ELEKTRONICA-ICT

Project Onderzoek 2020-2021

Met opmerkingen [Bart Stuk1]: Ga naar http://poz.pxl-ea-ict.be/qr.php en genereer de juiste QR-code. Voeg die hier in en zet je juiste URL in de voettekst, zowel pagina 1 als de andere (eerste pagina is niet gelinkt, dus 2x te doen).

What is the most efficient way to control hardware by software with given input from the AI?

Author *Jesse Everts*
Olaf van Beers
Ralph Moeskops
Jayson Gorissen
Bram Willems
Company PXL

Abstract

Met opmerkingen [Bart Stuk2]: Minimaal 150 woorden en maximaal 300 woorden.
Te schrijven tegen fase 3 en 4 (Bachelorproef).

This Application Note describes the research and development done for the communication between all the different devices and services used in the "*How do you feel today*" art project. The "*How do you feel today*" project consists of many different parts which need to be integrated with each other to function correctly. All the steps taken to ensure the best and most efficient communication possible between all these different modules is described in this AN. The communication problem was solved by using a message queue protocol on a microcontroller that communicates with other modules over Wi-Fi. All the research and development has been done with the research question in mind: "What is the most efficient way to control the hardware using the data given by the Facial recognition AI?". There are discussions available of what communication system to use, which microcontroller to utilize and which kind of protocol to use for data transfer. The research question of this project has been answered as can be read in this document. This application note will be a guide through the communication section of the "*How do you feel today*" art project.

Content

Met opmerkingen [Bart Stuk3]: Alle genummerde titels komen hier. Beperk in overdosis van titels en maximale aangeraden diepte is 3 niveaus, maar meeste kunnen zich best beperken tot 2 niveaus.

1 Introduction

Met opmerkingen [Bart Stuk4]: Minimaal 150 woorden en aangeraden 300 woorden (meer mag).
Te schrijven tegen fase 1 en 4 (Bachelorproef).

Mr. Stukken introduced us into the project so we could start with the project, he is also the scrum master. Mr. Martens has also helped with a lot of information, he is the scrum owner. Other teachers Like Mr. Vanrykel have helped us with his expertise on PSoC.

Our project is mostly of the software aspects of the art piece. We have researched which kind of communication is better, what microcontroller suited us the most, looked how we parsed data the best. We have made some example code to be ensured that when we effectively start building it is going to be good on the first try.

Our project also follows the aspects of the SMART criteria. Every letter of SMART has a meaning which helps us to complete our main objectives and not to get side-tracked.

In this application note you will first read our used material and methods after that you can find our results we found. Afterwards you will find discussion we will discuss our found results and question the validity of it. Next up you will find the conclusion. And as last two you can find our reference list and added attachments.

2 Material and methods

Met opmerkingen [Bart Stuk5]: Minimaal 750 woorden en aangeraden 2000 woorden (meer mag).
Te schrijven tegen fase 1 (Bachelorproef).

2.1 Wired VS Wireless

2.1.1 Wired

2.1.1.1 Ethernet

Ethernet is commonly used in local area networks (LAN), metropolitan area networks (MAN) and wide area networks (WAN). Ethernet supports high bit rates, a great number of nodes and long link distances.

Over the course of its history ethernet data transfer rates have been increased from the original 2.94 Mbit/s to the latest 400 Gbit/s. The Ethernet standards comprise several wirings and signalling variants of the OSI physical layer in use with Ethernet. [1]

Ethernet is widely used in homes and industry. It interworks well with wireless Wi-Fi technologies. The Internet Protocol is commonly carried over Ethernet and so it is considered one of the key technologies that make up the Internet.

2.1.1.2 USB

Universal Serial Bus (USB) is an industry standard that establishes specifications for cables and connectors and protocols for connection, communication, and power supply for/between computers. A broad variety of USB hardware exists, including eleven different connectors, of which USB-C is the most recent. [2]

In table beneath you will find the USB transfer speeds of the current generation

Table 1: USB speeds

Specification	Data rate	Transfer speed
USB 3.0	5 Gbit/s	500 MB/s
USB 3.1	10 Gbit/s	1.21 GB/s
USB 3.2	20 Gbit/s	2.42 GB/s

2.1.1.3 Differences

1. USB is an interface for connecting peripheral devices while Ethernet is an interface for networking.
2. USB has a much shorter range than Ethernet.
3. USB provides power while Ethernet does not.
4. Ethernet is much Faster than USB.

2.1.2 Wireless

2.1.2.1 Wi-Fi

Wi-Fi is a for wireless data networks that work according to the international standard IEEE 802.11 (wireless ethernet or Wi-Fi). Such products use radio frequencies in the 2.4GHz and / or 5.0GHz bands.

