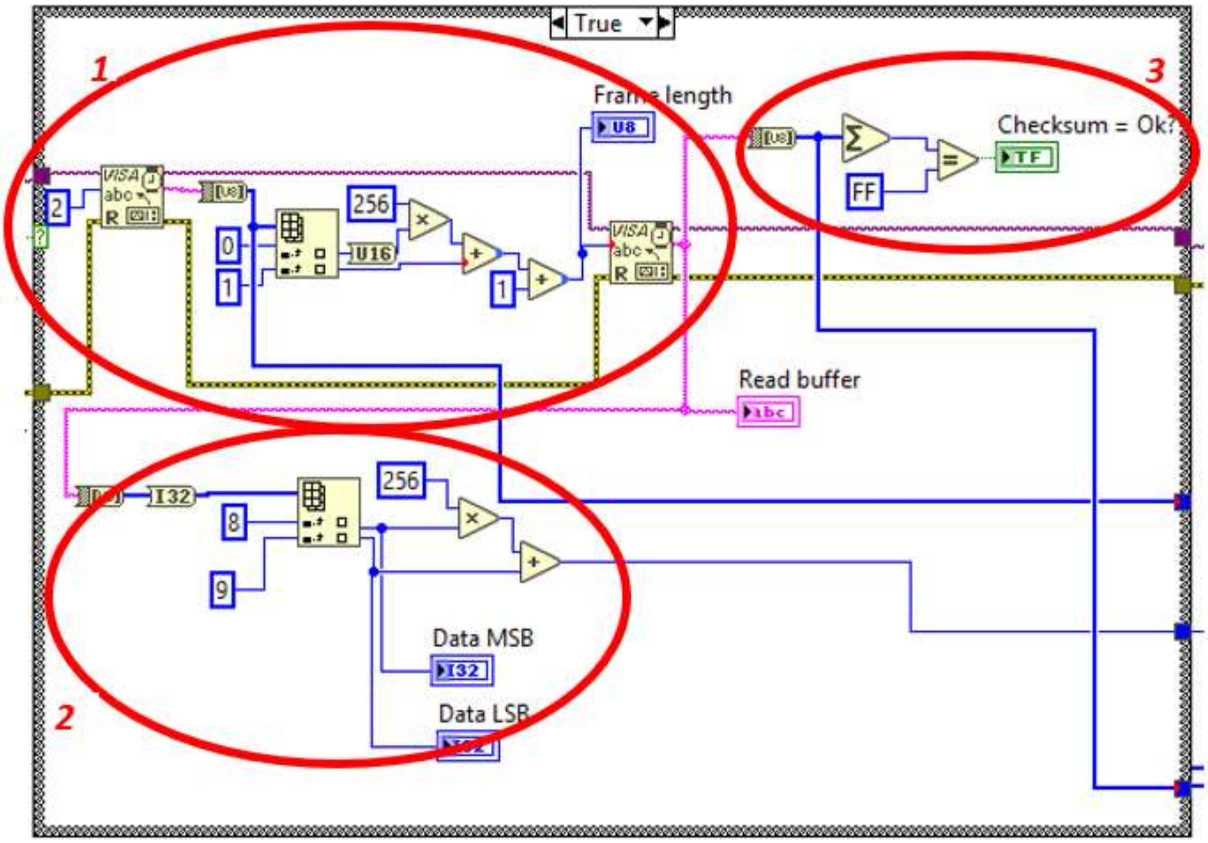


Figure 34: True case structure of XBee receiving

In the first part the two bytes after the start delimiter is checked. These bytes are the least significant byte and most significant byte of the length of the frame. The most significant byte is multiplied by 256 and added to the least significant byte. This value is then added by one for the checksum.

In the second part the data gets converted into a decimal value, this will be the receiving value. The data bytes are located on the eighth and ninth location in the frame.

The third part checks the checksum of the frame The sum of the entire frame is taken without the start delimiter and the frame length and check if this equals “0xFF”.

The last part is where the checksum of the frame is checked. It takes the sum of the whole frame without the start delimiter and the frame length and check if this is FF.

When the case structure is false , the start delimiter of “7E” was not found in the frame.

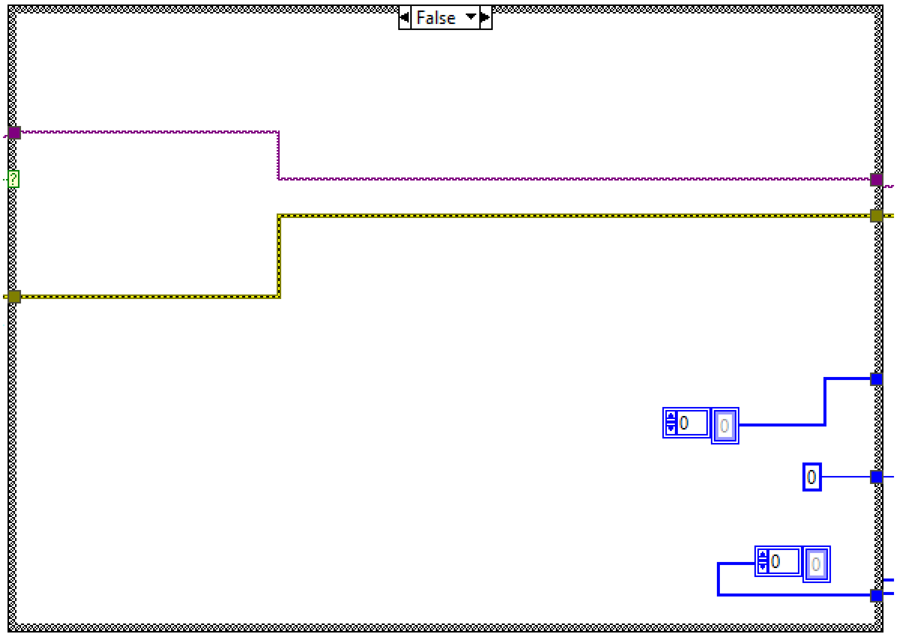


Figure 35: False case structure of XBee receive

The USB port that is being used and the error list gets transferred through and all the values that are being sent are zero.

The final part of the LabVIEW code contain the outputs.

USB port out is for the next serial connection. If there is an error it will go to the error out port. The data that was converted to a decimal value will go to the sending value. An array will be formed and every element of the frame is put in it. This is for checking the received frame.

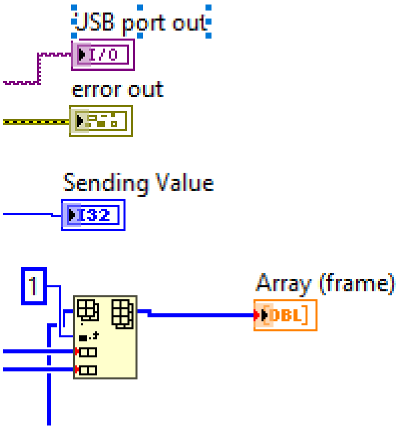


Figure 36: End of the XBee receive VI

5.4 Implementation of PI controller in LabVIEW

As stated before the control algorithm that was chosen for our control network is a PI controller. LabVIEW has a built in PID block, this is used. Only the P and I part of the controller is set.

The created signal between 0-1500 on the left side of the picture is converted to a value between 0-1023. This is the input for the PI controller. The two values above the PID controller block are the output range in between which the values can be sent. The blue line coming from the bottom of the figure is the received value. PID gains are the values that are being set for K and Ti factor.

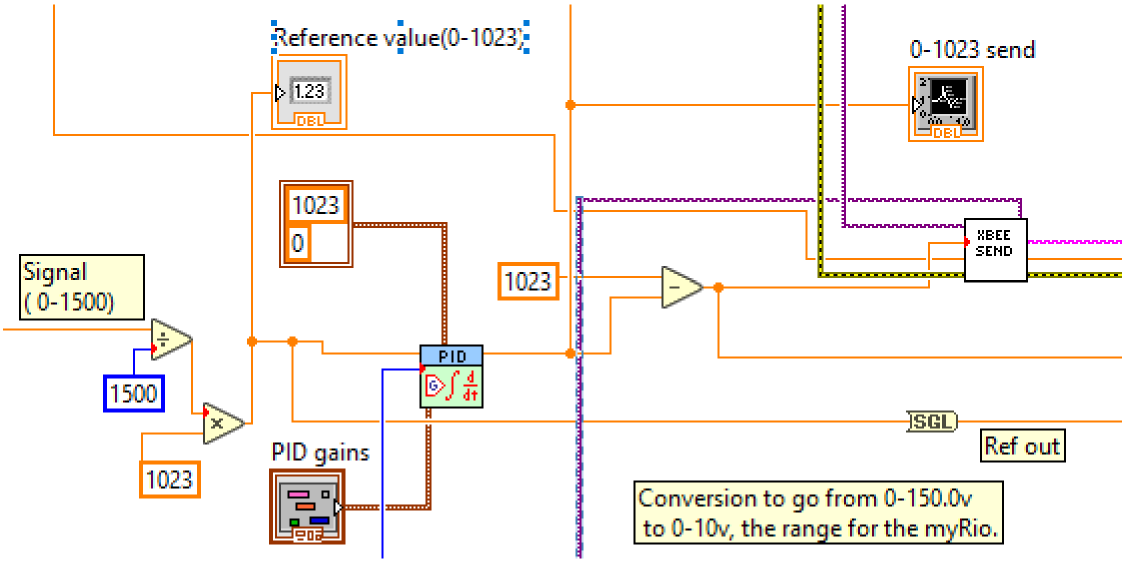


Figure 37: PID implementation in LabVIEW

Figure 38 and Figure 39 show the implementation of the PI controller into LabVIEW. Figure 38 shows how the VI accepts a few inputs such as: Setpoint value, Process variable, PID gains and Output range. As well as one output: Output value.

- Setpoint value: this is the desired value of the control variable.
- Process variable: this is the measured value of the control variable, looped back into the PID controller.
- PID gains: here the values for the control parameters are defined. These include: proportional gain, integral time and derivative time. Since we are only using a PI controller the derivative time is left at 0.
- Output range: the limits within which the controller is allowed to operate.
- Output value: returns the output of the control algorithm.

In Figure 38 a more detailed look at the PI controller and its inputs and outputs is shown.

In Figure 39 the front panel is shown in which we can configure the PID gains. This was done so that the control parameters could be changed quickly and easily. It also serves as a visual reminder on which parameters were chosen for a test.

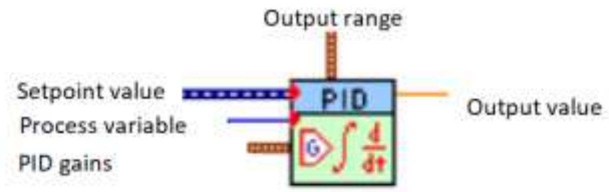
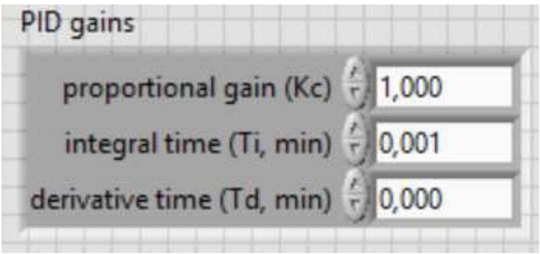


Figure 38 : Block diagram of PiD controller


PID gains

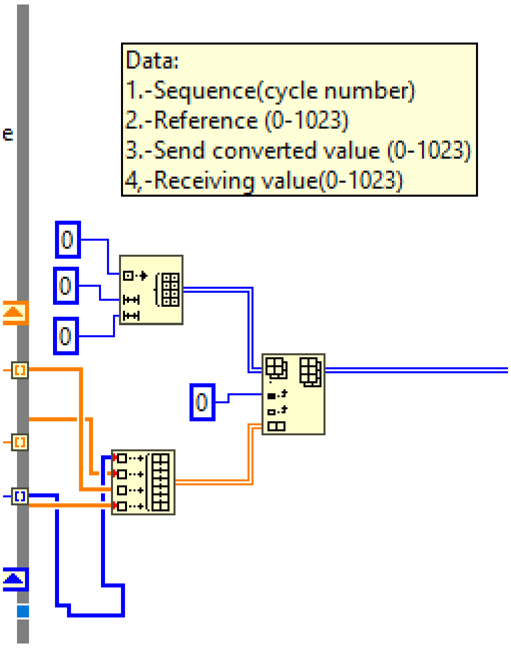
proportional gain (Kc) 1,000
integral time (Ti, min) 0,001
derivative time (Td, min) 0,000

Figure 39: Front panel of the PID gains

5.5 Saving data and analysing

The data from the test need to be saved for analysing. First the data gets collected at the case structure by indexing. After that the data gets put into an array and goes to a Write Delimited Spreadsheet. Here the data is written in to a CSV and txt file.

These files will be loaded into an Excel spreadsheet and from here the data can be analysed and graphs can be made.



e
Data:
1.-Sequence(cycle number)
2.-Reference (0-1023)
3.-Send converted value (0-1023)
4,-Receiving value(0-1023)
0
0
0
0

Figure 40: Saving data for analysis

6 Arduino

6.1 Code for tests with PI controller

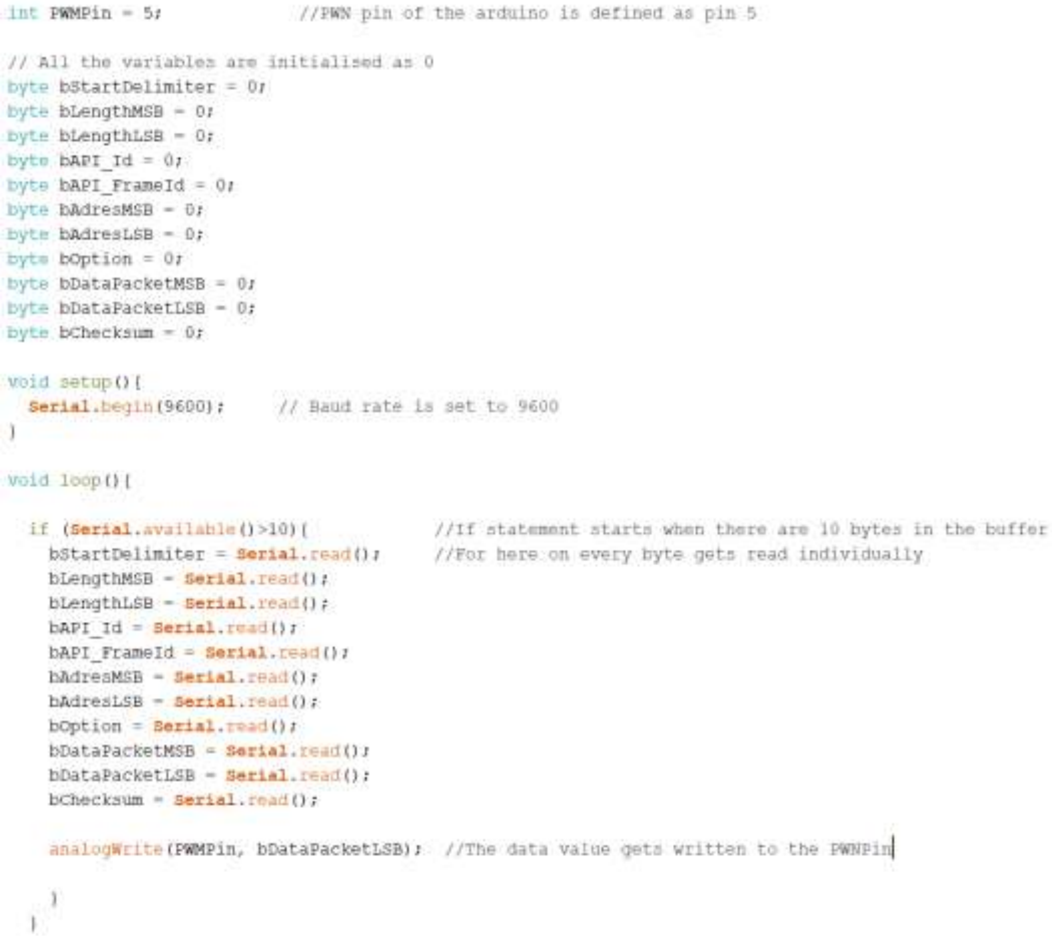
As explained in a previous chapter an Arduino Uno microcontroller is used to interact with the piezo electric actuator. The micro controller receives data from the XBee antenna through the XBee shield, but some code is needed in order to process this data.

Figure 41 represents all the code that is being used in the Arduino to control the actuator. Firstly all the variables need to be initialised. The pin which is chosen as the output for the PWN signal is defined as pin 5. All other variables used later on in the code are given an initial value of “0”.

In void setup the baud rate needs to be set for the serial communication. The value 9600 is set as this will be the same as the baud rate of the XBee antennas.

In void loop the main program runs continuously. If the serial buffer of the Arduino is filled with at least 10 bytes, the bytes can be read. Because the layout of the frame is known we know that the first byte is the start delimiter, second byte is the most significant byte of the length, … Therefore we can save these individual bytes in the correct variables.

Now we are only interested in the actual data byte, in which the position of the actuator is saved. This byte is saved in the variable “bDataPacketLSB”. We only use the least significant bit because the PWM signal of the Arduino only accepts an 8-bit value. This is then written to the previously defined PWM pin with the code AnalogWrite(PWMPin, bDataPacketLSB).

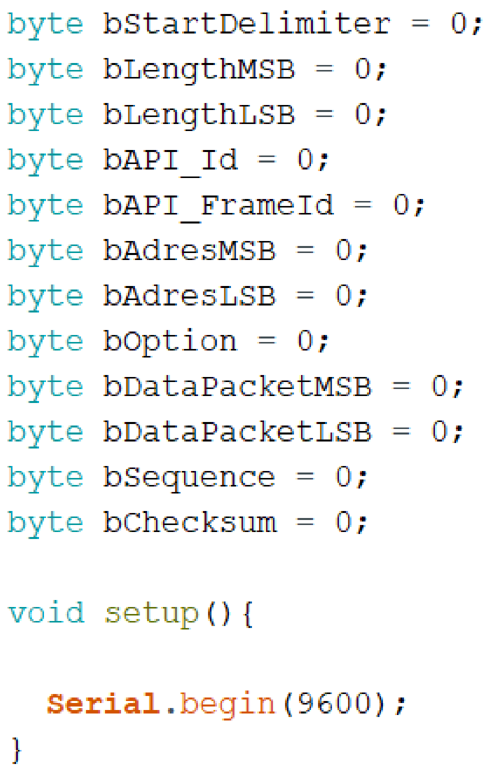


```
int PWMPin = 5;              //PWN pin of the arduino is defined as pin 5

// All the variables are initialised as 0
byte bStartDelimiter = 0;
byte bLengthMSB = 0;
byte bLengthLSB = 0;
byte bAPI_Id = 0;
byte bAPI_FrameId = 0;
byte bAdresMSB = 0;
byte bAdresLSB = 0;
byte bOption = 0;
byte bDataPacketMSB = 0;
byte bDataPacketLSB = 0;
byte bChecksum = 0;

void setup(){
  Serial.begin(9600);     // Baud rate is set to 9600
}

void loop(){

  if (Serial.available()>10){             //If statement starts when there are 10 bytes in the buffer
    bStartDelimiter = Serial.read();      //For here on every byte gets read individually
    bLengthMSB = Serial.read();
    bLengthLSB = Serial.read();
    bAPI_Id = Serial.read();
    bAPI_FrameId = Serial.read();
    bAdresMSB = Serial.read();
    bAdresLSB = Serial.read();
    bOption = Serial.read();
    bDataPacketMSB = Serial.read();
    bDataPacketLSB = Serial.read();
    bChecksum = Serial.read();

    analogWrite(PWMPin, bDataPacketLSB);  //The data value gets written to the PWNPin

    }
  }
```

Figure 41: Arduino code for piezo electric actuator

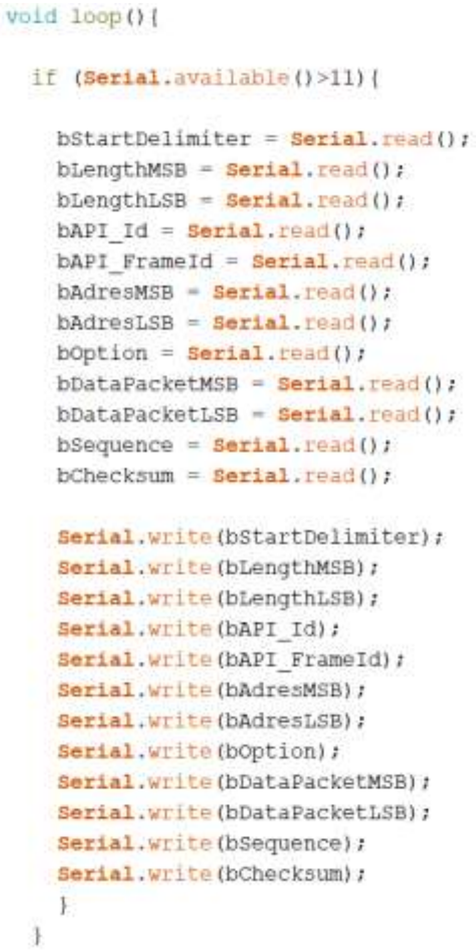
6.2 Code for communication tests

In the next chapter there will be a detailed description on how the communication tests were carried out. But in order to do these tests the code in the Arduino had to be altered. The Arduino now simply reads incoming data and sends it back. The delay between messages can then be measured as well as the package loss.



```
byte bStartDelimiter = 0;
byte bLengthMSB = 0;
byte bLengthLSB = 0;
byte bAPI_Id = 0;
byte bAPI_FrameId = 0;
byte bAdresMSB = 0;
byte bAdresLSB = 0;
byte bOption = 0;
byte bDataPacketMSB = 0;
byte bDataPacketLSB = 0;
byte bSequence = 0;
byte bChecksum = 0;

void setup(){

  Serial.begin(9600);
}
```

Figure 42: initialisation and setup communication code



```
void loop(){

  if (Serial.available()>11){

    bStartDelimiter = Serial.read();
    bLengthMSB = Serial.read();
    bLengthLSB = Serial.read();
    bAPI_Id = Serial.read();
    bAPI_FrameId = Serial.read();
    bAdresMSB = Serial.read();
    bAdresLSB = Serial.read();
    bOption = Serial.read();
    bDataPacketMSB = Serial.read();
    bDataPacketLSB = Serial.read();
    bSequence = Serial.read();
    bChecksum = Serial.read();

    Serial.write(bStartDelimiter);
    Serial.write(bLengthMSB);
    Serial.write(bLengthLSB);
    Serial.write(bAPI_Id);
    Serial.write(bAPI_FrameId);
    Serial.write(bAdresMSB);
    Serial.write(bAdresLSB);
    Serial.write(bOption);
    Serial.write(bDataPacketMSB);
    Serial.write(bDataPacketLSB);
    Serial.write(bSequence);
    Serial.write(bChecksum);
    }
  }
```

Figure 43: Main loop for the communication test

In Figure 42 the code starts, here all the variables are initialised with a value of 0. So that when the program restarts no data gets transferred and everything starts at “0”. These variables all represent the different bytes that will be sent in the frame by the controller.

Next is the void setup, here the serial communication is configured with a baud rate of 9600. This baud rate needs to match those of the controller and all XBee antennas, in order to guarantee good communication.

In Figure 43 the main loop of the Arduino code is displayed. This starts off with an if statement, here the rest of the code will only be executed if there are eleven or more bytes in the buffer of the Arduino.

At first all the bytes get read one by one, and saved in various variables. This allows for easy access to different parts of the frame. This part is almost exactly the same as with the PI controller test, except for one byte. One byte is added after bDataPacketLSB, namely bSequence. For this test it is important that the sequence number of the sent frame is known. Then when this frame is received again by the controller, it can be calculated how long it takes for the Arduino to send this information back.

Lastly the exact variables that were previously read, are sent back in the same sequence via the XBee antenna to the controller.

7 Controller

For the control algorithm a simple PI controller is used. This type of controller is chosen because it is simple to implement, simple to configure and is most commonly used in industrial processes. It is also easy to implement this into LabVIEW as it has its own VI. This VI allows for quick and easy changes to the control parameters.