This network can connect with a high number of devices. Wi-Fi can be used for many different devices. The connection can be secured. This connection can reach up to 100 meters.

2.1.2.2 Bluetooth

Bluetooth is a wireless technology standard used for exchanging data between fixed and mobile devices over short distances using UHF radio waves in the industrial, scientific and medical radio bands, from 2.402 GHz to 2.480 GHz.

It is a standard wire-replacement communications protocol primarily designed for low power consumption, with a short range based on low-cost transceiver microchips in each device. Because the devices use a radio communications system, they do not have to be in visual line of sight of each other. Range is power-class-dependent, but effective ranges vary in practice. Typically, less than 10 meters. [3]

2.1.2.3 Differences

In the table beneath you will find the general differences between Wi-Fi and Bluetooth.

Table 2: Difference between Wi-Fi and Bluetooth

Part:	Wi-Fi	Bluetooth
Bandwidth	High	Low
Hardware requirements	Wireless adapter on all devices within the network and a router	Bluetooth-adapter on all devices you want to link
User friendly	Configuration is complex	Fairly easy to use
Range	100 meters	10 meters
Safety	Secure but still some risks	Less secure
Energy consumption	High	Low
Frequency	2.4 Ghz and 5 Ghz	2.402 Ghz and 2.480 Ghz
Flexibility	Many devices support it	Not many devices support it

2.1.3 Pros & cons of using Wired VS Wireless

In the table beneath you will see then general pros and cons for wired communication in comparison with wireless communication.

Table 3: Pros and Cons of using Wired VS Wireless

Pros	Cons
Controllability	Maintenance
Safety	Cable management
Speed	Mobility

2.2 Microcontroller

We need to use a microcontroller to receive, process and forward all the data that our team receives. One of the first things to do was look for a suitable microcontroller. A number of different microcontrollers have been compared with different specifications.

2.2.1 ZYNQ

For the ZYNQ we looked at the ZYNQ-7000 SOC. This microcontroller has powerful dual processors. There is also a possibility to run parallel codes. If this sign were to be used, it would mean programming in Vivado. Because bitstreams have to be generated, testing cannot be done quickly and easily. There is also no Wi-Fi on this controller, which is necessary for a wireless connection. In addition, there is also no possibility to program Python on this microcontroller. A brief overview can be seen below in table 4.

Table 4: Pros and Cons ZYNQ

Pros	Cons
Parallel control	No onboard Wi-Fi
Good hardware and software security	Vivado / takes a long time
Flexible	No Python
	High cost

2.2.2 ESP

There has been looked at a ESP32. An ESP controller is a small and inexpensive board. This makes this microcontroller well suited for small calculations. Python can be put on an ESP32, but this is MicroPython. This means that Python 3 works on this and that a small number of standard libraries are optimized for the ESP. A brief overview can be seen below in table 5.

Table 5: Pros and Cons ESP

Pros	Cons
Wi-Fi on board	Not a full-fledged python
Low cost	Little processing power

2.2.3 PYNQ

PYNQ is an open source project from Xilinx that makes it easier to use Xilinx platforms. The Python libraries can be used for this. PYNQ itself is not a micro controller but uses the original ZYNQ boards. This means that the ZYNQ shares the same advantages and disadvantages. A brief overview can be seen below in table 6.

Table 6: Pros and Cons PYNQ

Pros	Cons
Parallel control	No onboard Wi-Fi
Good hardware and software security	Vivado / takes a long time
Flexible	High cost
Python	

2.2.4 PSoC 6 wifi-BT pioneer kit

The PSoC 6 Wi-Fi-BT Pioneer kit from Cypress contains a powerful processor. You can program with Python by using Zerynth Studio. This is not the full version of Python. Some libraries can't be installed. In addition, the PSoC 6 includes Wi-Fi for wireless connections to other controllers or networks. Because the PSoC does not contain an operating system, it will be faster. A brief overview can be seen below in table 7.

Table 7: Pros and Cons PSoC 6

Pros	Cons
Wi-Fi on board	High cost
Low power	
Python mixed with C	
Enough processing power	
No operating system	

2.2.5 Raspberry Pi

A Raspberry contains Wi-Fi, which means that wireless communication is also available here. A Raspberry contains its own operating system. As a result, a Raspberry works slower than a micro controller without an operating system. A Raspberry is able to execute multiple processes at the same time. It also includes the full version of Python with an option to install all libraries. The price of a Raspberry Pi is in the middle when compared to prices of the other compared products. A brief overview can be seen below in table 8.

Tabel 8: Pros and Cons Raspberry Pi

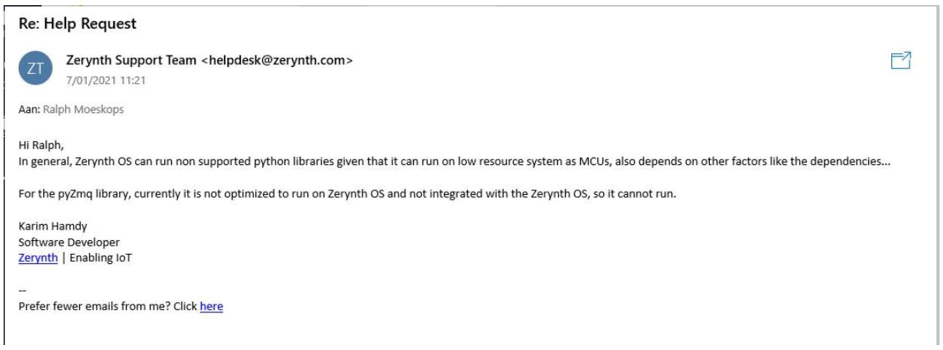
Pros	Cons
Wi-Fi on board	Average cost
Parallel control	Operating system (Linux)
Python	
Enough processing power	
Average cost	

2.3 Message queue protocol

2.3.1 ZeroMQ

ZeroMQ (also known as ØMQ, 0MQ, or zmq) is a lightweight, asynchronous message queueing protocol with TCP and UDP possibilities. ZeroMQ can run without a dedicated message broker and supports common messaging patterns like pub/sub, request/reply and client/server. In this project the desired message pattern is pub/sub and thus zmq will support the needs of this project. The ZeroMQ library is easily installable for Python on MAC, Linux and Windows requiring Python pip to be installed.

However, since the chosen language to write this project in is Python, some issues with implementation occur on the PSoC 6. The PSoC 6 uses a modified version of Python which requires libraries to be imported using .py files. Since the zmq.py file does not exist the implementation of ZeroMQ in python on PSoC 6 is impossible to our knowledge. Like you can see in the following image, a developer of the Zerynth Studio confirmed that ZeroMQ can indeed not be run on the PSoC 6 in python.



Re: Help Request

Zerynth Support Team <helpdesk@zerynth.com>
7/01/2021 11:21

Aan: Ralph Moeskops

Hi Ralph,
In general, Zerynth OS can run non supported python libraries given that it can run on low resource system as MCUs, also depends on other factors like the dependencies...

For the pyZmq library, currently it is not optimized to run on Zerynth OS and not integrated with the Zerynth OS, so it cannot run.

Karim Hamdy
Software Developer
Zerynth | Enabling IoT

--
Prefer fewer emails from me? Click here

Figure 1: Mail from Zerynth

In table 9 pictured below, a brief overview of the pros and cons of the ZeroMQ protocol.

Tabel 9: Pros and Cons ZMQ protocol

Pros	Cons
Simple broker you can make yourself	Won't run in python on PSoC 6
Easily implementable in python	
Easy to understand	
Pub/sub message pattern	
More lightweight than MQTT	
Better documentation than MQTT	

2.3.2 MQTT

MQTT is a standard messaging protocol for the Internet of Things (IoT) and is designed as an extremely lightweight publish/subscribe messaging pattern. The main advantage MQTT has over ZeroMQ is that it is implemented in the Zerynth IDE and thus can run in Python on the PSoC 6. However, this comes at the cost of having to use a 3rd party broker to make this protocol work. This means that it is impossible to modify the broker to the needs of the project. Just like ZeroMQ MQTT supports the pub/sub messaging pattern that is desired for communication between the different nodes in the network. Another advantage of MQTT over ZeroMQ is the fact that the topic is not in the same string as the message, this reduces some parsing of the message on the server side.

In table 10 pictured below, a brief overview of the pros and cons of the MQTT protocol.

Tabel 10: Pros and Cons MQTT protocol

Pros	Cons
Will run in python on PSoC 6	3rd party broker is required
Easily implementable in python	Less lightweight than ZeroMQ
Easy to understand	
Pub/sub message pattern	

3 Results

Met opmerkingen [Bart Stuk6]: Minimaal 250 woorden en aangeraden 1000 woorden (meer mag).
Te schrijven tegen fase 2 en 3 (Bachelorproef).

3.1 Wired VS Wireless

We eventually have chosen to use wireless over wired communication because the pros outweigh the cons. More specifically we will be using Wi-Fi. This is because many microcontrollers don't have the option of using Bluetooth therefore, we opt to use Wi-Fi instead.