7.1 LabVIEW real-time programming

"The LabVIEW Real-Time Module combines LabVIEW graphical programming with the power of a real-time operating system, enabling you to build deterministic real-time applications.

A misconception about real-time is that it means quick. More accurately, real-time means in-time. In other words, a real-time system ensures that responses occur in time, or on time. With general purpose operating systems, you cannot ensure that a response occurs within any given time period, and calculations might finish much later or earlier than expected.

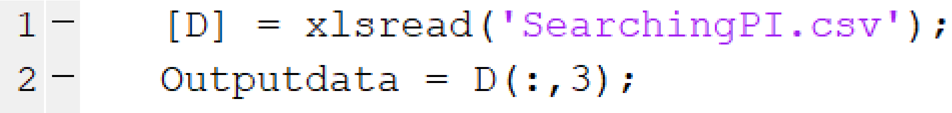
A real-time system consists of software and hardware components. The software components include LabVIEW, the RT Engine, and the LabVIEW projects and VIs you create. The hardware components of a real-time system include a host computer and an RT end-effector." (National Instruments Corporation, 2010)

An option could also have been to make use of this Real-Time programming in LabVIEW. This would make our wireless control network more reliable and more accurate. Unfortunately this method of programming was not used in this paper because of time constraints.

When this programming gets implemented in a real life application this could very beneficial for the control network. It would definitely be a good next step in the right direction to make the control structure more responsive and in general faster.

7.2 Parameter configuration

In order to configure the PI controller a MATLAB toolkit was used to determine sufficient parameters. This toolkit uses a series of data which we acquired through LabVIEW. This program allows us to perform a step-response from which we then can collect data. This data is stored into an excel file, which we then read via MATLAB and stored into a 1-dimensional array.

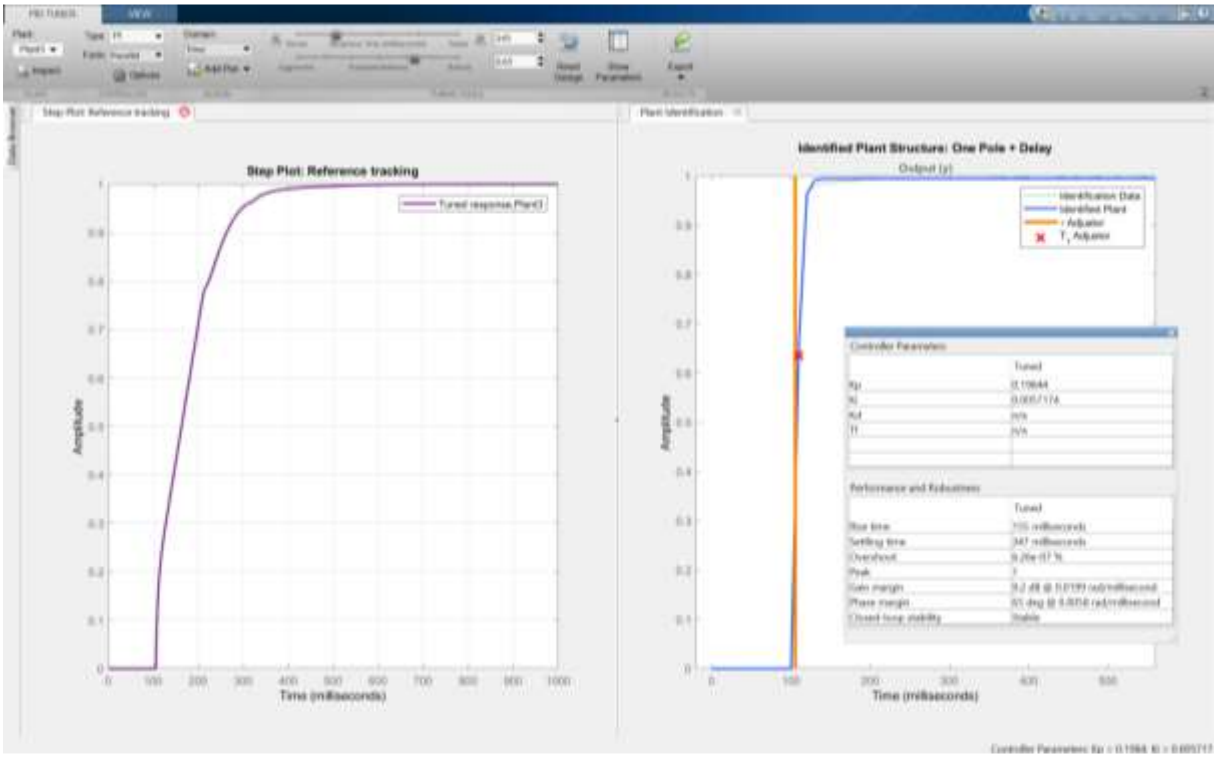


```
 [D] = xlsread('SearchingPI.csv');
Outputdata = D(:,3);
```

Figure 44: MATLAB program to read data from excel file

As you can see this is a very small program, but invaluable in order to determine the PI control parameters. This “Outputdata” is then used as the basis on which a step-response plot is created.

Next a tool called PIDTuner is used to get a rough estimation of the PI control parameters. By visually tuning the step response curve a value for K and Ti is calculated. This is not the most optimal value for the controller but it is a good starting point.



Controller Parameters

	Tuned
Kp	0.15644
Ki	0.0057174
Kd	n/a
Tf	n/a

Performance and Robustness

	Tuned
Rise time	155 milliseconds
Settling time	347 milliseconds
Overshoot	9.26e-07 %
Peak	1
Gain margin	9.2 dB @ 0.0159 rad/millisecond
Phase margin	61 deg @ 0.0058 rad/millisecond
Closed-loop stability	Stable

Controller Parameters: Kp = 0.1564, Ki = 0.005717

Figure 45: PIDTuner and control parameters

These parameters were initially used and a few tests were concluded, and data was collected. The parameters were a good start but required more finetuning. So these values were changed a few times until better results were obtained.

7.3 Lost packages filter

PACKAGE
LOST
FILTER

Figure 46: Package lost filter VI

To have a better controlled system, the package lost filter is made. If there is a package lost , the PI controller will try to make the difference between the measured value and demanded value as small as possible. It will also do this as fast as possible. In reality the measured value is different than the receiving value. Therefore the filter will use the previous received value to control the system.

The package lost filter was the first solution to make the controlled system better. This solution was temporary and there were some doubts about the communication and installation of the plant. Therefore the communication was separated from the plant. This will be explained in a following chapter called communication. The better way to check the frame is with the checksum. This will be explained in chapter modifying the XBee Receive VI.

On the next page a graphic explanation of the LabVIEW program will be given. The VI block works as follows. First it is going to check if the receiving value is equal to 0. If it is not equal to 0 it will just take the receiving value for controlling the system. When the receiving value is 0 it will check if the previous sending value is a 0.

For example if the previous sending value and the receiving value are equal the system just receives the 0 otherwise it will take the previous received value.

It will make sure the PI controller does not need to control extremely when it is not necessary.

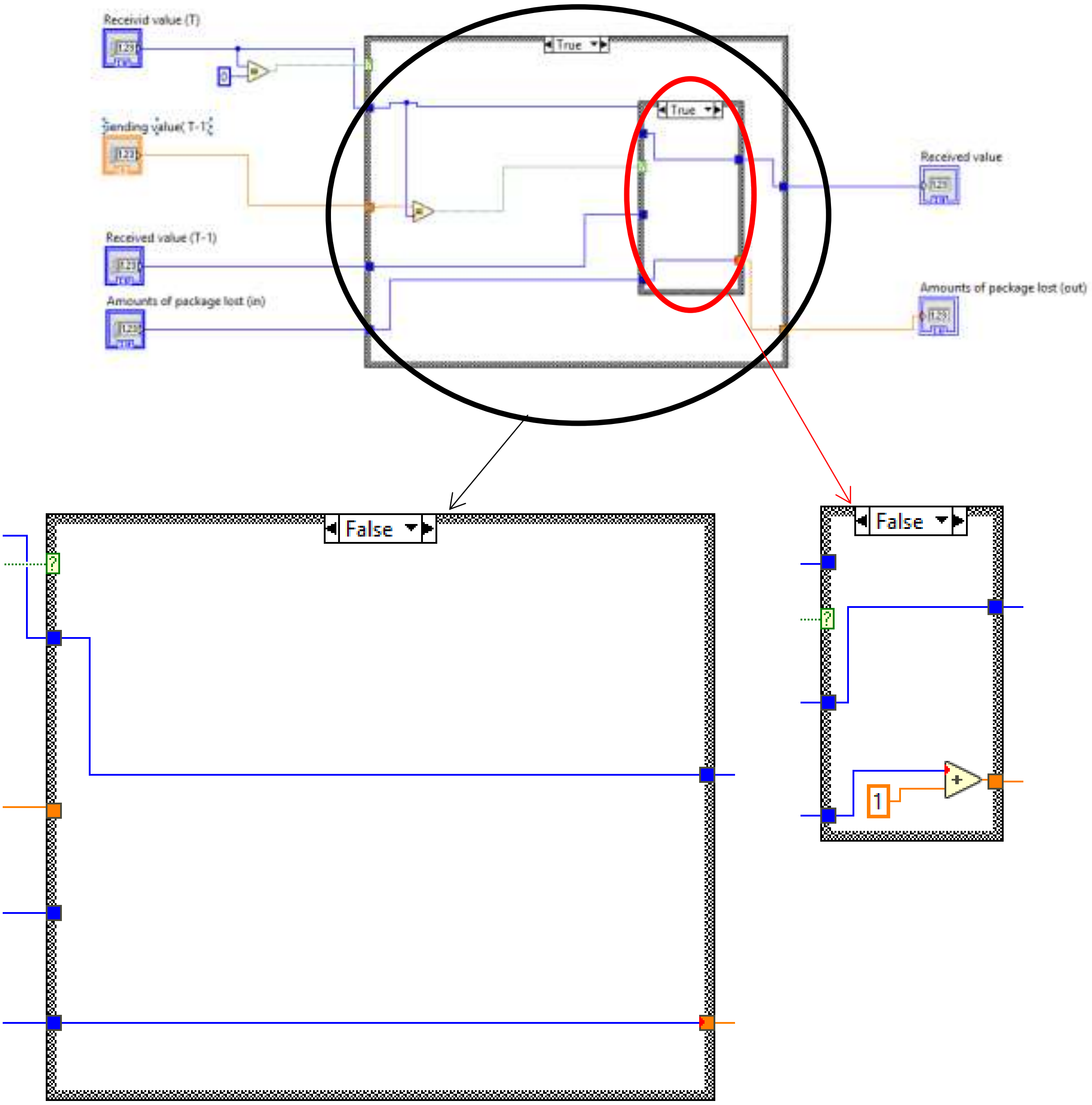


Figure 47: Lost package filter explanation

8 Communication

As discussed in the first chapter, communication is a big part of this paper. To provide the wireless communication XBee antennas were chosen, but are they adequate for control applications? This is what needed to be researched.

In order to test the accuracy and reliability a few tests needed to be done. For this the experimental setup discussed previously needs to be changed.

Only the myRIO controller and Arduino Uno microcontroller are being used. The piezo electric actuator and sensor combination will not be used because this does not affect the communication. The principle behind these tests is to see how the frames are sent and received. And to measure the time delay on these messages also called latency, the amount of messages that get lost as well as the effect of the range on these factors.

8.1 Changes made to the LabVIEW program

In order to test the communication characteristics properly, a few changes needed to be made to the LabVIEW program discussed in chapter 5. These changes will be discussed and explained in this chapter.

8.1.1 Modified creating signal

The tests that were done before were rather short, for the communication tests longer tests needed to be performed. The creating signal block was trimmed down to just include the ramp signal and the steady reference signal. The ramp signal was modified to go up and down between the values of 100 and 900. The steady reference signal was just copied from the previous program.

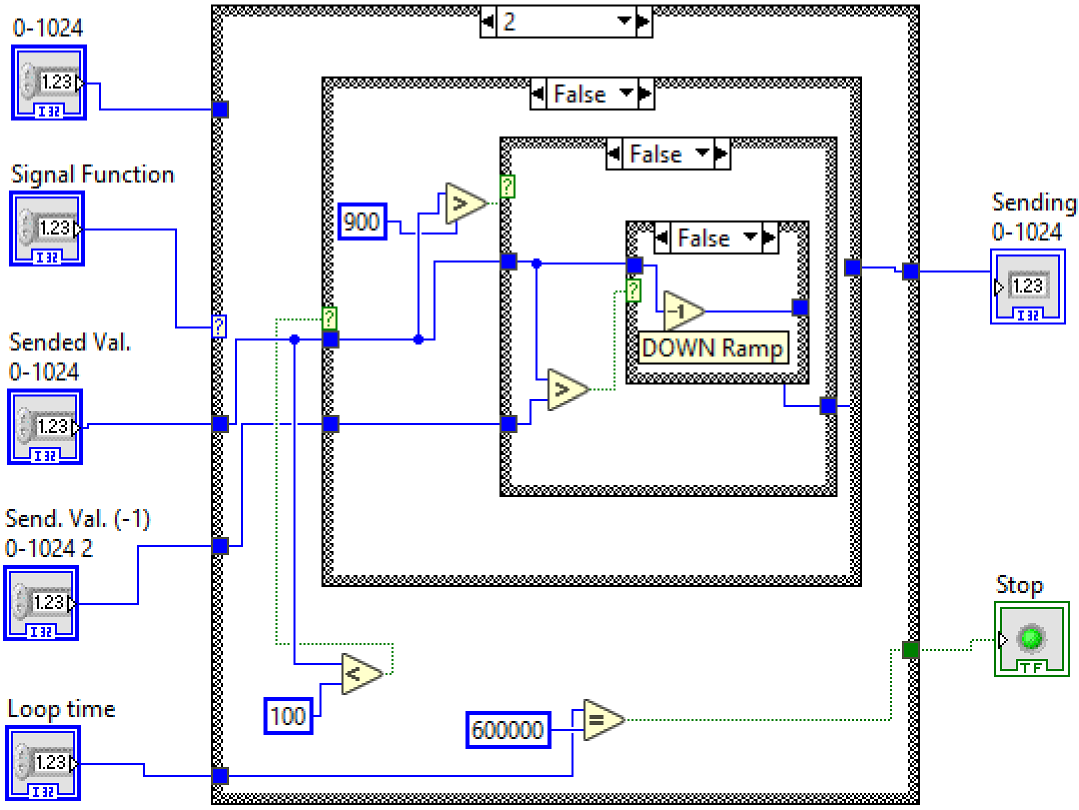


Figure 48: Modified ramp signal

In this test , the number of samples that were taken was 600000. In the first case, if the value is lower than 100 it will just count up with 1. If the value is higher than 900 it will count down. If it is higher than 100 and lower than 900 it will check the two previous values. If it was counting up it will still count up and if it was counting down it will still count down.

This ensures that the correct amount samples is counted.

8.1.2 Modified XBee send

For this VI there where almost no changes. Except the case that there is no possibility to send more than 1023. When the value is higher it will automatically send 1023. If it is less than 1023 it just sends the created signal.

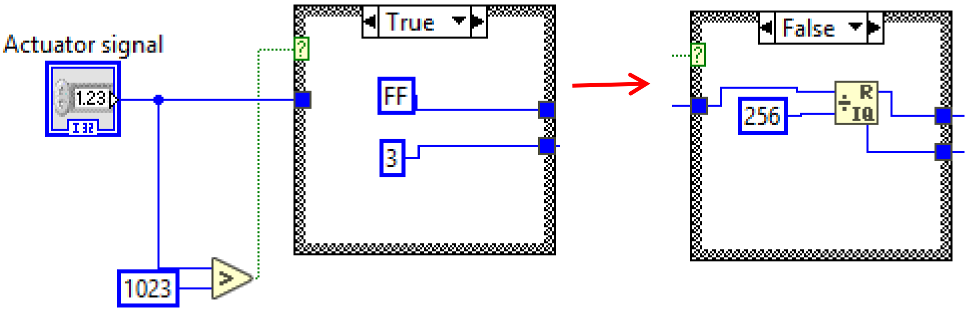


Figure 49: Modified XBee send

8.1.3 Modified saving

For testing the communication there was more interest in knowing which was the sending frame, receiving frame and the checksum.

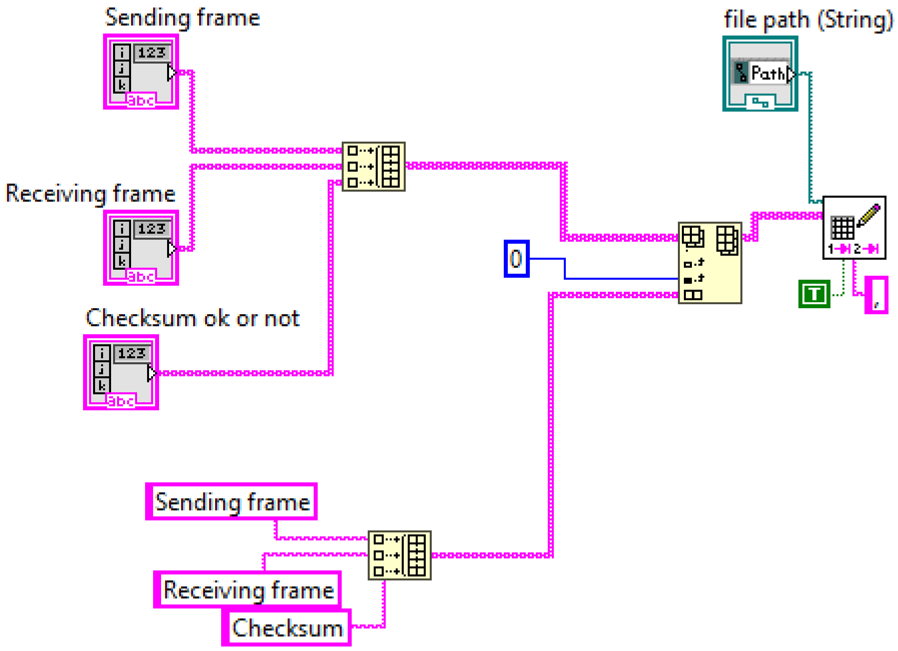


Figure 50: Modified saving code

8.2 First tests

For the first tests a simple ramp test is performed in order to see if the communication works correctly. The results are show in Figure 51 and Figure 52, these results were the best of the tests that were done.

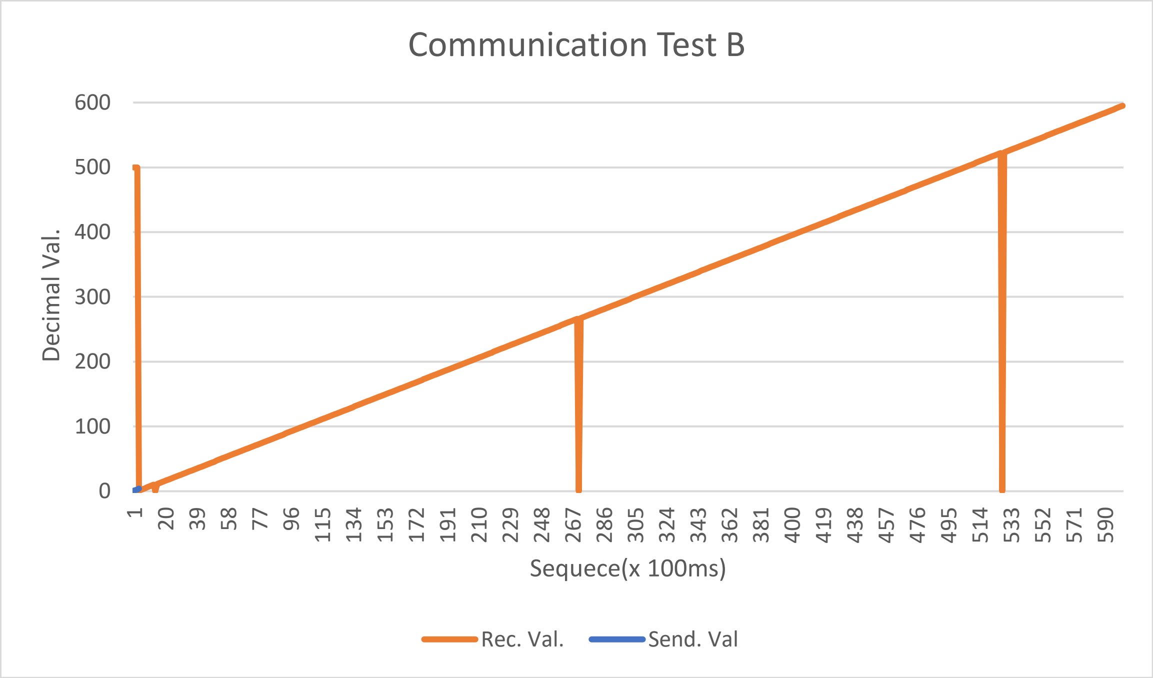


Figure 51: Communication test B

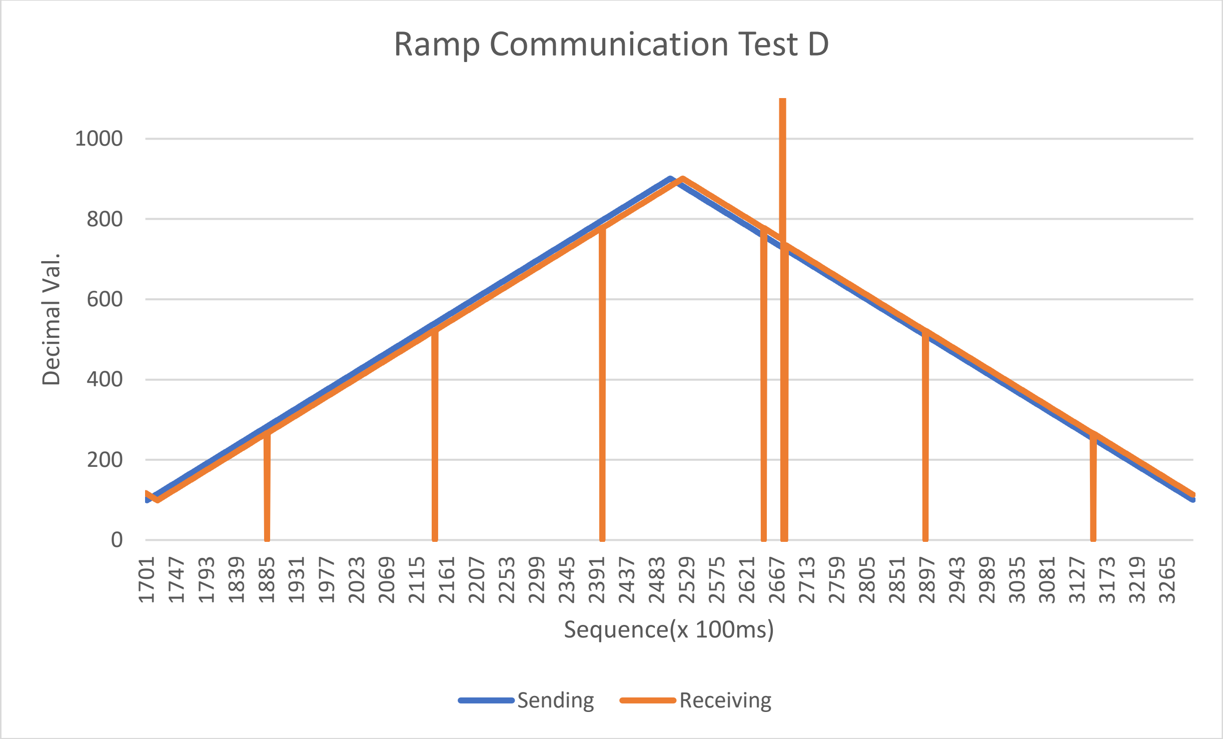


Figure 52: Communication test D

From the first graph it is clear there is a starting error in the beginning of the test. Every time the communication starts there are a couple of strange receiving values. After that the correct values come in, but at a latency.

The second problem that can be noticed is these peaks that are actually zeroes that are being received by the XBee antenna. These zeroes also seem to be periodic, from every test that was carried out these zeroes came back at a periodic cycle. This could be credited towards a fault in reading the frame, but this needs to be researched further in order to get a clear conclusion.