3.2 Microcontroller

There were several requirements that the microcontroller had to meet in this case:

- Wi-Fi on board
- Python
- Easy to code/test

If you look at the various micro-controllers, it can be seen that a number of them immediately fall off. The use of an ESP was quickly ruled out. It would not be powerful enough for this application. An incomplete version of Python has also confirmed this choice.

On the PYNQ and ZYNQ boards it is not possible to make a wireless connection with Wi-Fi. Both of these are not suitable for this application. Running a test would also take a long time compared to the other options. So only 2 options remained open, The PSoC 6 and the Raspberry Pi.

Both meet the requirements that were set. They both contain Wi-Fi and can be programmed in Python. A purchase of a PSoC 6 has been made for testing. The tests themselves are discussed under the message queue protocol part. Tests have also been conducted with a Raspberry Pi. These results are also discussed there.

3.3 Message queue protocol

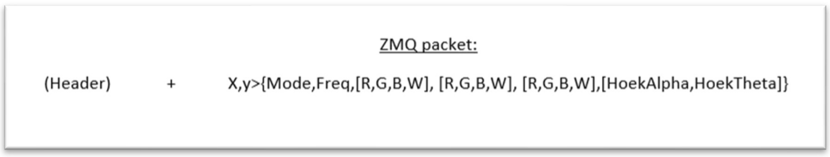
After communication with the different teams working on this project, it was decided to use Message queueing with the pub/sub pattern. Pub/sub functionality is desired because there are multiple programs acting as server and client at the same time. With a pub/sub architecture multiple services can publish at the same time over the same broker without interfering with each other's signals.

3.4 ZeroMQ

To establish a working ZeroMQ communication, a publisher, broker and subscriber was written in Python script. The result is a working publisher that can publish a ZeroMQ-packet to the broker, which will forward the message to all his subscribers. The subscriber will accept or drop the packet depending on the topic.

3.4.1 ZeroMQ Packet

To ensure a smooth communication between team 5 and our team, a short meeting was held to establish certain rules around the ZMQ packet. An example code has been prepared of what the expected JSON file that our team receives might look like. Out of this code, the ZMQ packet representation was established. The intention is to forward our data such as the agreed representation of the ZMQ packet. The chosen representation of the ZMQ packet can be seen in figure 2. The x, y variables are the topic of the packet. Based on this topic, the recipient will know whether it is intended for him or not.


ZMQ packet:

(Header) + X,y>{Mode,Freq,[R,G,B,W], [R,G,B,W], [R,G,B,W],[HoekAlpha,HoekTheta]}

Figure 2: Representation of the ZMQ-packet

3.4.2 Broker

A broker was developed for ZeroMQ to test if the protocol is usable for this project. A quick demo of how the broker runs as well as the code for the broker can be found as an attachment: “Demo ZMQ broker” & attachment “ZMQ broker” to this application note. For the code to work the ZeroMQ library needs to be installed and included in the code. For any issues with the installation refer to attachment “ZMQ documentation” The working principle of the broker is as following, a service (data for “stengels” and AI data) sends a packet to the broker on the brokers listen port using the PUSH socket architecture. The broker takes this message and forwards it to its pub port, all clients (in this case the different “stengels” and the PSoC 6 can subscribe on the topic that concerns them and receive the data without being concerned with the data not destined for them.

3.4.3 Broker on Server

The broker will eventually be running on site where the How do you feel today project will take place. However, since the choice of message queue protocol being MQTT in the end was getting clear by now, the broker will not be used in practice.

3.4.4 Publisher

After the ZMQ packet has been set up, the packet must be sent. A publish code has been written to achieve this. The publish code contains the created ZMQ packet. When this code is run, it makes a connection with the broker. It will then send a ZMQ packet every one second to the broker address. The flowchart in figure 3 explains how the code works. The code itself can be found in attachment “Publisher ZMQ Python” .