From these graphs the conclusion can be made that the latency is going to be a problem in controlling the plant. The latency depends on the starting error and every time a wrong frame value is read the latency gets bigger. We can derive the next formula from this conclusion:

$$Latency = Starting\ error + amount\ of\ wrong\ frames$$

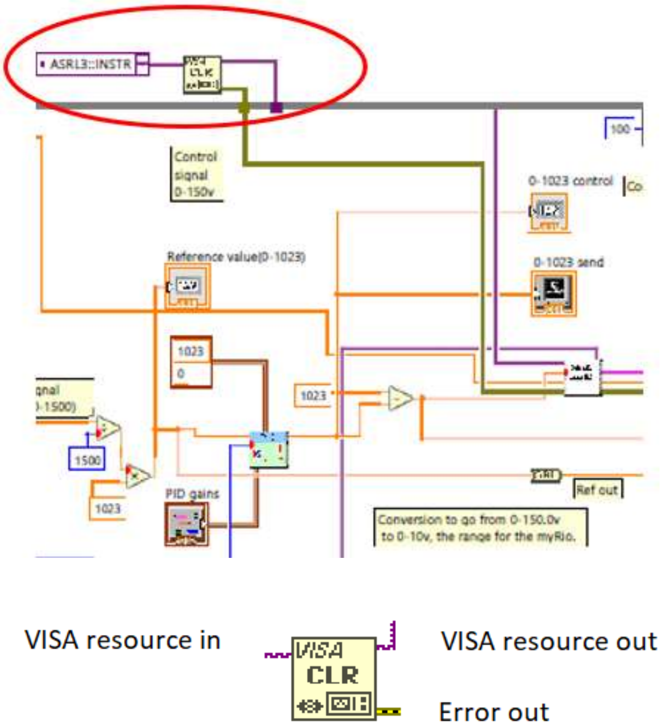
Equation 1: formula to calculate the latency

This means that when the communication cycle runs for a long time, the latency will get worse over time. This will have severe consequences for the controlling aspect of the system. As the latency will get so big that it becomes impossible for the PI controller to effectively control the system.

8.3 Buffer error

To understand where this starting error comes from, a closer look was taken at the received data in the beginning. At first it was thought that these values were random, but after a few tests and upon further inspection it became clear what the problem was.

These values were not random at all, they were the values that were dropped or read wrong in the previous test. This meant that the myRIO would hold these values in a buffer and does not clear this upon starting a new test. This is something that can be solved easily with the VISA clear function in LabVIEW.



Conversion to go from 0-150.0v to 0-10v, the range for the myRio.

Figure 53: VISA clear function in LabVIEW

This function has one input and two outputs. The VISA resource in specifies the VISA resource to open and VISA recourse out is simply a copy of the resource used as an input. Error out contains information about an error that may occur.

8.4 Lost packages

There are a couple of things that can be the cause of this problem. To solve this problem , the cause needed to be more clear.

From the first tests it was clear that these wrong frames were periodic. Upon further inspection is also became clear that these frames were missing the data and checksum. In the next loop no frame was received and after that the myRIO is getting the frame that was wrong or not received entirely.

First of all this wrong frames are at a specific time and are periodic. The frame that the myRio receives is missing the data and the checksum. The loop after the fault there is no frame and after that the myRIO continues with the next value. An example is given in Table 1

Previous rec. Val.	Current Rec. Frame
8	0100 1111 0100 09D2
9	0100 1111 0100
10	
0	0100 1111 0100 0BD0
11	0100 1111 0100 0CCF

Table 1: example of missing frame

There are a couple of possibilities for the problem. There can be a problem with the baud rate, frame length and the loop time in LabVIEW. Maybe more characteristics will affect this problem but these are the ones that are researched in this paper.

To check if it relies on the loop time in LabVIEW, A simple way is to just change the time of the wait function. This quick change proved that there were no changes, frame were stilling being read wrong. Figure 54 and Figure 55 shows the results of two test with different loop times. Of course more tests were done but were not included here because they all showed the same results.

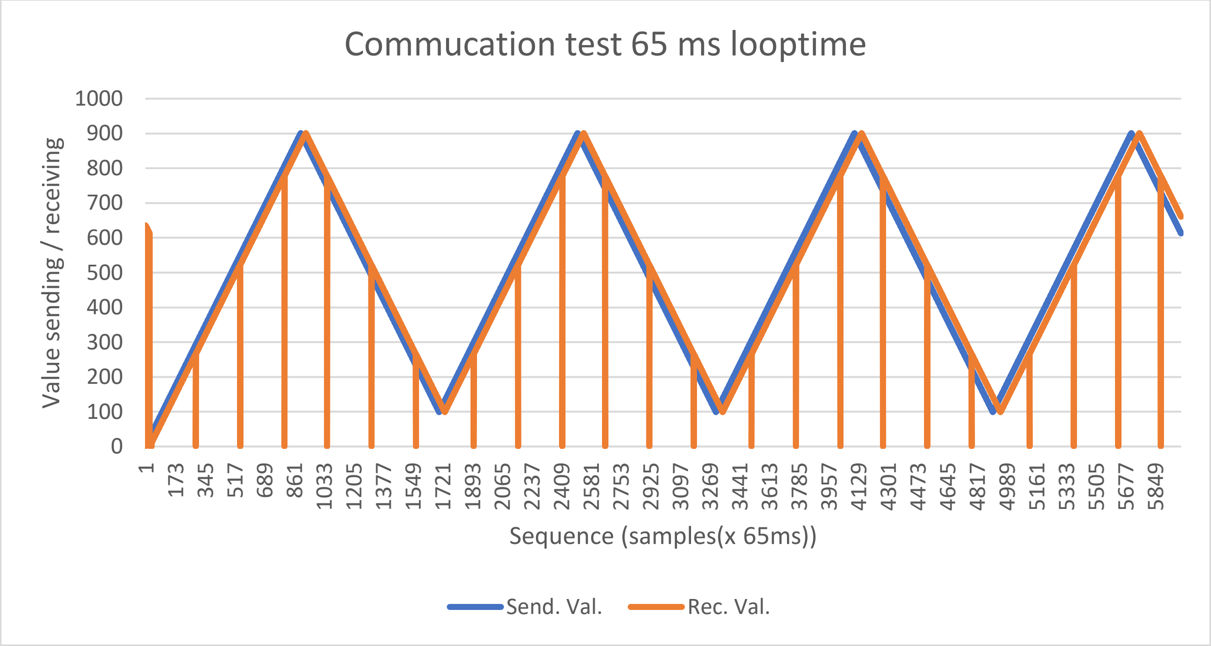


Figure 54: Communication test 65 ms looptime

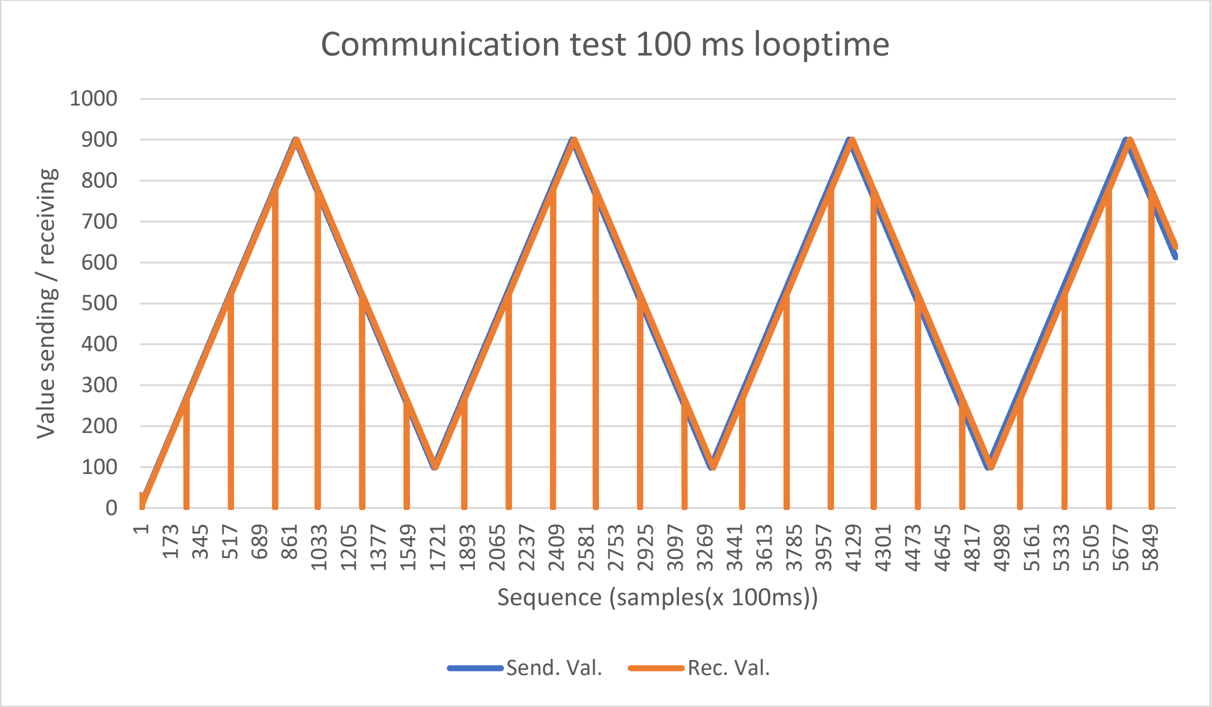


Figure 55: Communication test 100 ms looptime

The Second possibility was a change in frame length. This needed to be done in the LabVIEW code as well as in the Arduino code. Instead of only sending data to the Arduino, the myRIO is sending now also the sequence number so it can always be checked which frame was read wrong or lost entirely. Again after a couple of tests there was no change in receiving the wrong frames. These graphs looked the same as the ones in Figure 54 and Figure 55 so these were also not included here.

The last option was that the clocks were not in sync. Therefore the baud rate needed to be changed so that the periodic nature of these wrong frames would change, and we could record this data. The serial communication of the myRIO and Arduino were set to 9600 as standard, this value was changed to 4800 and 19200 and a couple of communication tests were done. The wrong frames still kept coming in at the same intervals, so this was not the solution after all.

After many more failed attempts at trying to find the solution something remarkable was noticed while reviewing the data. When constantly sending the same value, no frames were read wrong, no matter how long the test was carried out.

The only time the frames were read wrong was when the hexadecimal value “0A” was sent via the XBee. This is the hexadecimal number for 10, but is also found in the numbers 266, 522 and 778. This would explain why the lost packages seemed periodic because the nature of the tests had always been a ramp.

To test this further we plugged both antennas directly into the computer and tried to send a frame with the data byte “0A”. Via the XCTU software console it could be checked what the antennas sent and receive. The frame got sent correctly from one XBee to the other XBee. From this we could deduct that the antennas were not the problem.

The next step towards finding the solution is to check the LabVIEW program. In order to investigate this further it needs to be known how LabVIEW converts the data into values. To visualise this the VISA read function is changed to a String subset function.

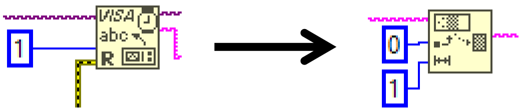


Figure 56: Example VISA read to String subset function

In Figure 56 the situation before and after the change is shown. The string subset function takes in this example the first byte out of the frame. This changes how the LabVIEW program reads the data, but this did not make a difference for the dropped frames.

8.5 Modifying the XBee receive VI

There were some problems with the 0A and the receiving VI doesn’t check if the checksum is ok. At this point it does not matter if the frame is correct of not, the program will use it anyway. Every time there was a problem with the 0A the checksum was also not correct. So these two problems can be solved at the same time.

The thought process to improve the controlling part is when there is a fault in the checksum, the myRIO is going to use the previous frame and use that data to control the system. All the changes are in the receive VI. In this paragraph, the changes are explained to the XBee receive VI.

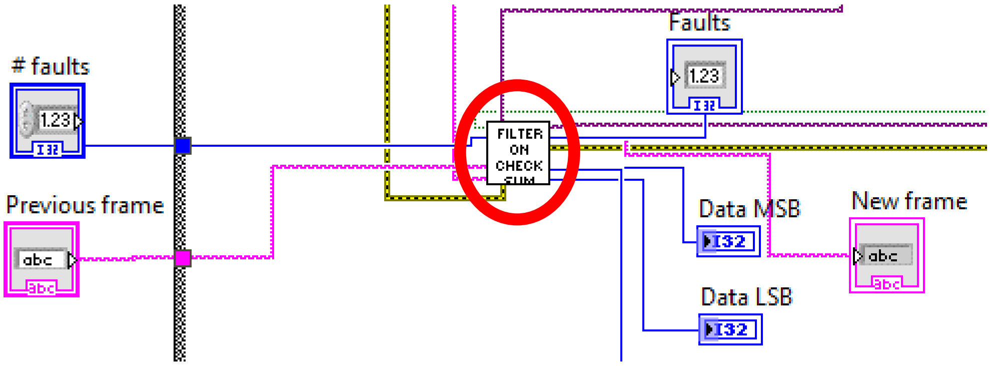
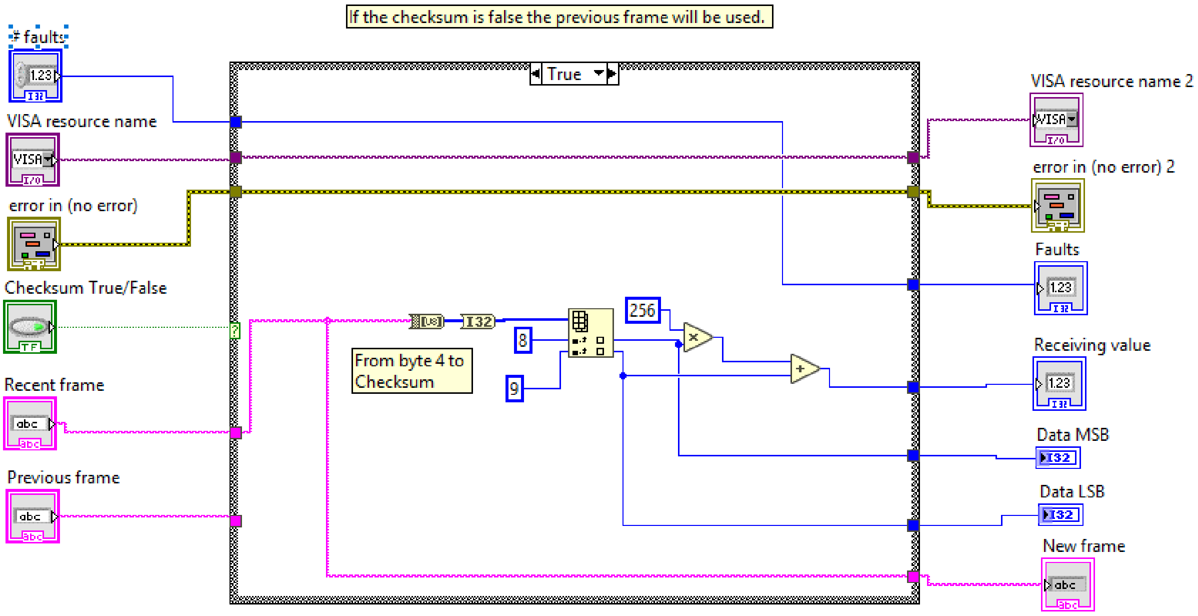


Figure 57: Changes to XBee receive VI

The first two changes are saving the faults and saving the previous frame. Every time a frame gets lost or interrupted and the previous frame gets sent, it will count up with one.

The red circle in Figure 57 is shown in detail in Figure 58.


If the checksum is false the previous frame will be used.

From byte 4 to Checksum

Figure 58: True case of the checksum check

There is a case structure in the file with a true and false case. If the checksum is correct it will carry out the true case. This will do the same as the other receive Xbee VI discussed in chapter 5.3.1.

The false case means that the checksum isn't correct and the previous frame need to get used.

Not only the previous frame will get used but the buffer will get cleared. It is also interesting to save the total number of frames that are lost. The rest of it will be the same as the previous receiving VI discussed in chapter 5.3.1.

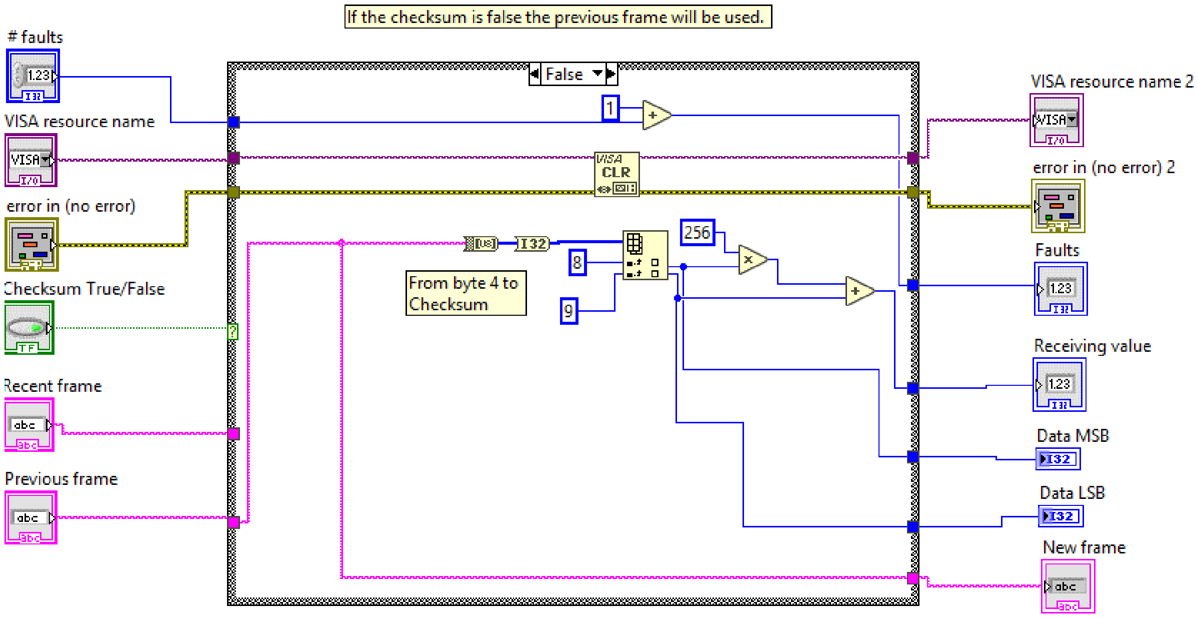


Figure 59: False case structure of the checksum check

9 Comparative system tests after solving the problems

Finding the problem for the frame lost every time there was a 0A in the data, was not successful.

But solving the problems was not an issue. The difference between the value you want and the receiving value is very small after solving the buffer error and the frame lost.

In the two tests below the sending value is the reference signal. The PI will control the sending value depended on the received value. Both PI controllers are configured with the same parameters. In the two examples they are not getting to their maximum value.

Every test is done with a loop time of 100 ms. So one sequence is equal to 100ms. The setup of the test was like in Figure 14 and Figure 15 earlier in this paper.

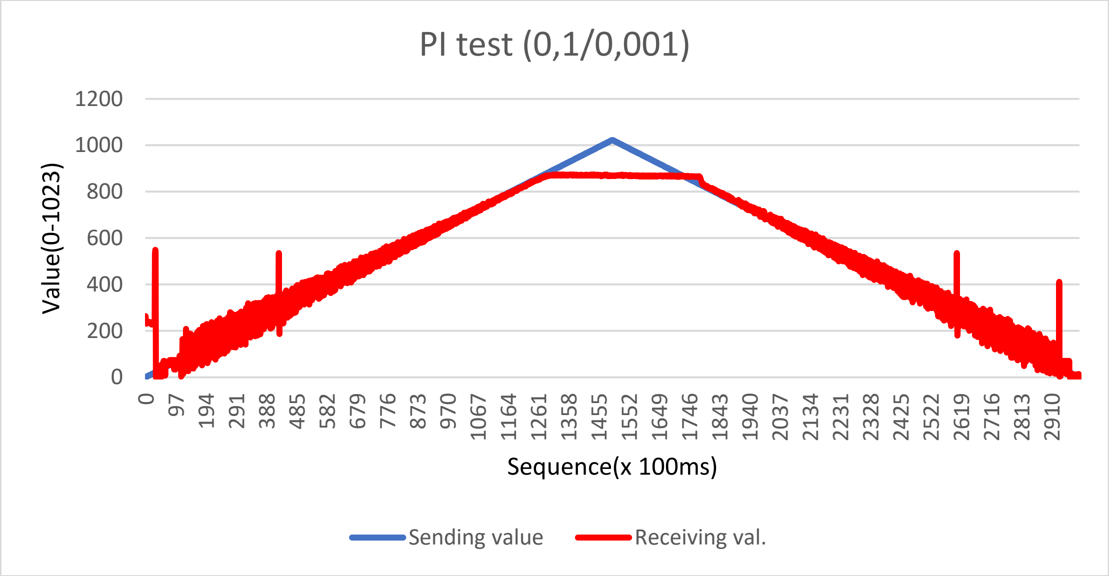


Figure 60: Test before optimisation

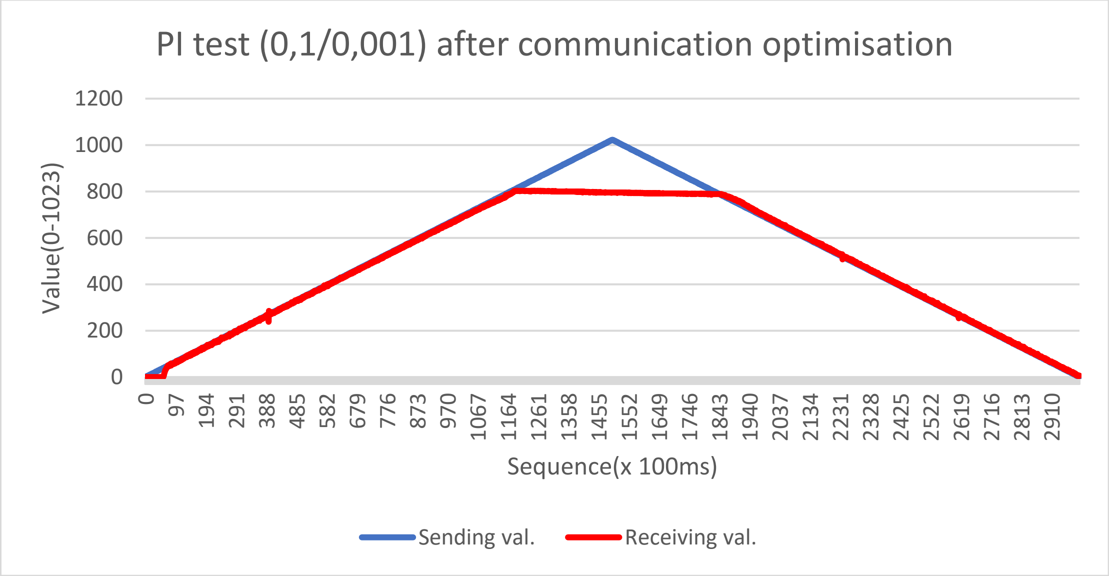


Figure 61: Test after optimisation

To test the difference between these two graphs, the deviation was taken between sending and receiving. Because of not reaching the maximum, the first 800 samples were taken. For the test before the communication the average deviation was 34,54. AT the second test the average deviation was 5,23. This is respectfully 3.58% and 0.51%.

9.1 Test after increasing the input voltage

To reach the maximum of the piezo electric actuator, the voltage between the emitter and collector was increased to 10.4 V from 10 V. To find out how the maximum voltage could be sent over the actuator, the option steady reference was chosen to create the signal. The myRIO sends a value of 1023 to the Arduino and after that the voltage over the emitter and collector was increased slowly until 150V. The results are shown in Figure 62 below.