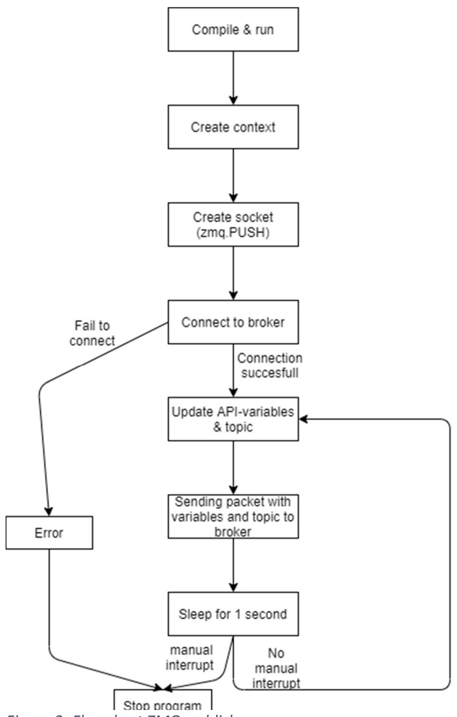


Figure 3: Flowchart ZMQ-publisher

3.4.5 Subscriber

The subscriber code is the code written to receive the ZMQ packet. When the code is run it will connect to the broker. It will then subscribe to the defined topic. From then on, the subscriber will accept all messages from the broker that have the same topic. It will display these messages on the terminal. This makes it possible to check the messages and to check if the packet is received as the agreed representation. When the broker sends a message with a different topic than the defined topic, the subscriber will drop the message. The flowchart in figure 4 explains how the code works. The code itself can be found in attachment "Subscriber ZMQ Python".

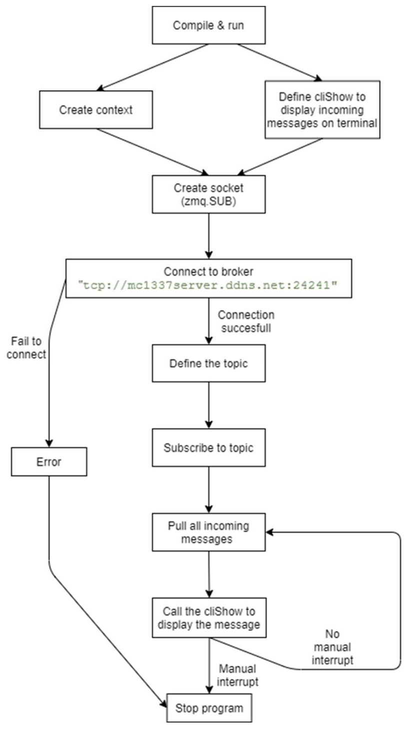

Compile & run
Create context
Define cliShow to display incoming messages on terminal
Create socket (zmq.SUB)
Connect to broker "tcp://mc1337server.ddns.net:24241"
Fail to connect
Connection succesfull
Define the topic
Subscribe to topic
Error
Pull all incoming messages
Call the cliShow to display the message
No manual interrupt
Manual interrupt
Stop program

Figure 4: Flowchart ZMQ-subscriber

3.4.6 RaspBerry Pi

Because ZMQ cannot be installed on the PSoC 6, it was decided to test this method on the Raspberry Pi. ZMQ works easily on the RaspBerry Pi as this is pure Python code. As a result, the flowchart mentioned above in Figure 2 and Figure 3 simply had to be copied. The code itself can be found in attachment "Publisher ZMQ Python" and attachment "Subscriber ZMQ Python". This is the same attachment than mentioned above. These codes have therefore been tested in practice. Both the subscriber and the publisher have been tested with the broker. The Raspberry Pi therefore receives and sends via the ZMQ protocol.

3.5 MQTT

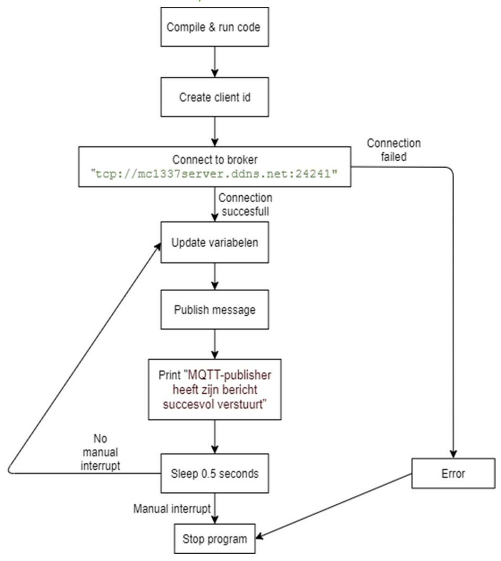
ZeroMQ could not be implemented on the PSoC 6. Therefore, all code written for ZeroMQ had to be rewritten and changed for MQTT. Almost the same protocol as ZMQ, but compatible with the PSoC 6.

3.5.1 Broker

The broker used for MQTT communication is Mosquitto. This broker can be installed on any Linux machine by issuing the command "sudo apt install Mosquitto". After installing the broker, the command "sudo systemctl enable mosquito" can be executed to start the service. This broker will eventually be running on site where the How do you feel today project will take place. For testing purposes, the broker is running online for the moment. This way both the PSoC 6 and ESP8266 modules used by Team 5 can communicate and test their applications while at home. The broker is running at the following IP-address: "mc1337server.ddns.net" using port "24240". There are also two simulations running on this server to simulate data. One of these simulations sends data to the broker with the topic "AIdata/", this topic simulates data that the AI for facial recognition might send out and can be used for testing by team 6. Another program running simulates data coming from team 6's PSoC 6 destined for the ESP8266's of team 5. This data contains "stengel"ID, mode, colors and servo angles as defined in the attachment "communication to team 5.docx". These data packets can be used by team 5 to test and develop their software for the project. An example of how they use the data being send over the broker can be found in the attachment "Demo final result". A quick demo for the MQTT broker can be found in the attachment "demo MQTT broker". In this demo you can see that a client can subscribe to the broker and receive data destined for him from a service that sent this data to the broker.

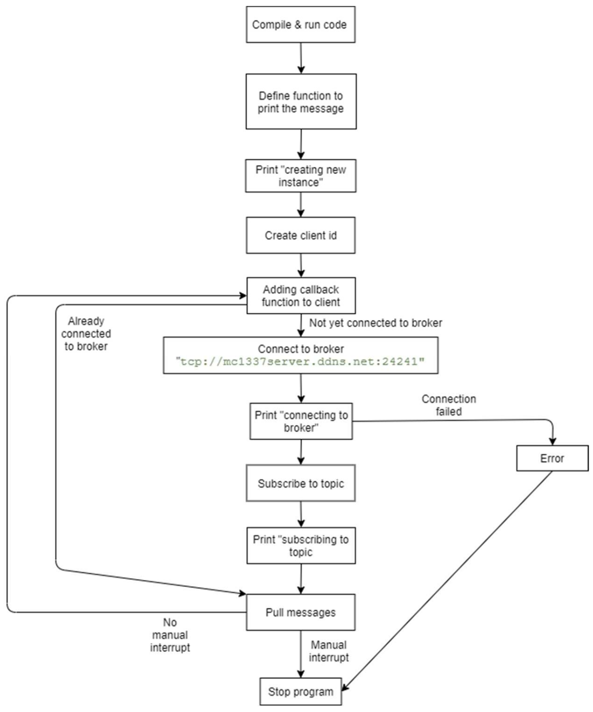
3.5.2 Publisher in Python

The idea behind the publish code for MQTT is the same as the idea behind the publish code for ZMQ. The publisher makes connection with the broker and then publishes the MQTT-message every 0.5 seconds to the broker. In this code, the message still consists of the API variables represented as discussed in "3.4.1 ZeroMQ packet". However, the operation of the MQTT code differs slightly from the ZMQ code. Therefore, modifications had to be made for the code to work. You can find the code in attachment "Publisher MQTT Python" and the flowchart can be seen in figure 5.



Figuur 5: Flowchart MQTT- publisher

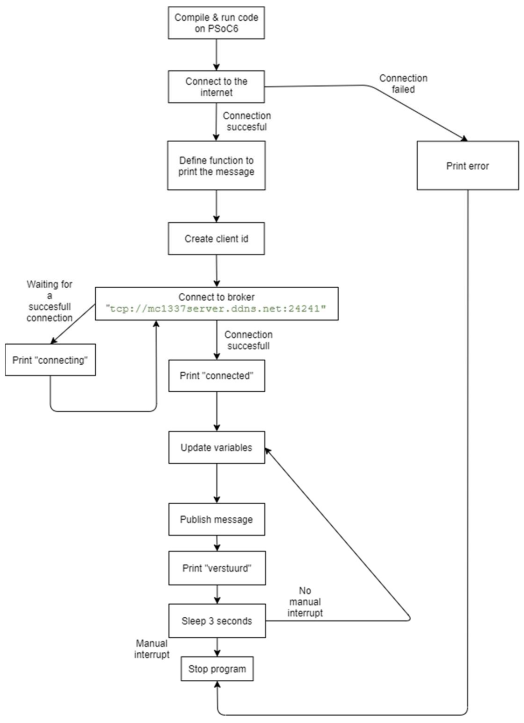
3.5.3 Subscriber in Python


Compile & run code
Define function to print the message
Print "creating new instance"
Create client id
Adding callback function to client
Already connected to broker
Not yet connected to broker
Connect to broker "tcp://mc1337server.ddns.net:24241"
Print "connecting to broker"
Connection failed
Error
Subscribe to topic
Print "subscribing to topic
Pull messages
No manual interrupt
Manual interrupt
Stop program

Figuur 6: Flowchart MQTT- subsriber

The idea behind the MQTT subscriber code matches the idea behind the ZMQ subscriber code. When the subscriber is run, it will make connection with the broker. After that he will pull all messages with the same topic. If the message does not have a similar topic, the subscriber drops it. With this code, the operation is slightly different from the operation of the code in ZMQ. Therefore, several adjustments had to be made in order for the code to work. The flowchart can be seen in figure 6 and the code is visible in the attachment "Subscriber MQTT Python."