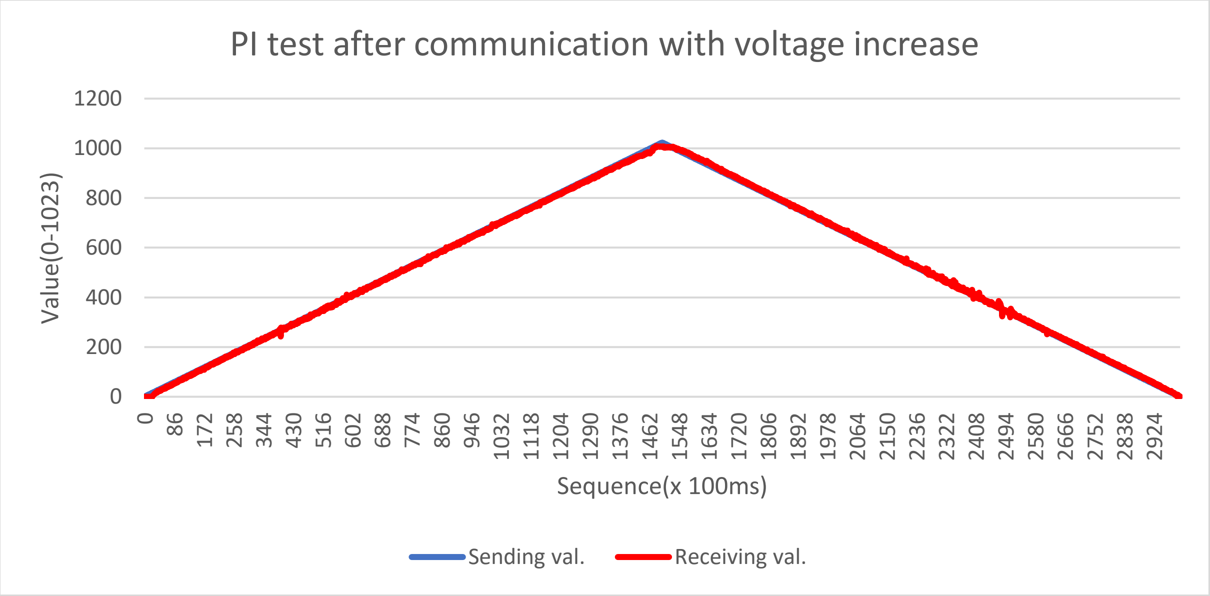


Figure 62: PI test after optimisation with voltage increase

This test shows very promising results because there is not much delay. Furthermore the plant that is being used for this test is delicate and not optimized.

The deviation for this test was also very promising because the average deviation is 0.44. This percentage is the deviation of the total value(1023). The average deviation is bigger around the maximum value of the system. This deviation error is shown in Figure 63.

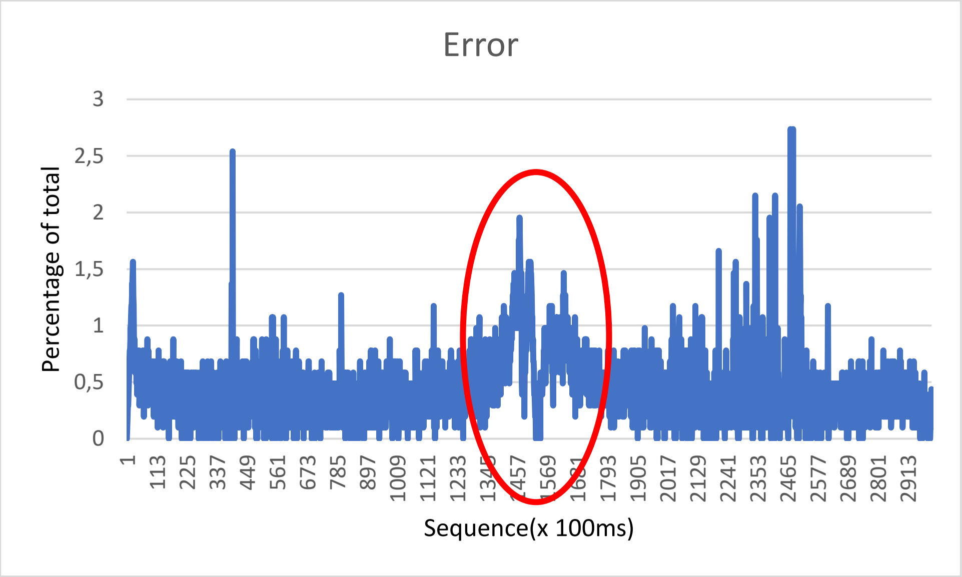


Figure 63: Deviation error between sending and received value

10 Conclusion

As stated in the first chapter of this paper, the main goal was to create a wireless control loop with a PI controller, sensor and end effector. As also discussed in the previous chapter, this was successful. We of course did not stop at just the implementation. Some problems were encountered such as package losses and latency. These are characteristic to wireless systems but it was deemed necessary to further research these aspects in order to determine the validity of using wireless networks in a control structure.

During the communication tests more problems were found with the wireless transmission of the data. The latency problem was fixed relatively easily by clearing the buffer before starting the communication but the package loss problem was not solved so easily. A lot of tests and solutions were tried but often without success. In the end we were still unsuccessful in finding the cause of this problem but at least a solution was found. By implementing the package loss filter discussed in a previous chapter, this was no longer an issue and further tests could be done.

With the communication problems solved, new tests with the PI controller could be carried out. Something that immediately became clear was that these communication errors caused a lot of interference with the controlling aspect of our setup. Without the package losses the controller did not have to react so harshly and the end result was a much cleaner and accurate control signal.

Overall we can conclude that the implementation of the PI was successful, however there are some things to take into account. There are a few problems inherent to the wireless transmission of data. For instance this should only be used with non-critical systems. Also for systems that require real time control, the response time of the wireless systems is just too slow. This could be possible by using the real-time programming way with LabVIEW but this remains to be tested.

11 Future work

One of the things that need to be researched is why the frames with “0A” data bytes get lost. There is a possibility that this is a programming error but a closer look needs to be taken.

It might also be interesting to do more long distance tests as well as trying to control multiple plants. Maybe a change in controlling algorithm might be significant to research, or at least get better control parameters for the PI controller.

And finally, trying to convert the current LabVIEW programs to real-time LabVIEW programs. We think this could be of great value, and improve the system even more.

This project was based more around an ideal PI controller in a wireless network. It will also be interesting to put in some external interferences and research the changes this causes to the control loop.

12 References

Abrahams, S. (2018). *Creation of a Wireless Networked Control System using XBee antennae and LabView.* Opgehaald van Limo: http://depot.lias.be.kuleuven.ezproxy.kuleuven.be/delivery/DeliveryManagerServlet?dps_pid=IE10802480

Digi international. (2017). *XBee/XBee-PRO S1 802.15.4 (Legacy).*

DIGI International Inc. (2018, November). *XCTU user manual*. Opgehaald van Digi: https://www.digi.com/resources/documentation/digidocs/pdfs/90001458-13.pdf

Doering, E. (2016). *NImyRIO Project Essentials Guide.*

Harris, M. H. (2010). *Getting Started with XBee RF Modules.*

Jimblom. (sd). *Serial communication*. Opgehaald van Sparkfun: https://learn.sparkfun.com/tutorials/serial-communication/all

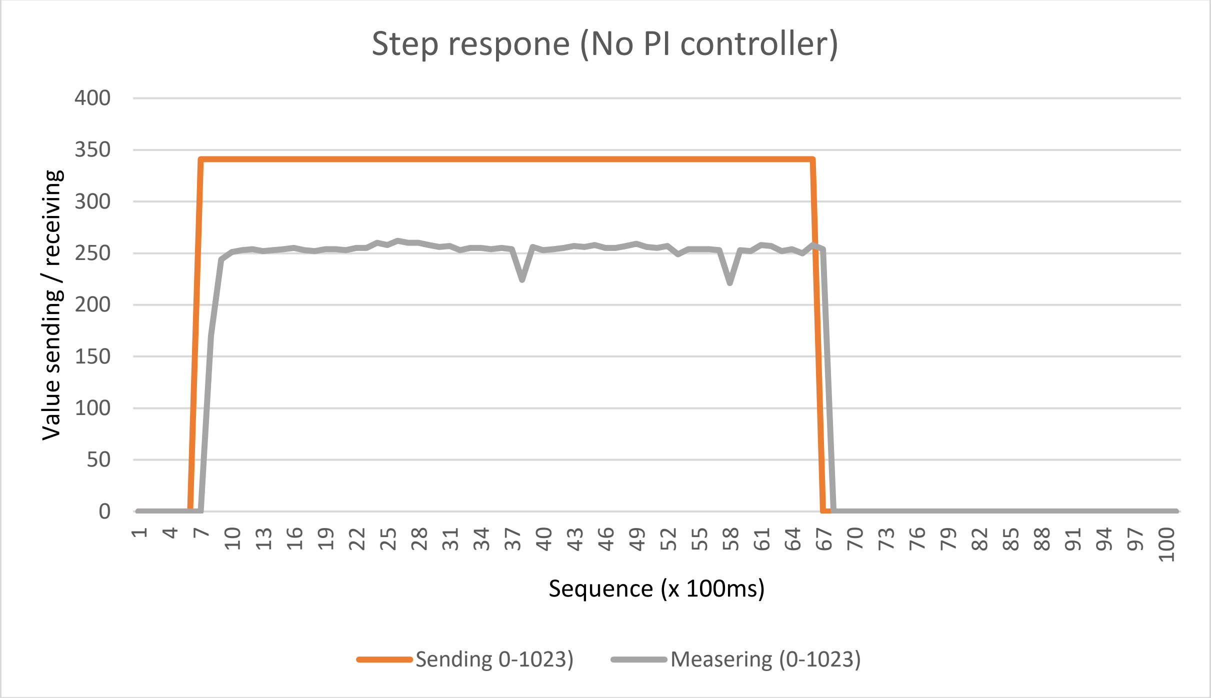
K Smriti Rao, R. M. (2014). *Comparative study of P, PI and PID controller for speed control of VSI-fed induction motor*. Opgehaald van ijedr.org: https://www.ijedr.org/papers/IJEDR1402230.pdf

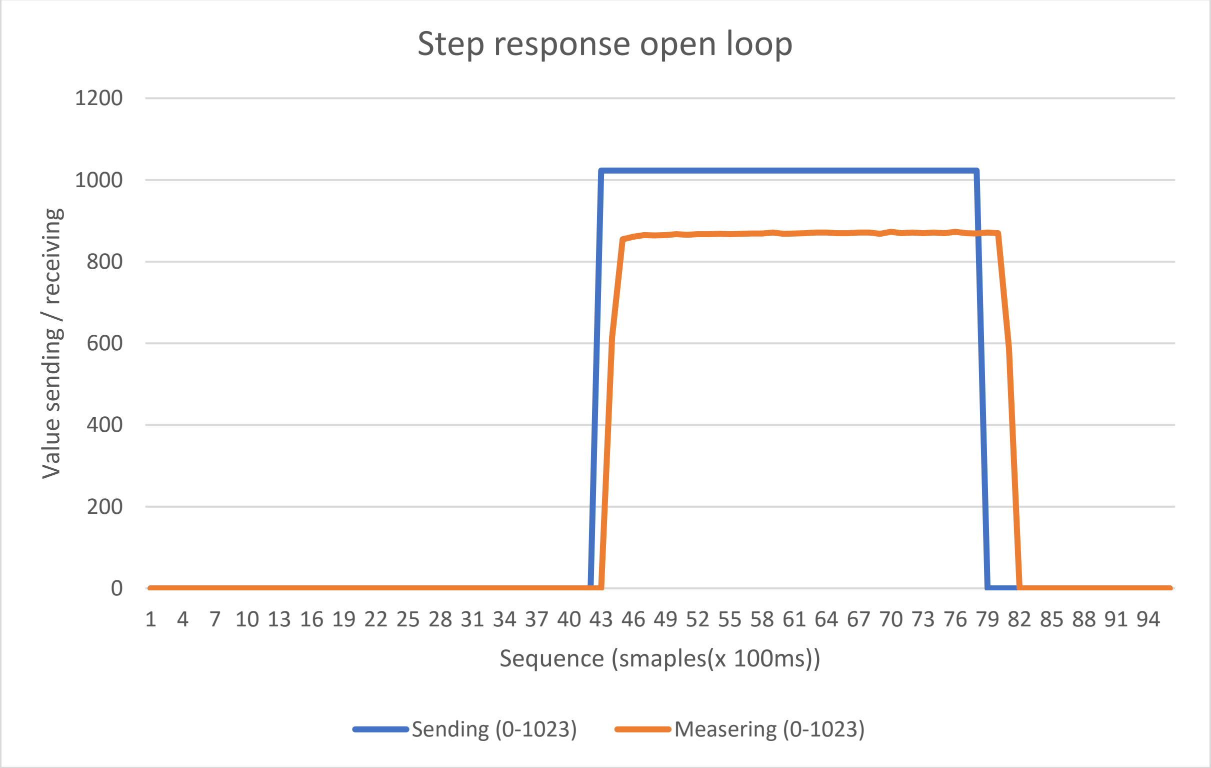
National Instruments Corporation. (2010, September). *twiki.cern.ch*. Opgehaald van CERN: https://twiki.cern.ch/twiki/pub/Main/IntroductionToLabview/Labview_RealTime_Module_-_basic.pdf

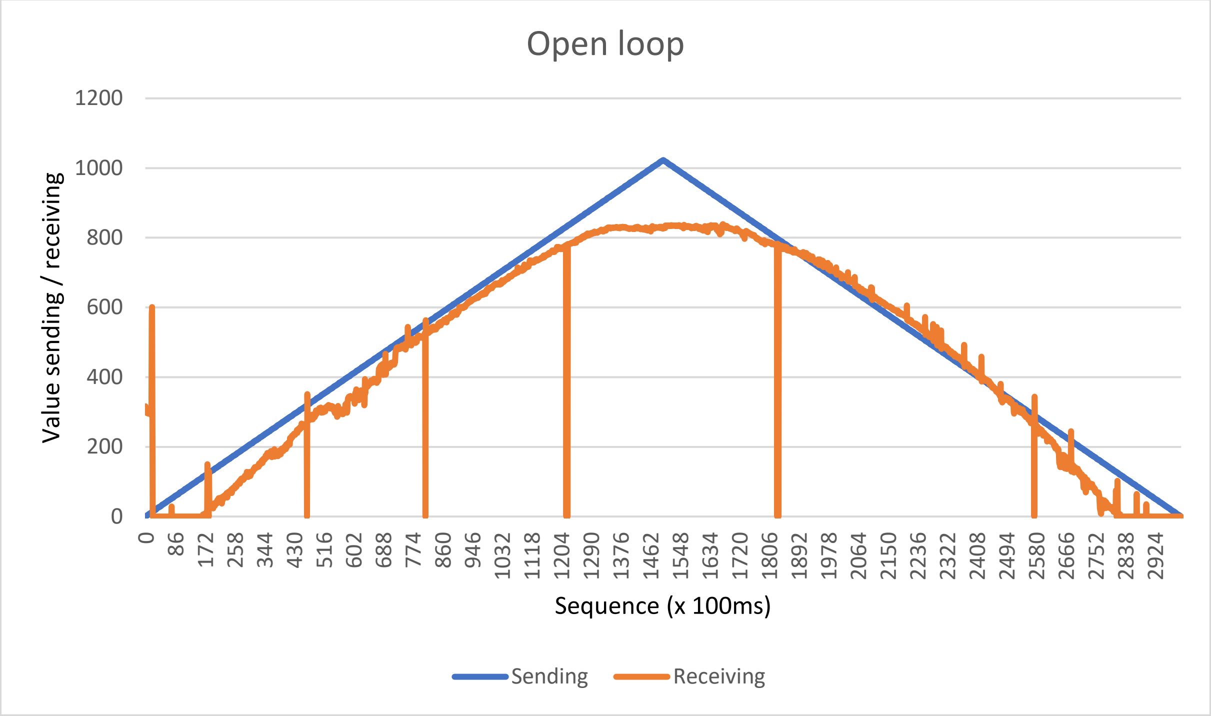
Park, P., Ergen, S. C., Fischione, C., Lu, C., & Johansson, K. H. (2017, December 06). *Wireless network design for control systems : A survey.* Opgehaald van Limo: https://ieeexplore-ieee-org.kuleuven.ezproxy.kuleuven.be/document/8166737/authors#authors

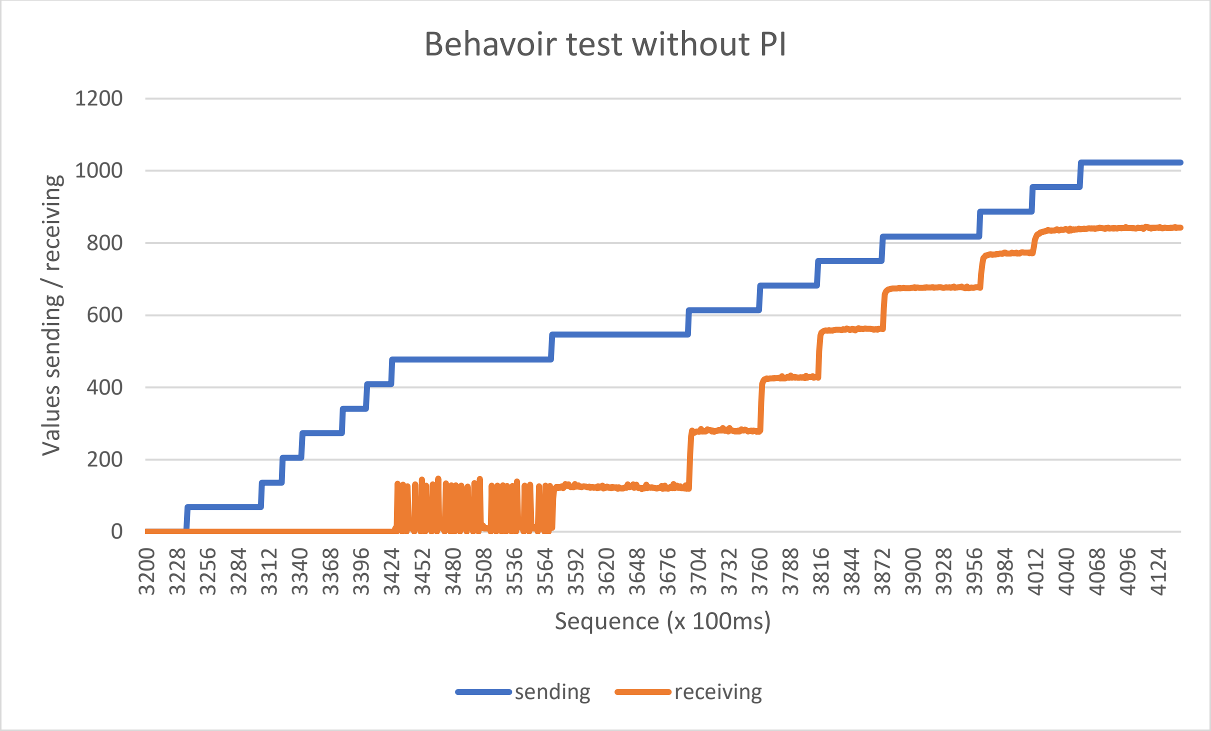
Appendices

The following graphs represent the tests that were done at the very beginning, with the experimental setup and without a PI controller.

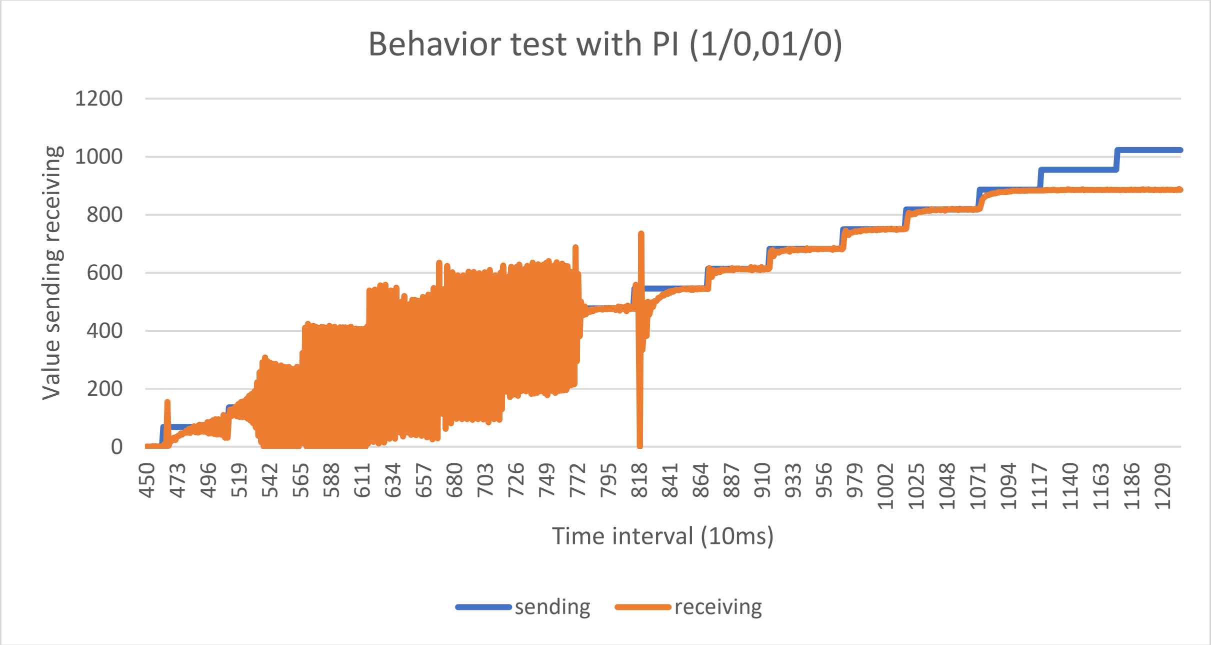


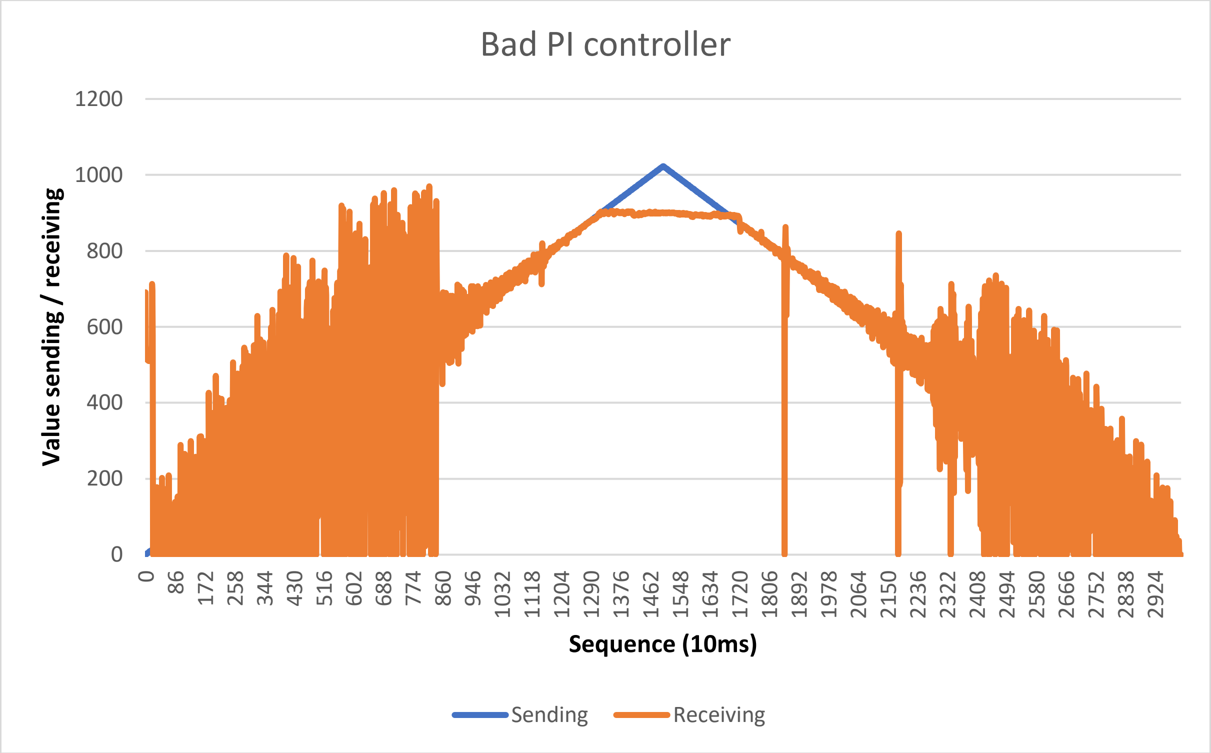


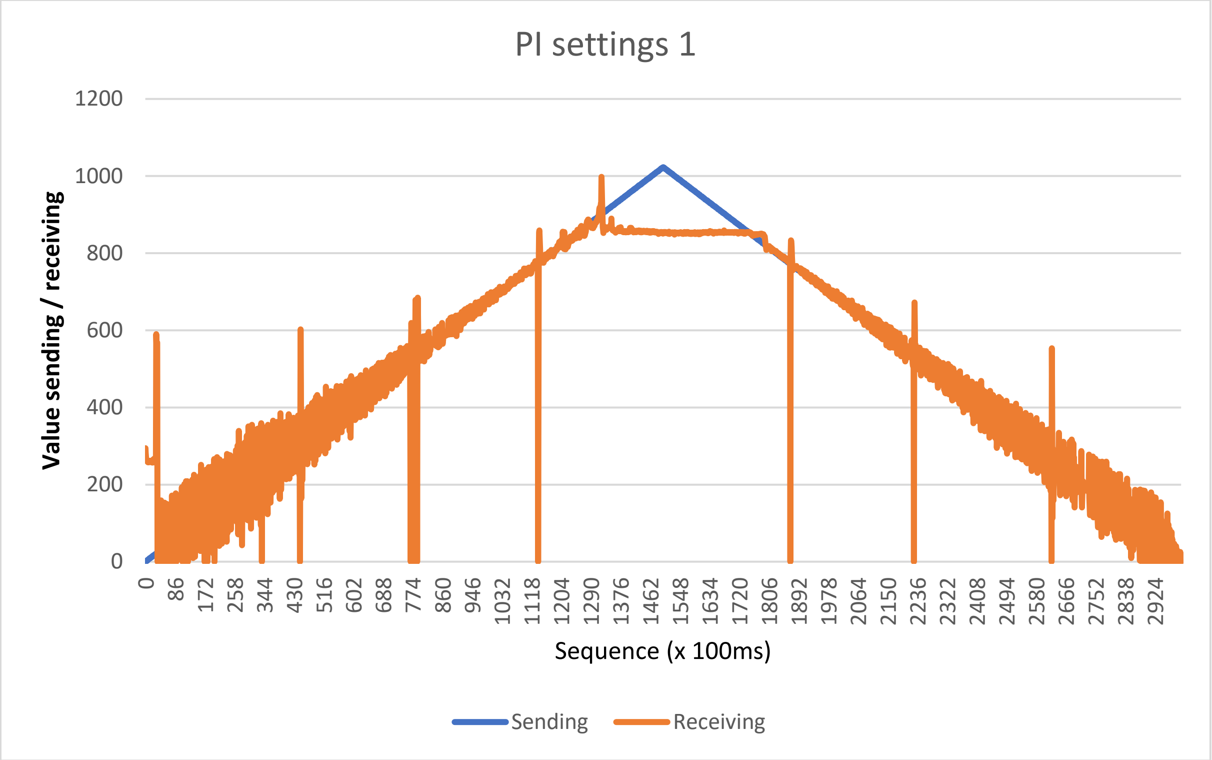

Open loop
Value sending / receiving
1200
1000
800
600
400
200
0
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence (x 100ms)
Sending
Receiving