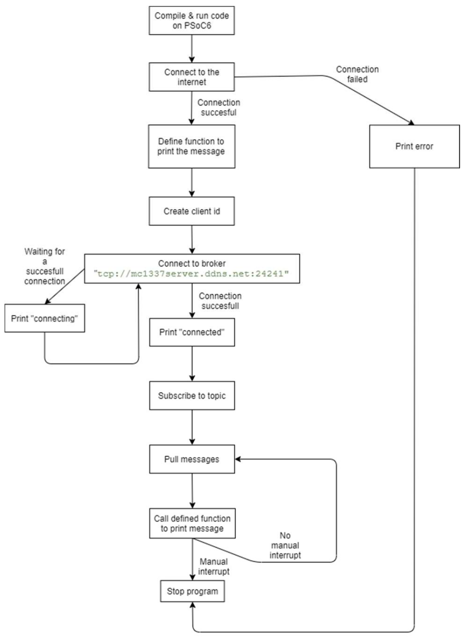
3.5.4 Publisher in Zyrenth



Figuur 8: Flowchart MQTT- publisher PSoC

In general, this code works exactly the same as the Publisher Python code. However, there are some details that are different as can be seen in figure 8. A connection is first made to the internet and additional functions are created for printing later in the code. You can find the code in attachment “Publisher MQTT Zyrenth”.

3.5.5 Subscriber in Zyrenth



Figuur 9: Flowchart MQTT- subsriber PSoC

In general, this code works exactly the same as the subscriber Python code. However, there are some details that are different as can be seen in figure 9. A connection is first made to the internet and additional functions are created for printing later in the code. You can find the code in attachment" Subscriber MQTT Zyrenth" Subscriber MQTT Zyrenth".

4 Discussion

Met opmerkingen [Bart Stuk7]: Minimaal 750 woorden en aangeraden 2000 woorden (meer mag).
Te schrijven tegen fase 2 en 3 (Bachelorproef).

4.1 Wired vs Wireless

Wi-Fi has been chosen to connect the devices to on another. This is because Wi-Fi is wireless, this brings the advantage that no cables must be pulled to every microcontroller. This is significant since every individual "stengel" will use an individual ESP8266 module. If all these controllers need to be connected by cable, a vast number of wired connections would have to be erected. This is not an efficient way to connect this many controllers since this would take up a lot of output pins on the PSoC and this could mean that an IO shield is needed for proper functionality. Rather to make all the different modules communicate a wireless communications system was chosen. Furthermore, the extra connection speed that a wired connection would provide is not needed since the bottleneck of the application is not in this part of the project.

Wi-Fi is also much more flexible and powerful compared to Bluetooth. Bluetooth is very slow in comparison and does not support many devices opposed to a Wi-Fi connection. For example, Wi-Fi can support protocols such as ZeroMQ and MQTT, these would be impossible to implement using Bluetooth. Since message queueing has been researched and decided upon as the most efficient way of controlling the hardware Wi-Fi is the communication system of choice.

4.2 Microcontroller

Finally, we started testing on the PSoC 6. This is because the PSoC 6 has no operating system. This reduces the chance of a crash due to external reasons. Also, it answers the research questions part about efficiently controlling the hardware better since no processing power or electricity has to go to powering the Linux operating system a Raspberry Pi uses.

The security of this system is also more sophisticated than the Raspberry Pi. This is the case since the PSoC 6 does not need an operating system to function, this negates any security flaws that might be present in the Linux operating system. It will work more efficient than the Raspberry Pi that does have an operating system. An advantage of the Raspberry Pi is that parallel tasks can be performed. Although this is only in theory, in fact a Raspberry pi will also run all the code in sequential order. This is due to the Pi only having one processing core. The advantage comes in when it is considered that two or more terminals can be open at the same time executing tasks, this may simplify the coding process in some cases but has so far not been proven to be more efficient

So far, it has not caused any problems to only be able to run one terminal/program at a time. If it should become an issue in the future, a switch to a Raspberry Pi is always a possibility. This is the case since Python code that runs on the PSoC 6 will be able to run on the Raspberry Pi as well with some minor tweaking. This is not the case the other way around, which gives us all the more reason to start on PSoC since migrating the code if something goes south is always a possibility.

At the start of this project, a good and well thought out choice was made to cancel the other candidate micro-controllers. We would not have gotten to this point with those choices. Also, most of the other microcontrollers suggested have insufficient processing power or are not as efficient in controlling the hardware and thus speak directly against the research question.