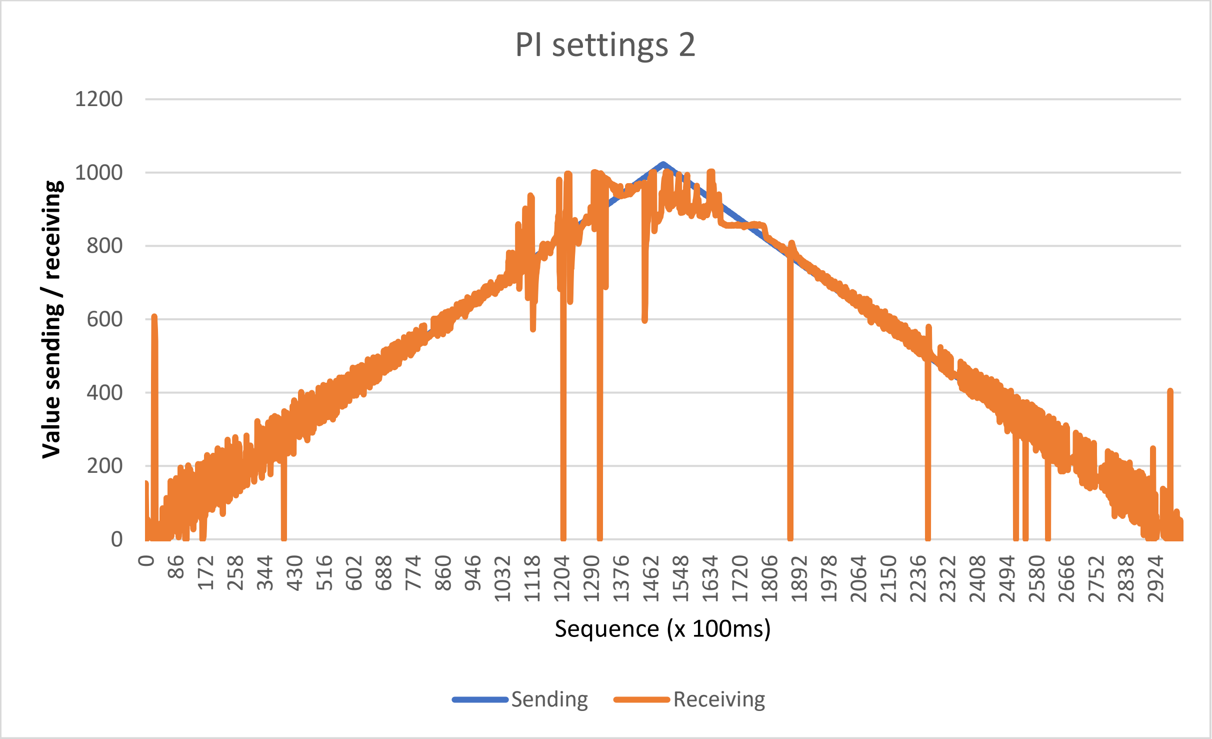

Behavoir test without PI
Values sending / receiving
1200
1000
800
600
400
200
0
3200
3228
3256
3284
3312
3340
3368
3396
3424
3452
3480
3508
3536
3564
3592
3620
3648
3676
3704
3732
3760
3788
3816
3844
3872
3900
3928
3956
3984
4012
4040
4068
4096
4124
Sequence (x 100ms)
sending
receiving

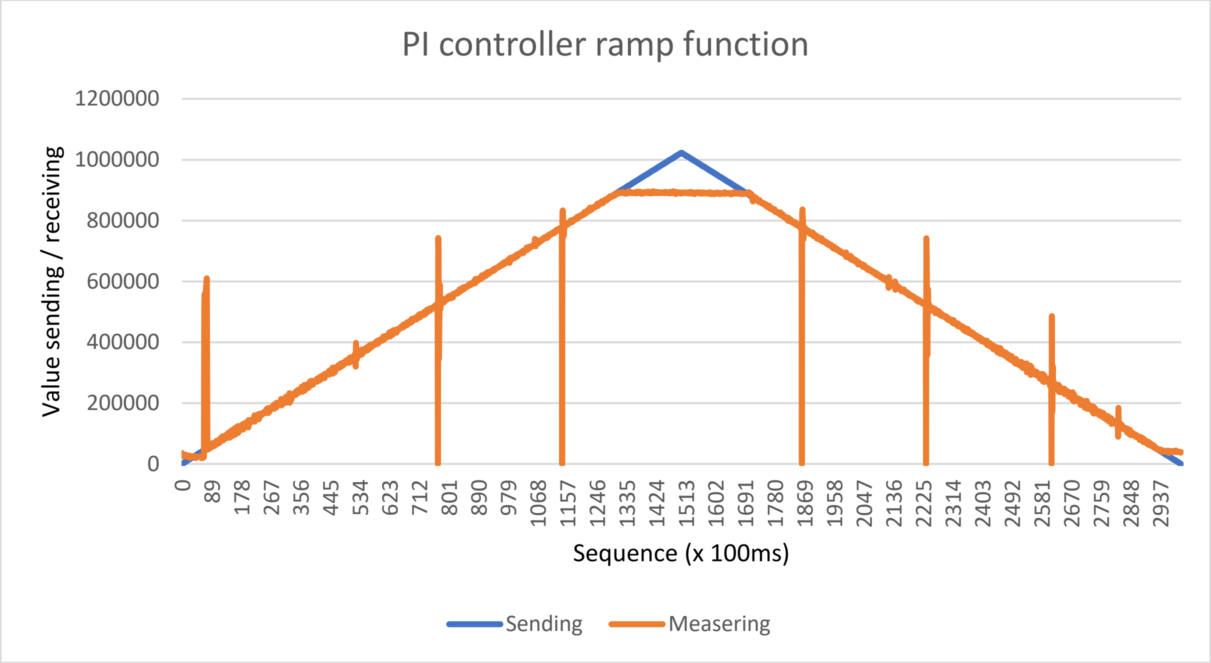
The following graphs represent the tests that were done with the experimental setup and with a configured PI controller. See the graph titles for more details. When information is written in brackets this means the PI settings (K / Ti / Td)

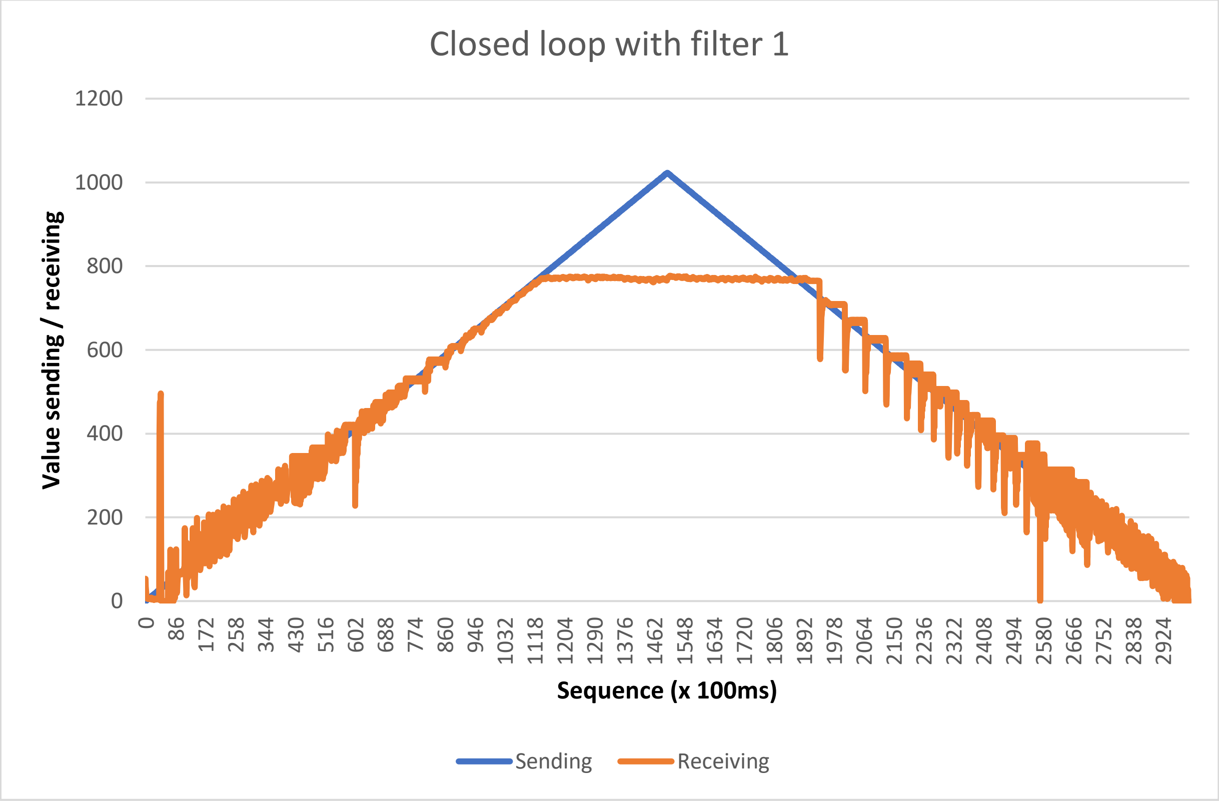


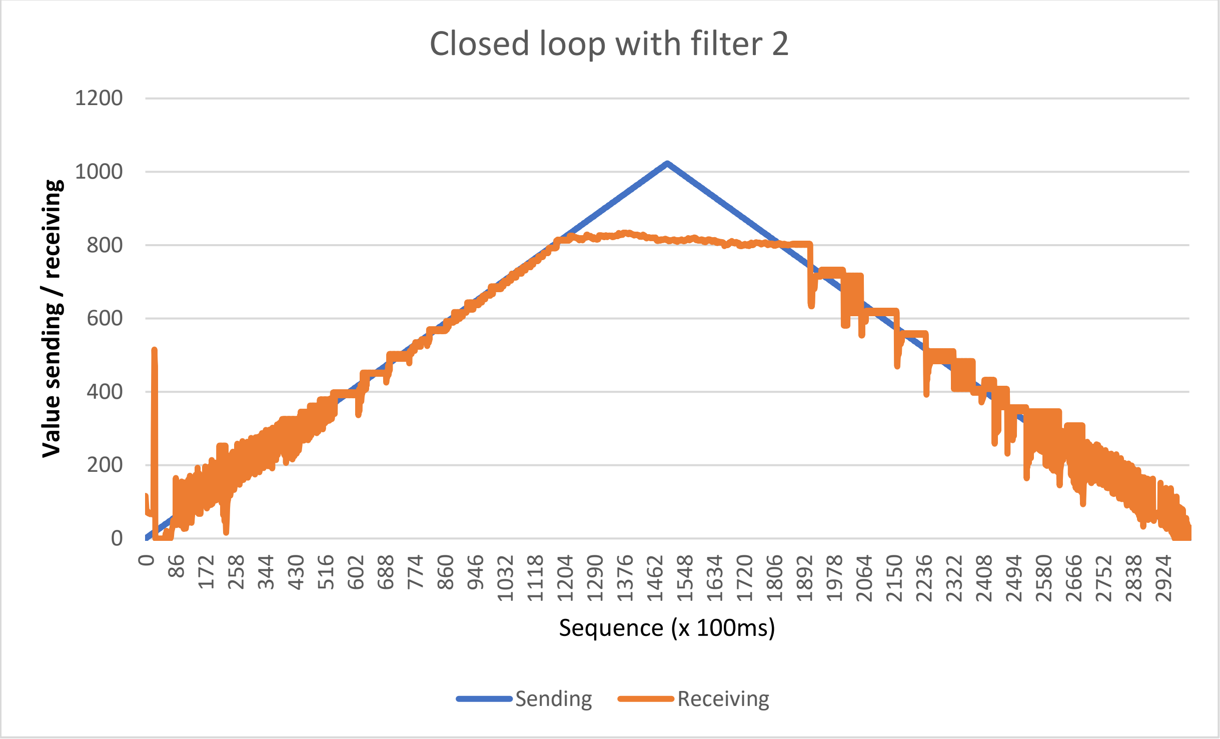


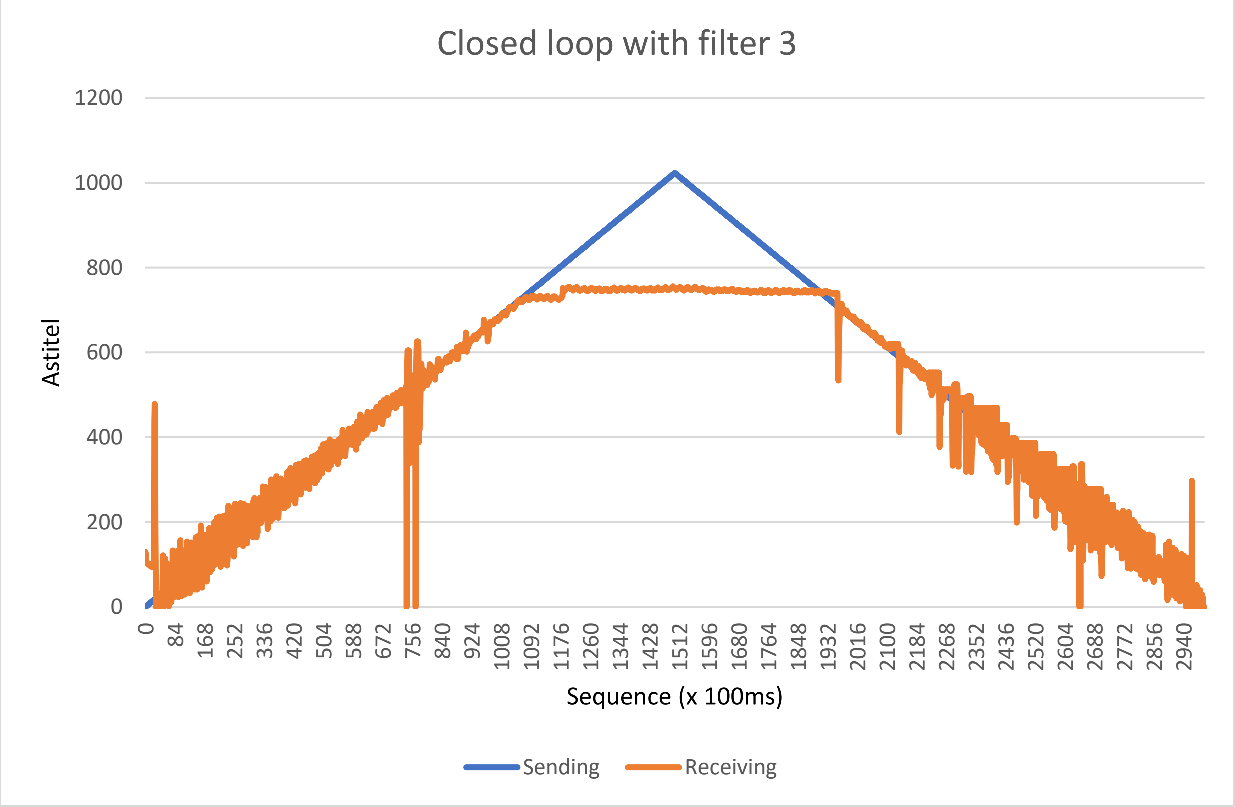

PI settings 1
1200
1000
800
600
400
200
0
Value sending / receiving
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence (x 100ms)
Sending
Receiving

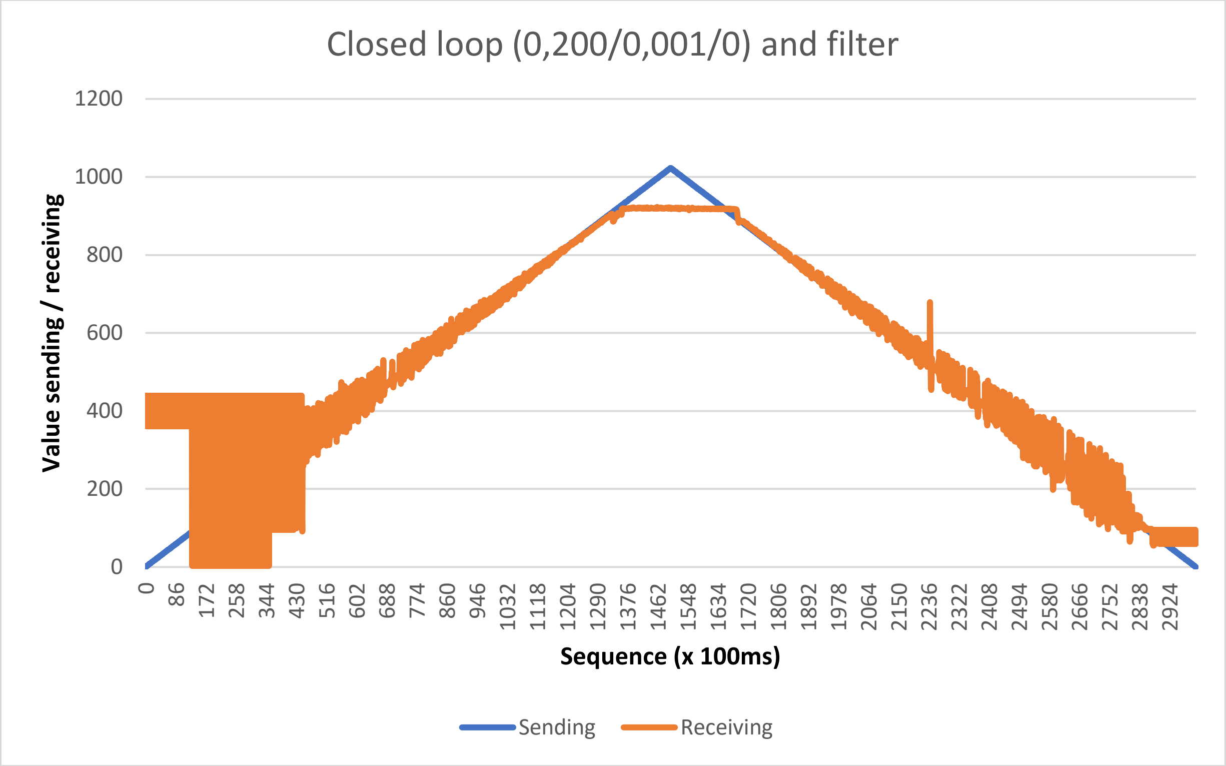

PI settings 2
1200
1000
800
600
400
200
0
Value sending / receiving
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence (x 100ms)
Sending
Receiving


PI controller ramp function
1200000
1000000
800000
600000
400000
200000
0
Value sending / receiving
0
89
178
267
356
445
534
623
712
801
890
979
1068
1157
1246
1335
1424
1513
1602
1691
1780
1869
1958
2047
2136
2225
2314
2403
2492
2581
2670
2759
2848
2937
Sequence (x 100ms)
Sending
Measering

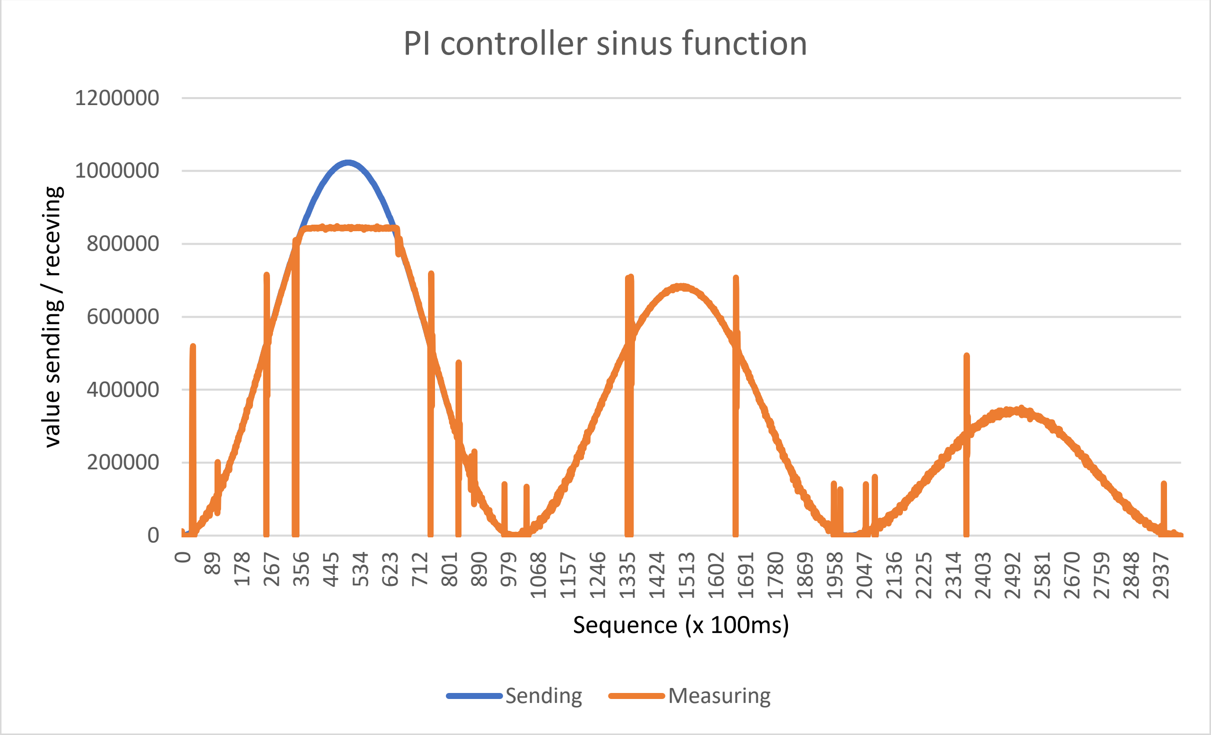

Closed loop with filter 1
1200
1000
800
600
400
200
0
Value sending / receiving
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence (x 100ms)
Sending
Receiving

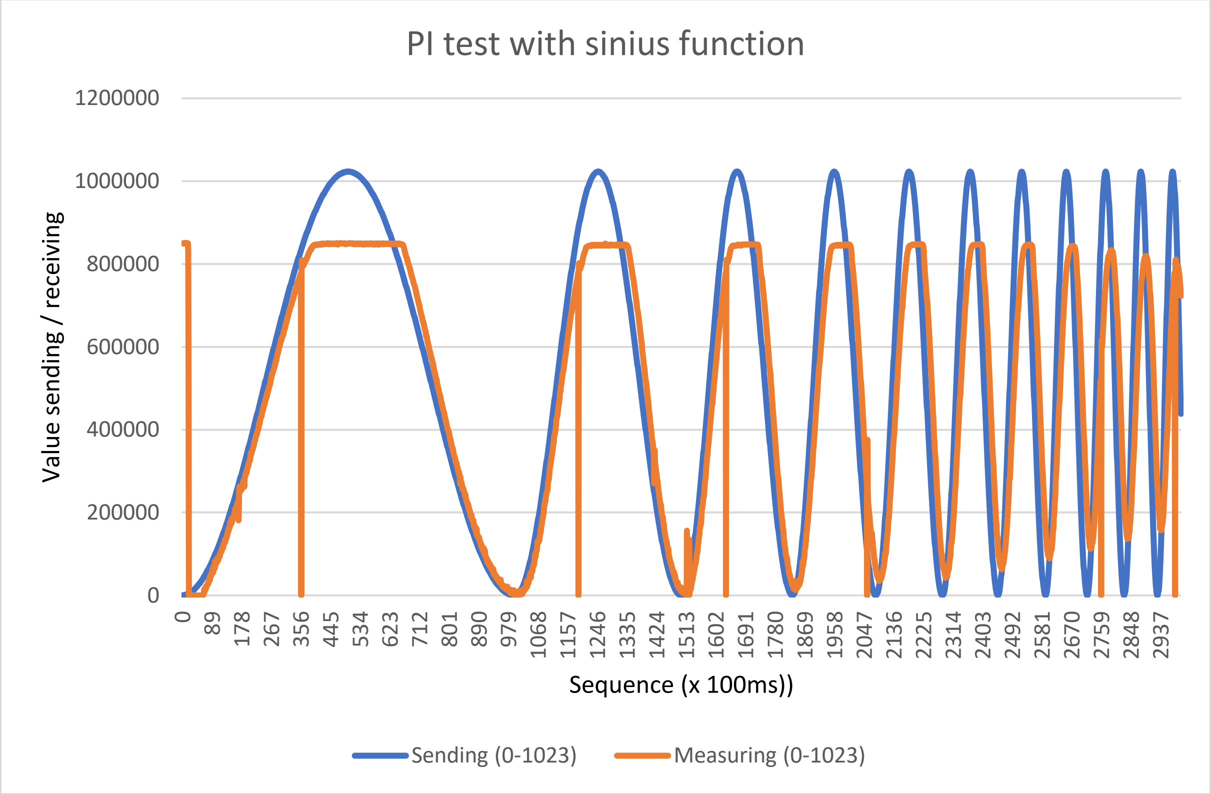

Closed loop with filter 2
Value sending / receiving
1200
1000
800
600
400
200
0
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence (x 100ms)
Sending
Receiving


Closed loop with filter 3
Astitel
1200
1000
800
600
400
200
0
0
84
168
252
336
420
504
588
672
756
840
924
1008
1092
1176
1260
1344
1428
1512
1596
1680
1764
1848
1932
2016
2100
2184
2268
2352
2436
2520
2604
2688
2772
2856
2940
Sequence (x 100ms)
Sending
Receiving

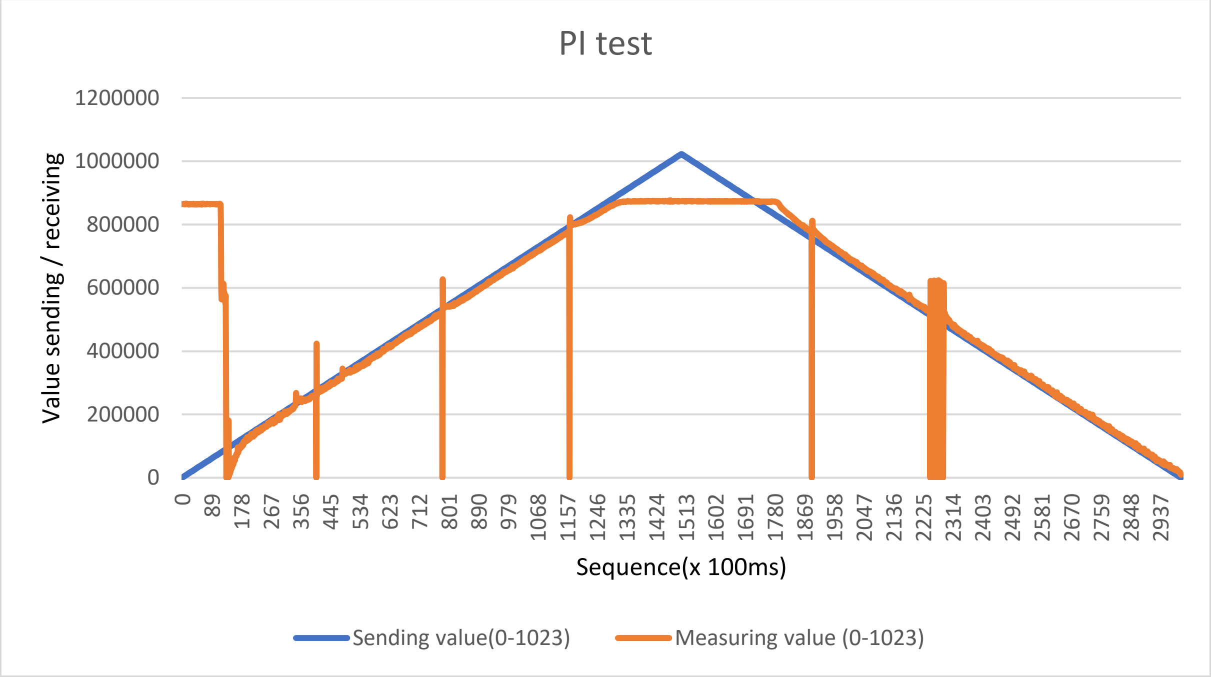

Closed loop (0,200/0,001/0) and filter
Value sending / receiving
1200
1000
800
600
400
200
0
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence (x 100ms)
Sending
Receiving

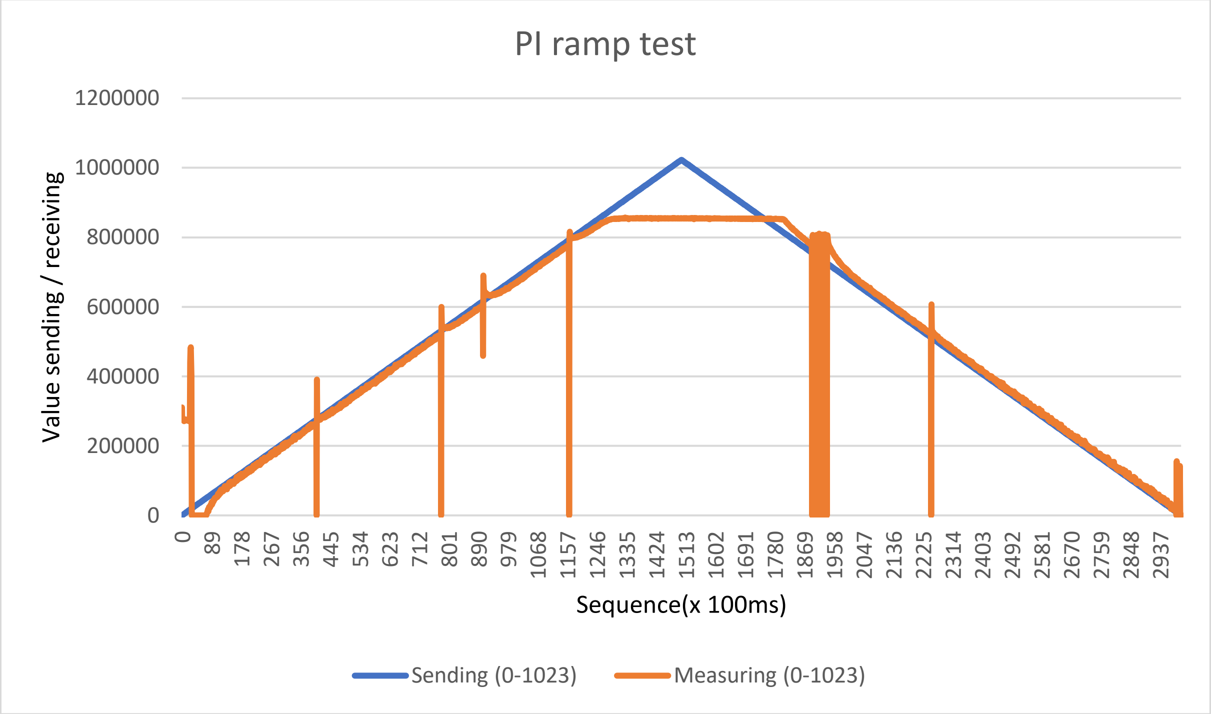
These are PI controller tests but instead of a ramp function, sinus functions are used as the input signal.

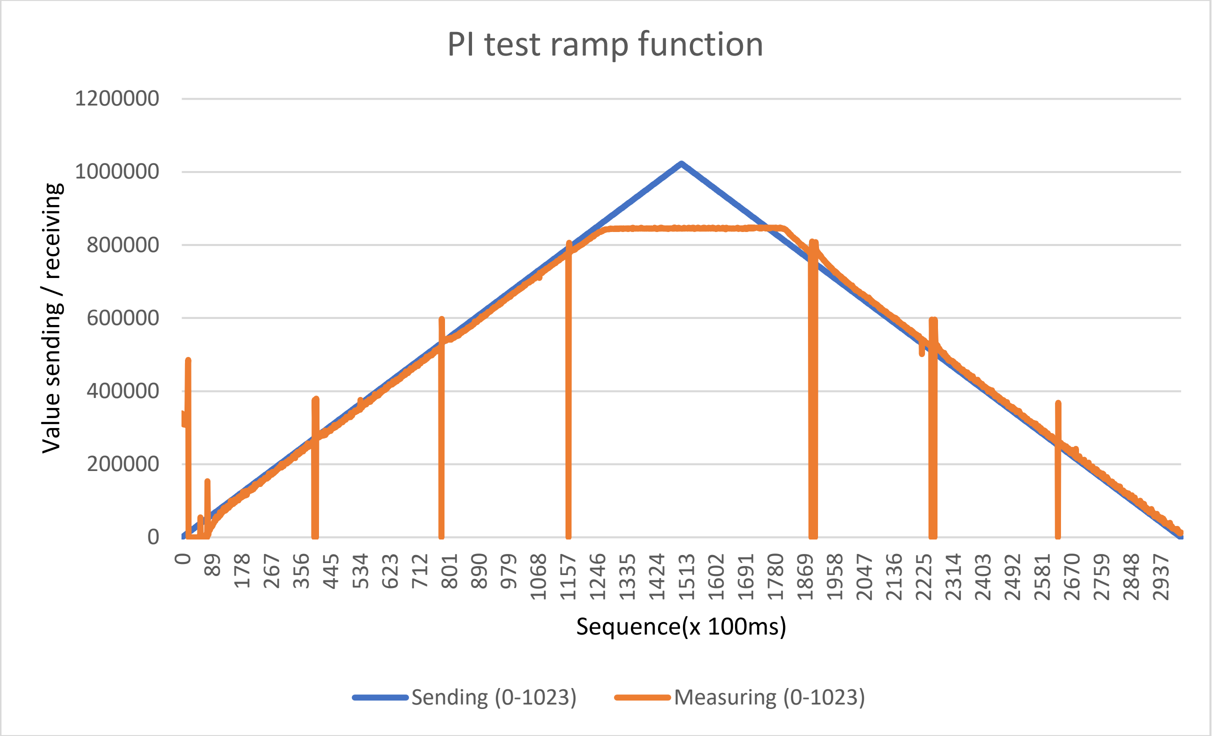




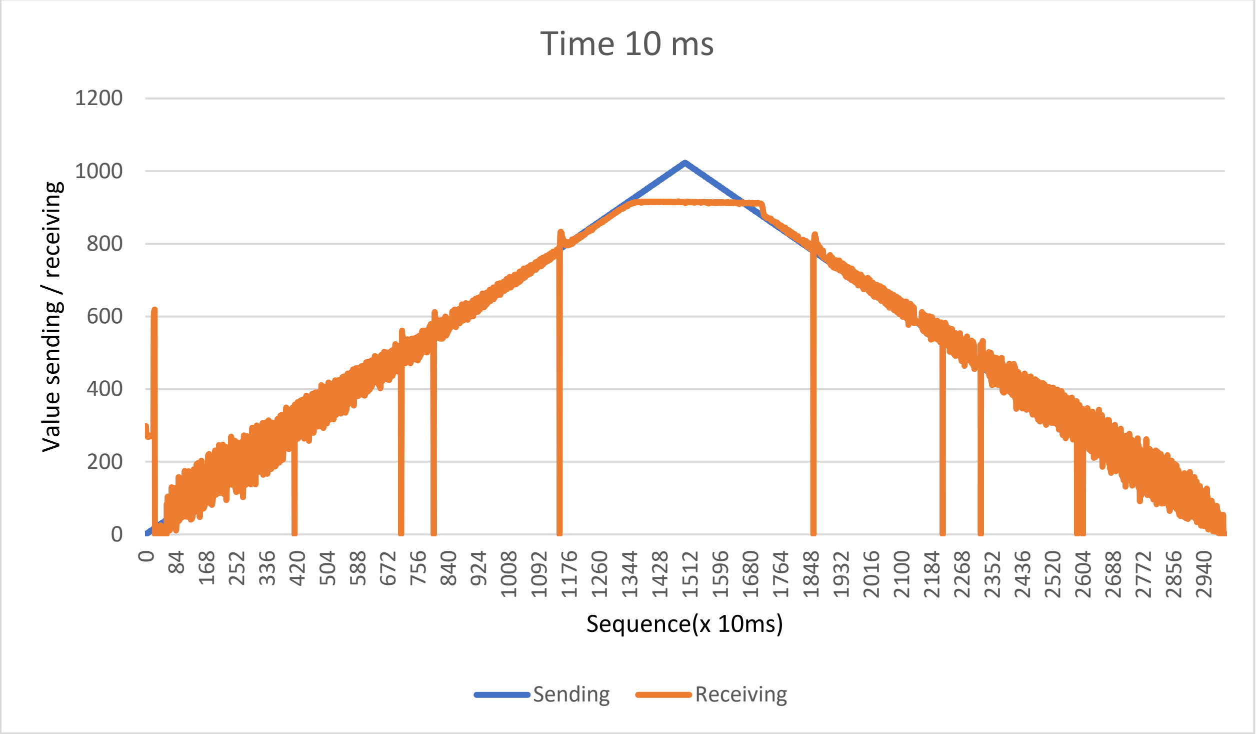
These graphs represent tests that were done with the best PI parameters that were found. Without the lost samples filter.

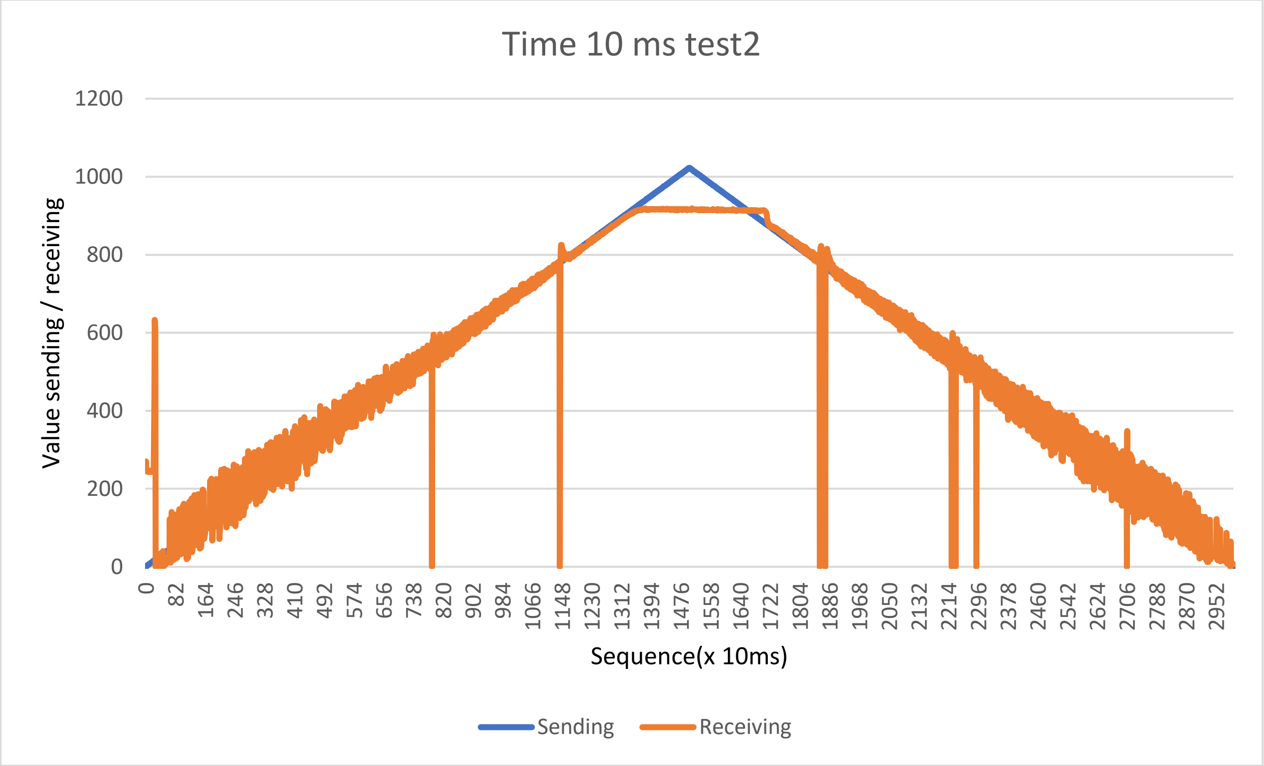


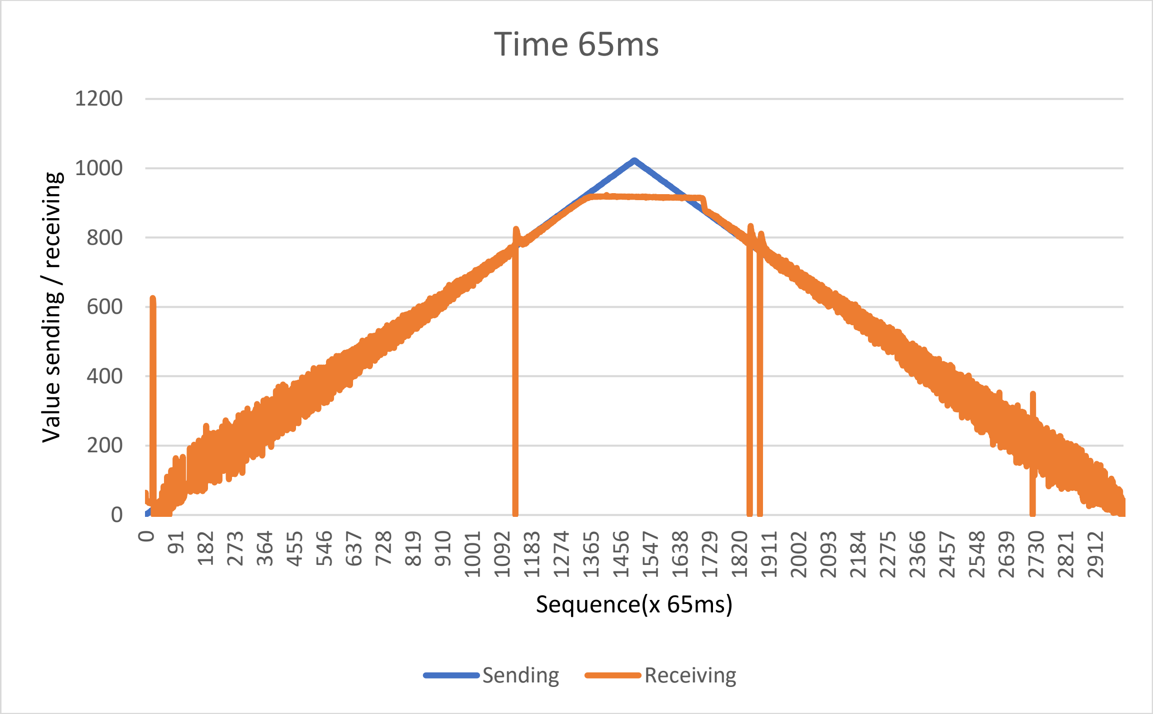


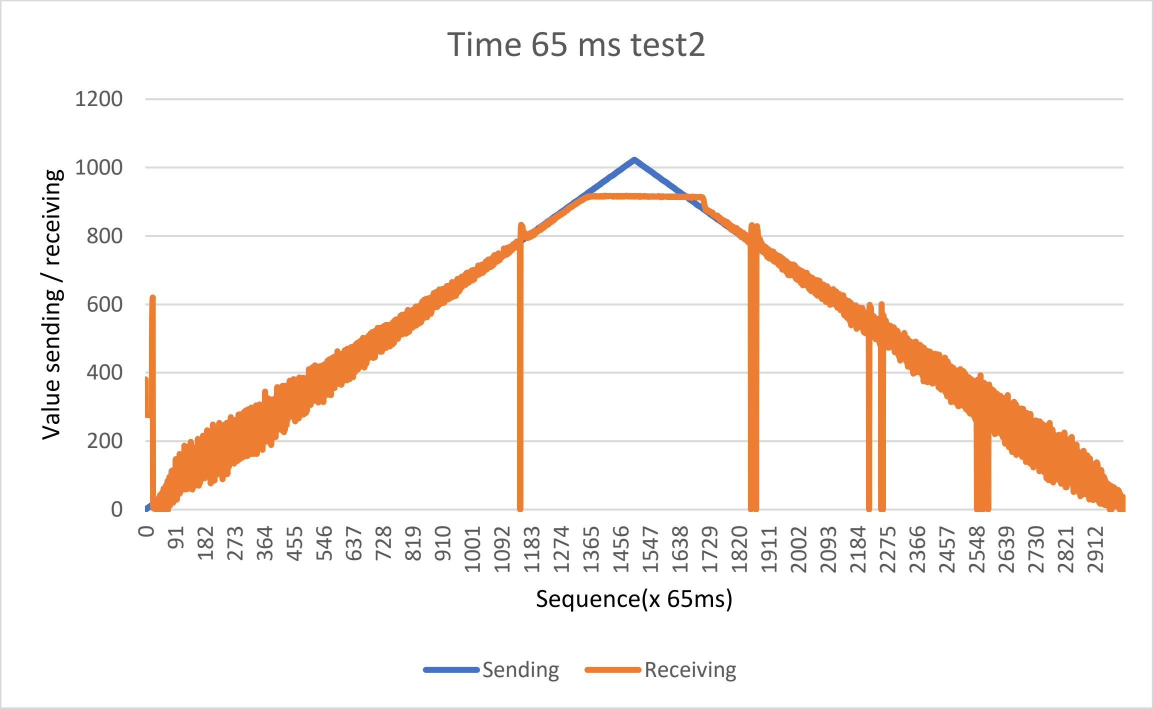

PI test ramp function
Value sending / receiving
1200000
1000000
800000
600000
400000
200000
0
0
89
178
267
356
445
534
623
712
801
890
979
1068
1157
1246
1335
1424
1513
1602
1691
1780
1869
1958
2047
2136
2225
2314
2403
2492
2581
2670
2759
2848
2937
Sequence(x 100ms)
Sending (0-1023)
Measuring (0-1023)

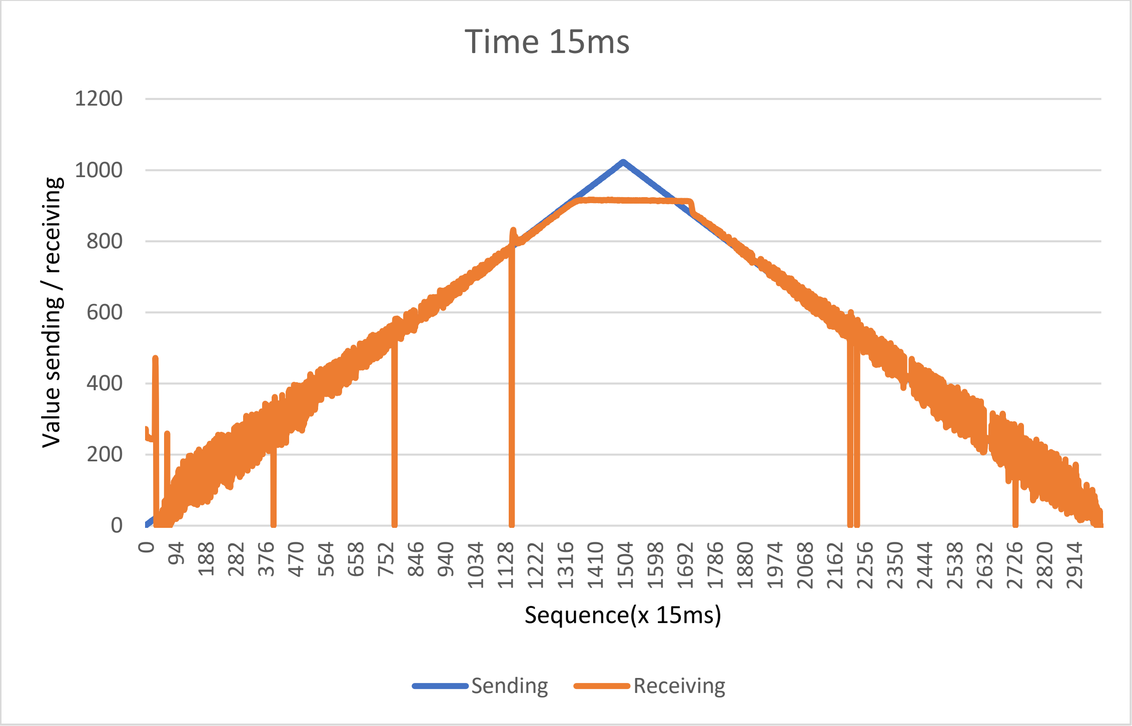
PI tests but at different loop times. To see what effect this has on the lost frames.

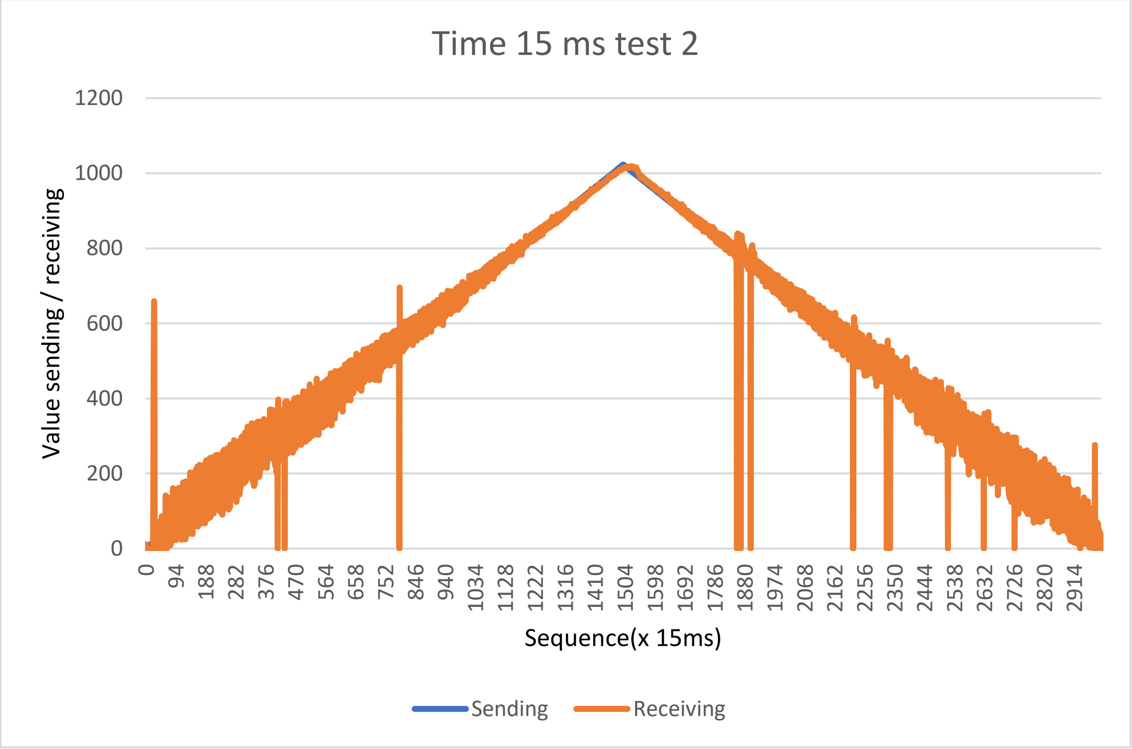




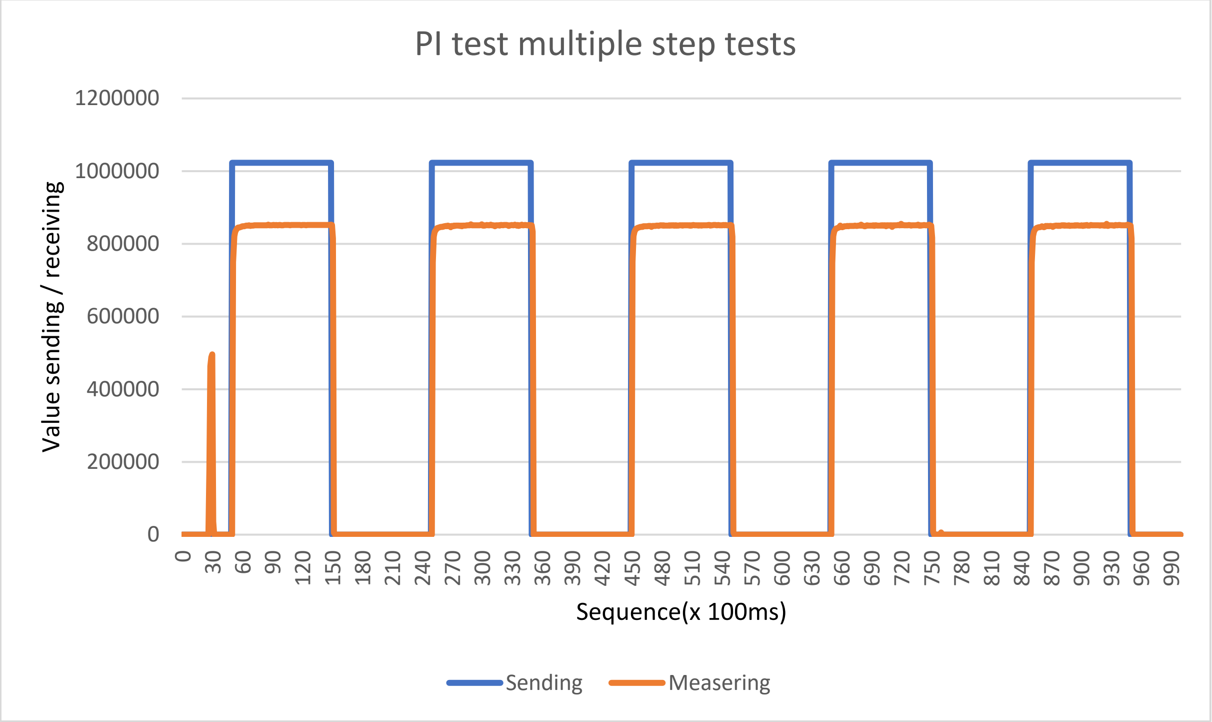

Time 65ms
1200
1000
800
600
400
200
0
Value sending / receiving
0
91
182
273
364
455
546
637
728
819
910
1001
1092
1183
1274
1365
1456
1547
1638
1729
1820
1911
2002
2093
2184
2275
2366
2457
2548
2639
2730
2821
2912
Sequence(x 65ms)
Sending
Receiving

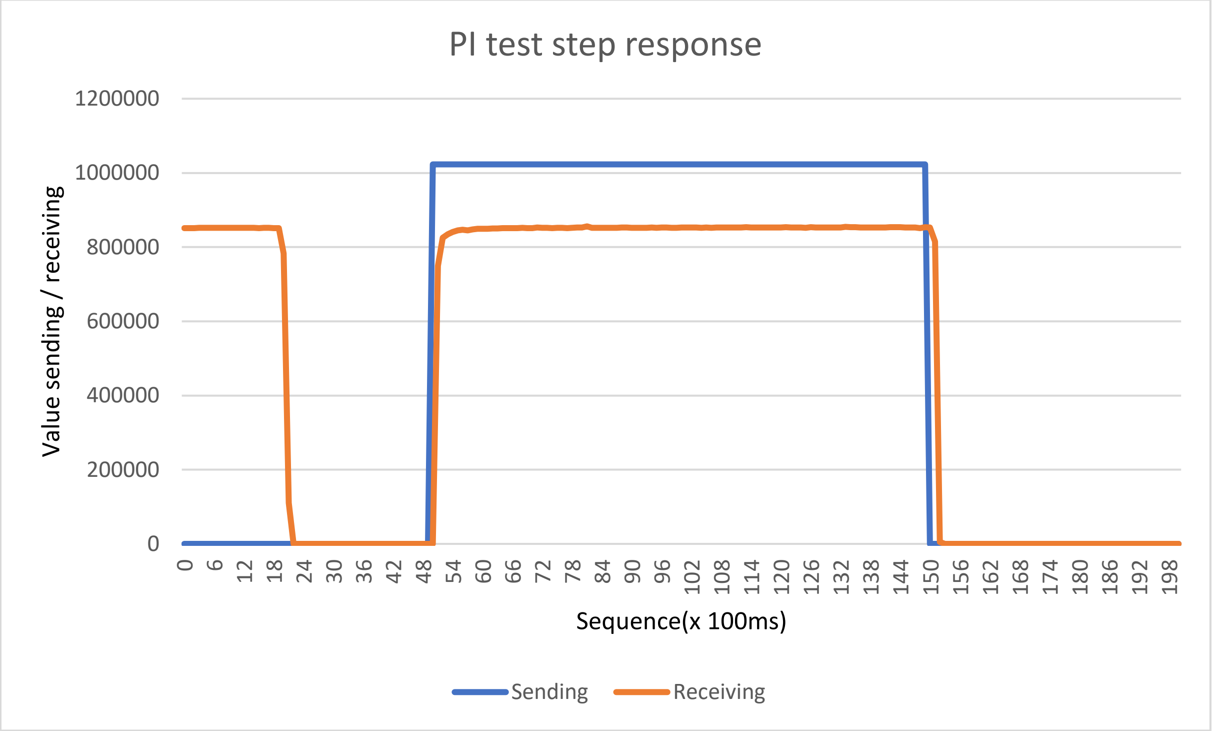

Time 65 ms test2
1200
1000
800
600
400
200
0
Value sending / receiving
0
91
182
273
364
455
546
637
728
819
910
1001
1092
1183
1274
1365
1456
1547
1638
1729
1820
1911
2002
2093
2184
2275
2366
2457
2548
2639
2730
2821
2912
Sequence(x 65ms)
Sending
Receiving


Time 15ms
Value sending / receiving
1200
1000
800
600
400
200
0
0
94
188
282
376
470
564
658
752
846
940
1034
1128
1222
1316
1410
1504
1598
1692
1786
1880
1974
2068
2162
2256
2350
2444
2538
2632
2726
2820
2914
Sequence(x 15ms)
Sending
Receiving

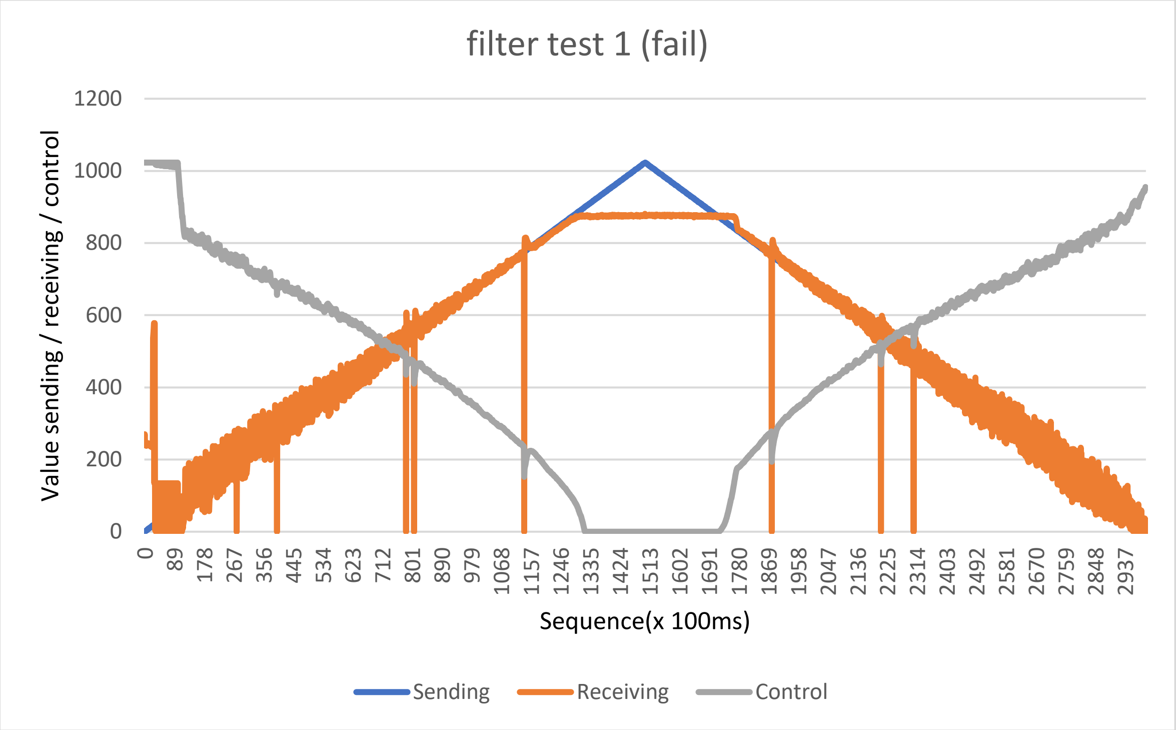

Time 15 ms test 2
Value sending / receiving
1200
1000
800
600
400
200
0
0
94
188
282
376
470
564
658
752
846
940
1034
1128
1222
1316
1410
1504
1598
1692
1786
1880
1974
2068
2162
2256
2350
2444
2538
2632
2726
2820
2914
Sequence(x 15ms)
Sending
Receiving

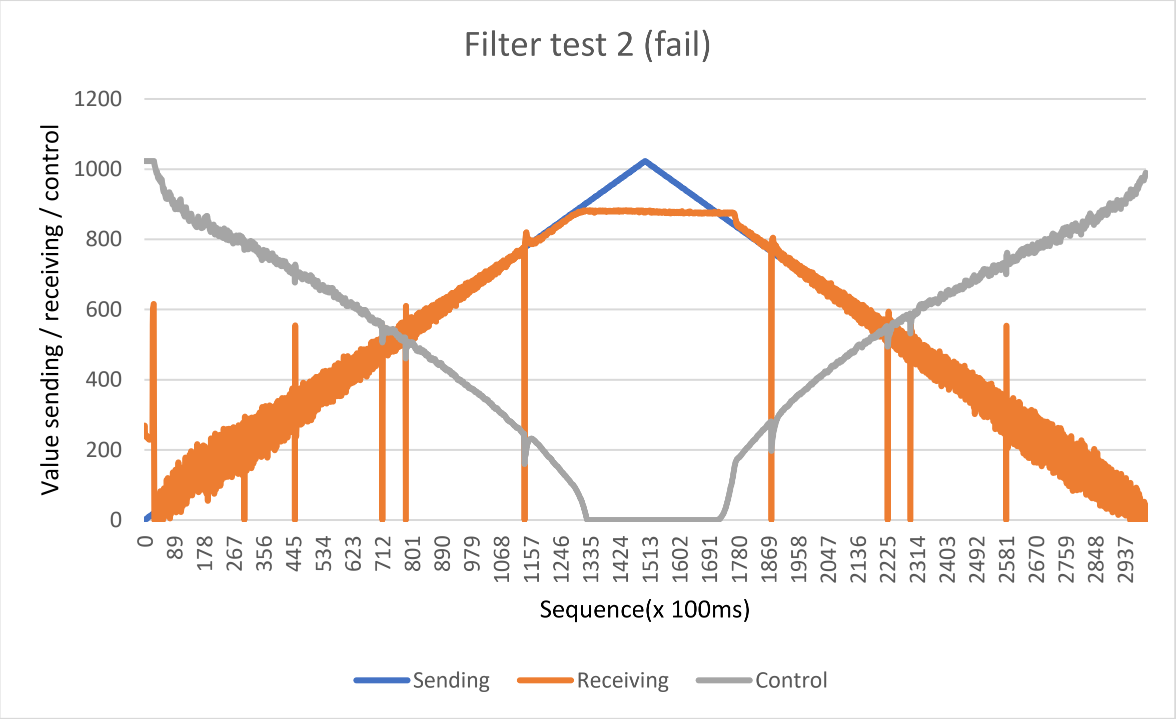
More step tests to see how fast the PI controller reacts.



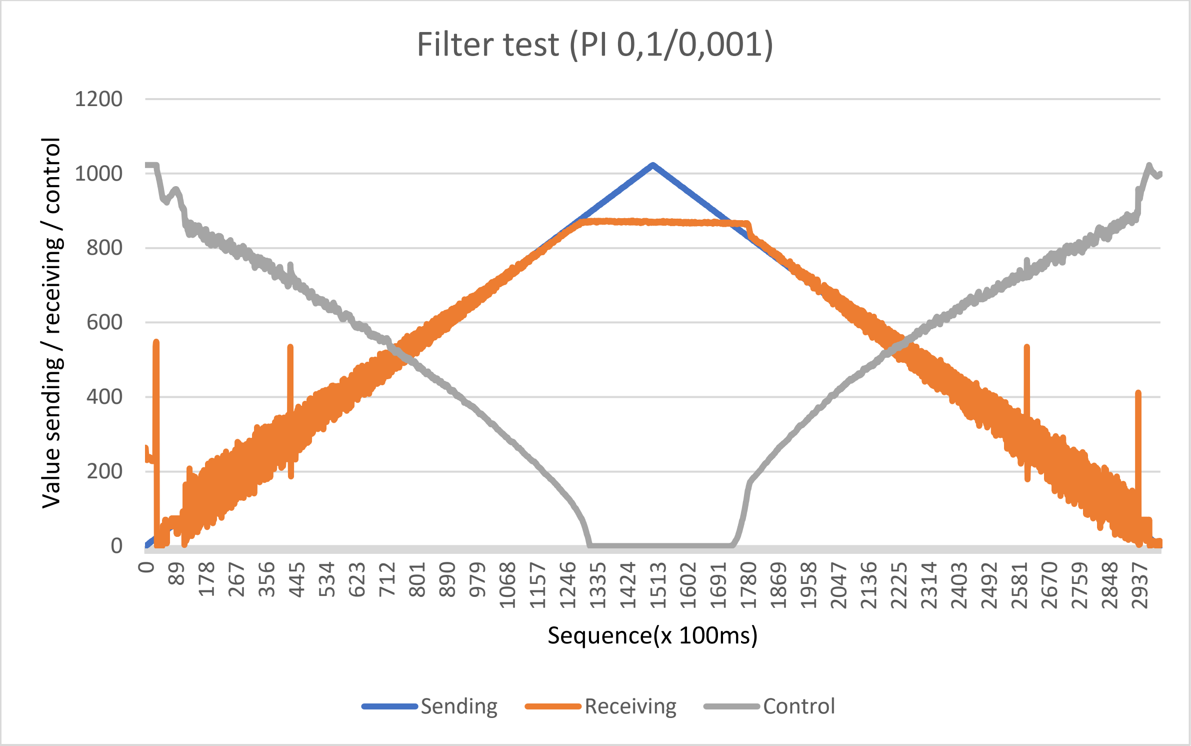


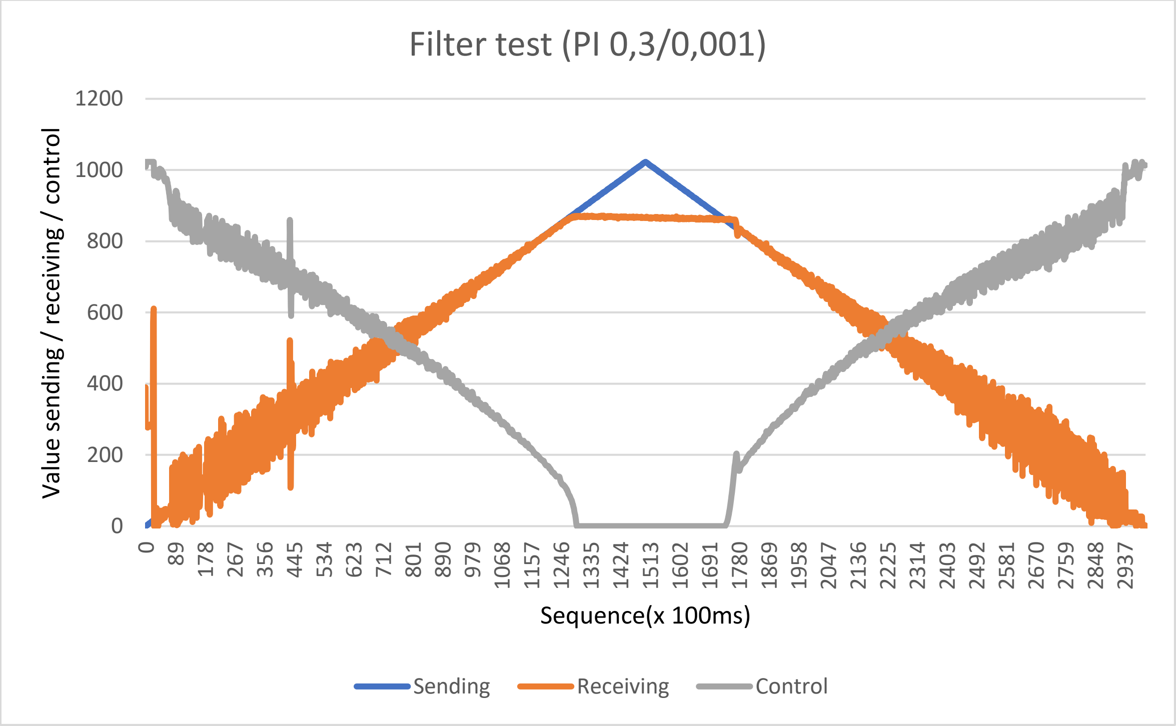
These graphs represent the tests that were done when the lost package filter was first implemented. The control signal is inverted so that it becomes easier to see on the graph, otherwise it would overlap with the receiving curve.

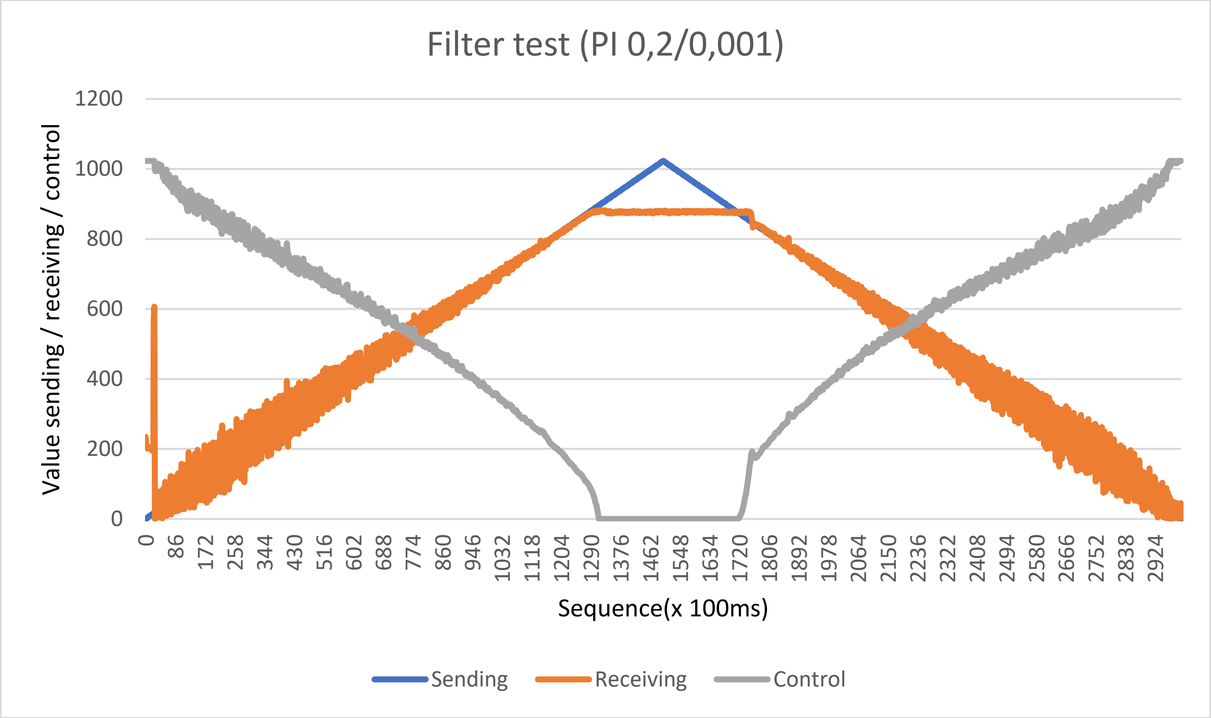




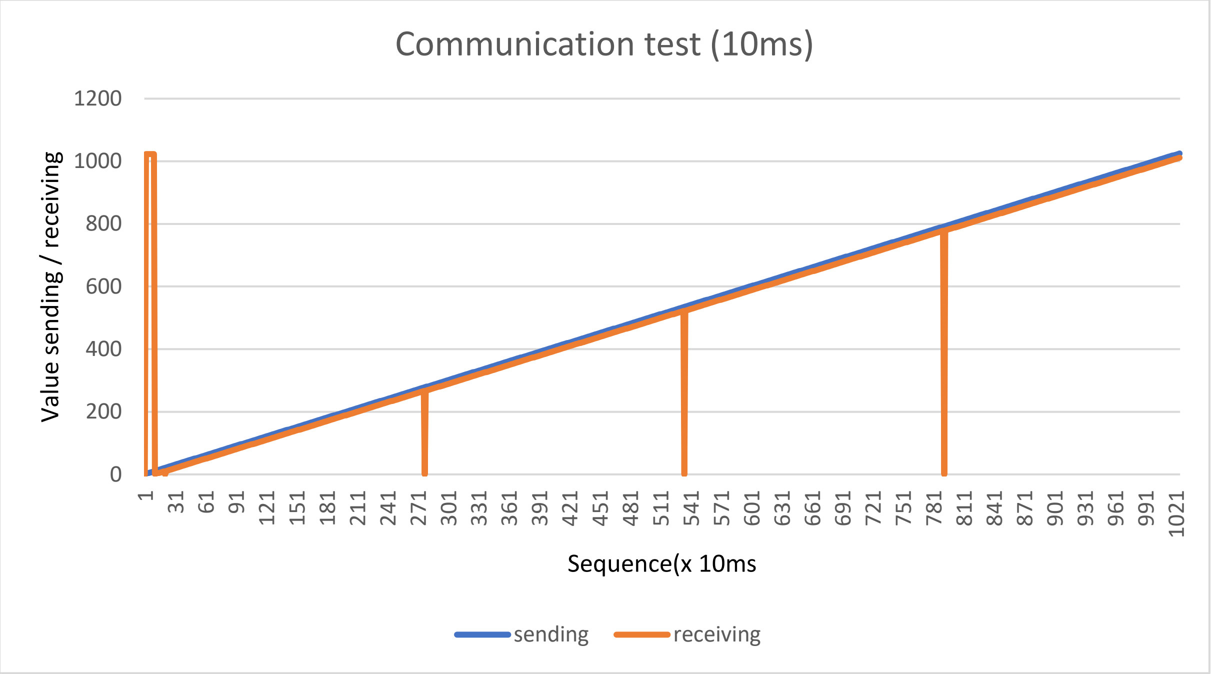
New control parameters had to be tested because the lost package filter changes the characteristics of the system.



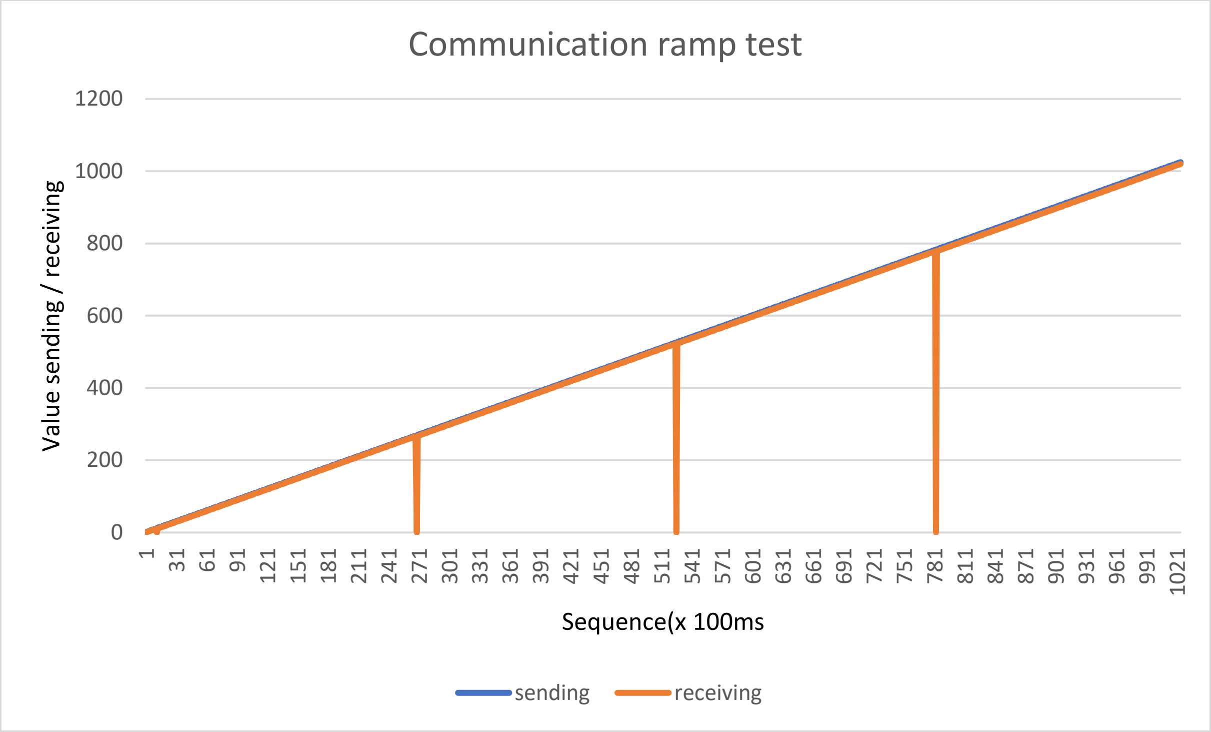



Filter test (PI 0,2/0,001)
Value sending / receiving / control
1200
1000
800
600
400
200
0
0
86
172
258
344
430
516
602
688
774
860
946
1032
1118
1204
1290
1376
1462
1548
1634
1720
1806
1892
1978
2064
2150
2236
2322
2408
2494
2580
2666
2752
2838
2924
Sequence(x 100ms)
Sending
Receiving
Control

These graphs represent the communication tests that were carried out with the use of the experimental setup. These tests were done by only using the myRIO device and the Arduino Uno microcontroller. In the tables the amount of samples in between a lost sample is recorded.

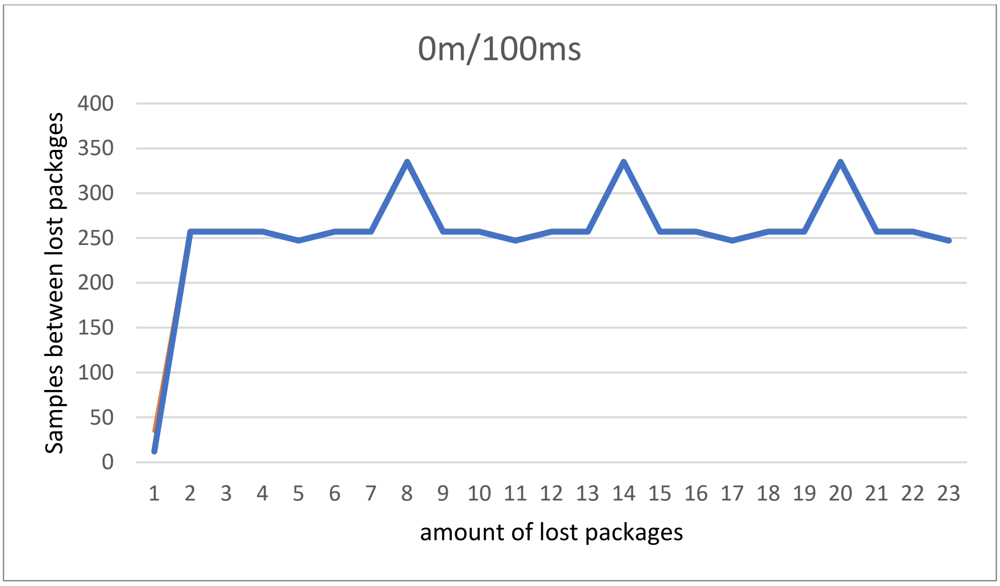


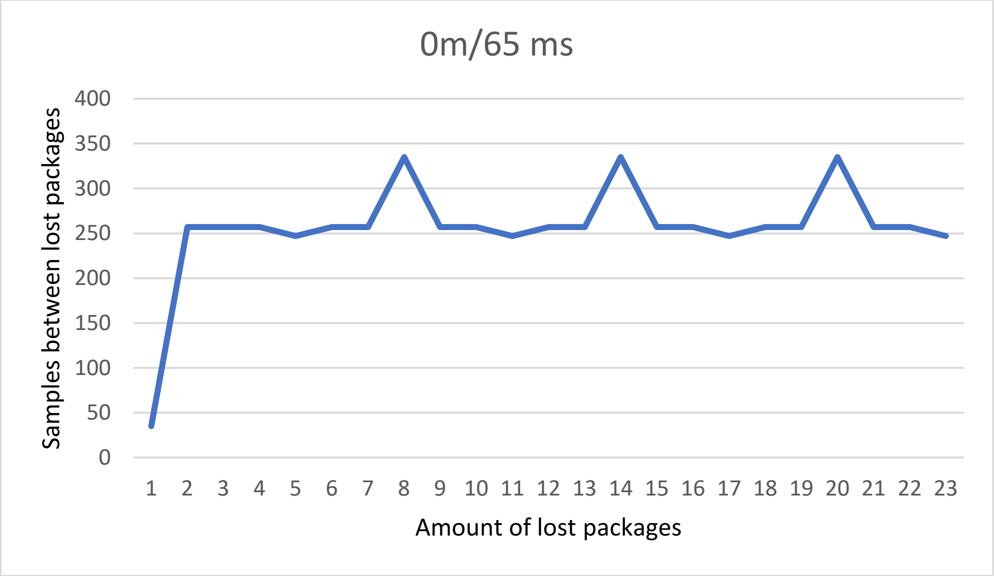
package lost	
Sample time	Difference
278	278
535	257
792	257

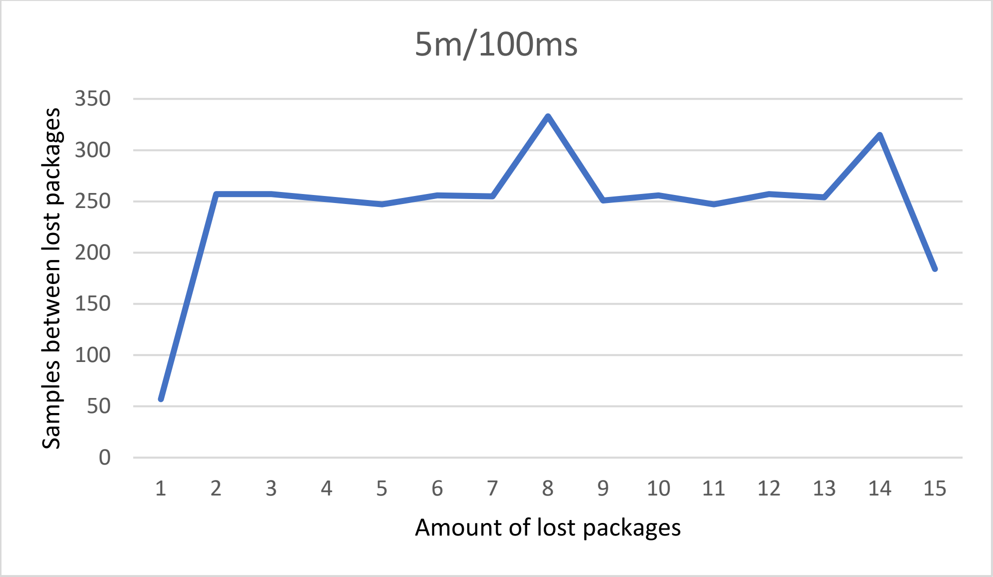


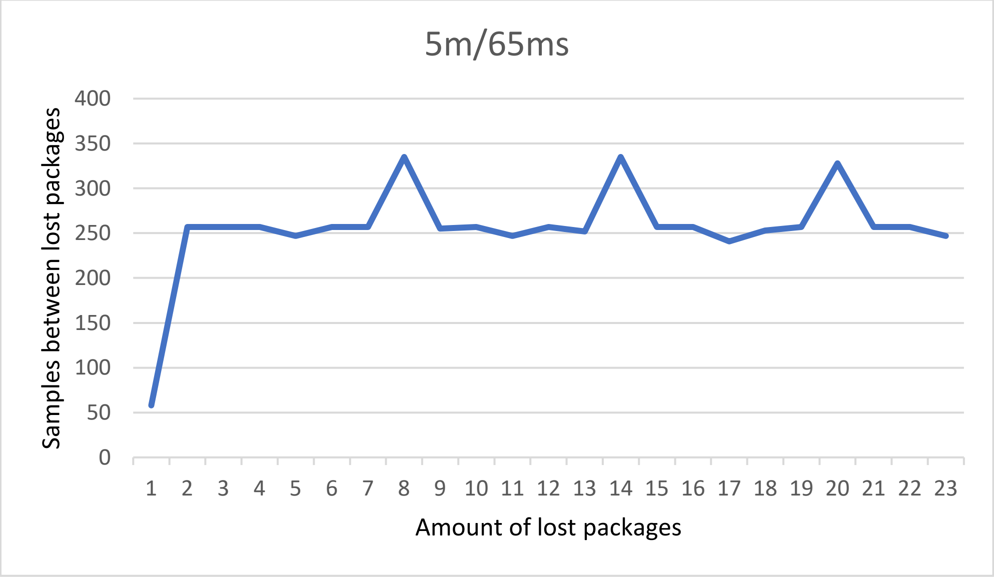
package lost	
Sample time	Difference
12	12
269	257
526	257
783	257

These graphs represent the amount of samples in between lost packages, the periodic nature of the problem can be clearly seen on these graphs.

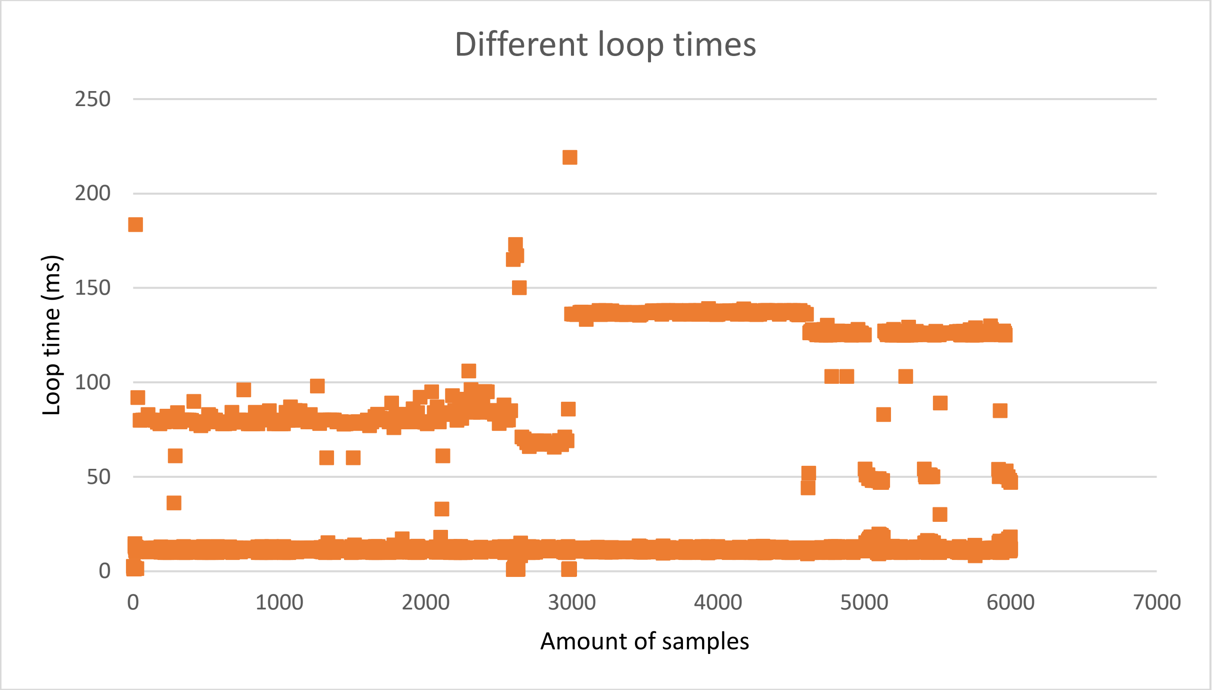




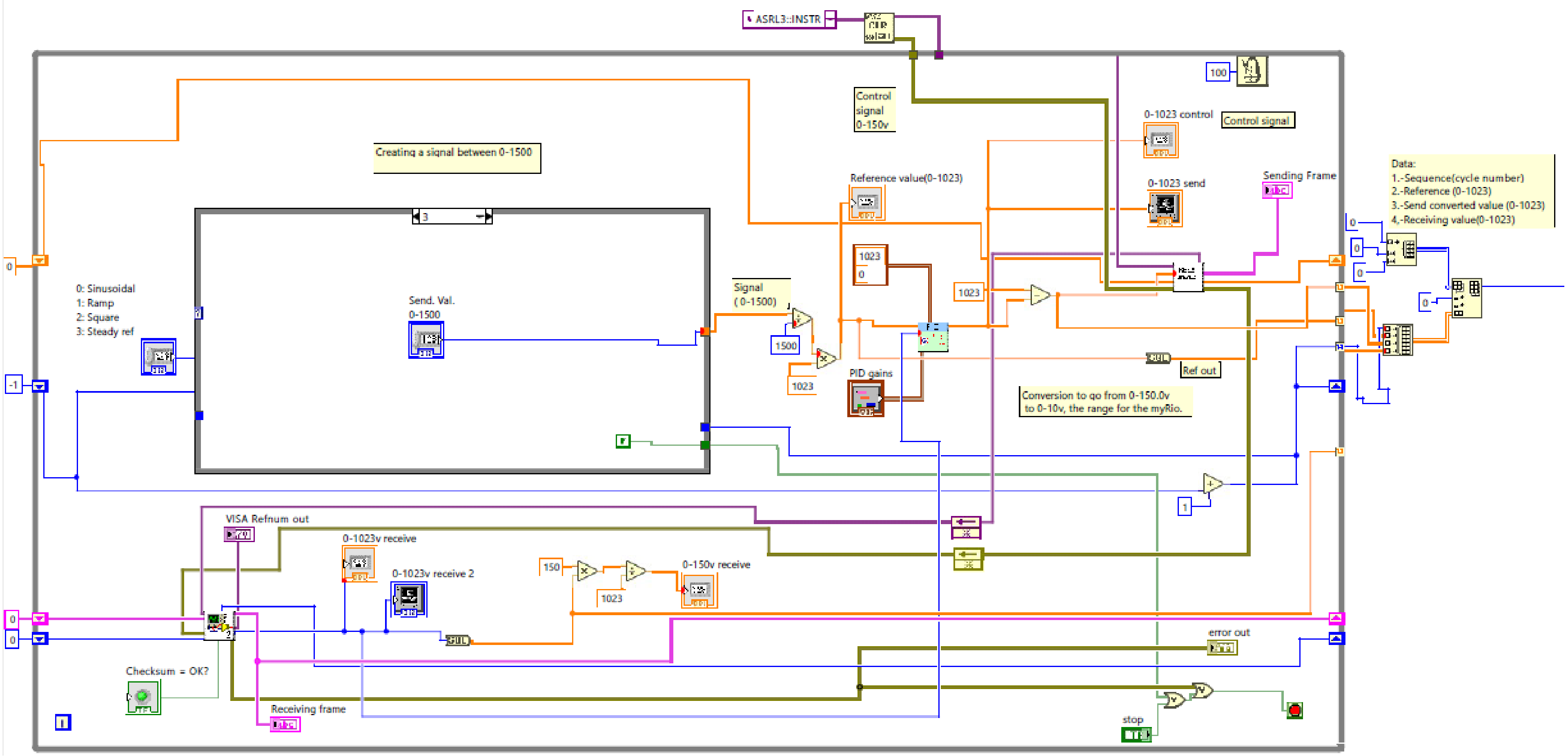

5m/100ms
Samples between lost packages
350
300
250
200
150
100
50
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Amount of lost packages


5m/65ms
Samples between lost packages
400
350
300
250
200
150
100
50
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Amount of lost packages

In this graph the LabVIEW program would send packages as fast as it could, the times you see here is the time it takes for LabVIEW to construct a frame. The average time needed was around 20 ms.



This is the global LabVIEW program, this is only to give a clearer view than in chapter 5.


ASRL3::INSTR
Creating a signal between 0-1500
Control signal 0-150v
100
0-1023 control
Control signal
Reference value(0-1023)
0-1023 send
Sending Frame
Data:
1.-Sequence(cycle number)
2.-Reference (0-1023)
3.-Send converted value (0-1023)
4,-Receiving value(0-1023)
0: Sinusoidal
1: Ramp
2: Square
3: Steady ref
3
Send. Val.
0-1500
Signal
(0-1500)
1500
1023
1023
0
1023
PID gains
Ref out
Conversion to go from 0-150.0v
to 0-10v, the range for the myRio.
-1
0
1
VISA Refnum out
0-1023v receive
0-1023v receive 2
150
1023
0-150v receive
error out
Checksum = OK?
Receiving frame
stop