4.3 Message queue protocol

The pub/sub communication pattern was decided upon because of the way the vast amount of "stengels" need to get their data. With a pub/sub architecture each "stengel" can just subscribe to its own topic and get all and only the data that is destined for the "stengel" itself. Using this way of communicating, the same broker can be used for all data traffic between the facial recognition and the PSoC 6 as well, making this the most efficient communication pattern available.

A request/reply architecture would be a possible solution as well, but since the research question asks for the most efficient way to control the hardware with the data given by the Facial recognition AI, the extra message a request/reply communication pattern requires contributes to a less efficient method of controlling the hardware.

4.3.1 ZeroMQ

ZeroMQ works wonderfully for the project at hand. It has the ability to support all the communication between the Artificial Intelligence for face recognition, PSoC 6, ESP8266's in the "stengels" and the special recognition/smartphones used by team 4 with the use of a broker. At first glance ZeroMQ looks like the perfect solution if it is compared to the research question. It is efficient and can control the hardware given the inputs from the Artificial Intelligence.

The biggest problem with ZeroMQ is the implementation on microcontrollers. It is difficult for team 5 to implement it on their ESP8266 and is impossible to implement on the PSoC 6 in Python. Therefor another protocol has been found to be able to make the code function on PSoC 6.

4.3.2 MQTT

Ultimately MQTT is the decided message queue protocol for this project. Although ZMQ might be more lightweight and better documented, MQTT is the only protocol of the two that was able to function on the PSoC 6 hardware using Python. When discussed with team 5, it has come to our attention that they needed to translate ZeroMQ to MQTT anyway to make it function on their hardware. Both these facts make MQTT the obvious choice in the end. If another microcontroller was in use, for example a Raspberry pi ZeroMQ might have been a better choice for the ease of programming/scripting is this library. However, since team 5 needs to translate anyway we chose for the ease of integration above the ease of programming within the team.

This makes MQTT more efficient in the big picture and thus gives a better answer to the research question with regards to the part about efficiently controlling the hardware. This is the case since we use a protocol that is a bit heavier on its own but saves a lot of processing power by eliminating the need to translate from MQTT to ZeroMQ by team 5 on an external microcontroller.

5 Conclusion

Met opmerkingen [Bart Stuk8]: Minimaal 150 woorden en maximaal 300 woorden.
Te schrijven tegen fase 4 (Bachelorproef).

The use of the Psoc 6 seems very suitable for our project at the moment. If necessary, we can always switch to the Raspberry Pi. The programming language python also works very smoothly for this project because both the IT and the Electronics ICT teams can work with it. The AI of the IT team has become different than expected, so team 4 had to make a Json that must be completed next semester. Wi-Fi will be used for the connectivity between the various devices. Message queue will be used for communication between different hardware devices and software with the use of ZeroMQ and MQTT protocol. The AI has been received from the IT team to continue working with next semester. And the code from team 4 with flowchart to finish the Json next semester has also been received. Next semester the only thing left to do is to make the software to read out the AI and Json and to control the hardware with it.

6 Reference list

Met opmerkingen [Bart Stuk9]: Alle bronnen moeten hier in IEEE-standaard getoond worden. Steeds als je ergens informatie van haalt, moet er een bronvermelding zijn.

[1] XEROX, “A Personal Computer System Hardware Manual,” XEROX, California, 1976.

[2] S. L., “USB deserves more support,” The Boston Globe, 20 mei 1999. [Online]. Available: https://simson.net/clips/1999/99.Globe.05-20.USB_deserves_more_support+.shtml. [Accessed 2021 januari 13].

[3] B. SIG, “How Bluetooth Technology Works,” Bluetooth SIG, 2008. [Online]. Available: https://web.archive.org/web/20080117000828/http://bluetooth.com/Bluetooth/Technology/Works/. [Accessed 13 januari 13].

[4] Zerynth, “The gateway to the Zerynth platform,” Zerynth, 2020. [Online]. Available: https://www.zerynth.com/zsdk/. [Accessed 13 01 2021].

[5] ZeroMQ, “ZeroMQ An open-source universal messaging library,” ZeroMQ, 2020. [Online]. Available: https://zeromq.org/. [Accessed 13 01 2021].

[6] MQTT, “MQTT: The Standard for IoT Messaging,” MQTT, 2020. [Online]. Available: https://mqtt.org/. [Accessed 13 01 2021].

[7] Wikipedia, “Publish–subscribe pattern,” Wikipedia, 07 12 2020. [Online]. Available: https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern. [Accessed 13 01 2021].

[8] Mosquitto, “Eclipse Mosquitto,” Mosquitto, 2020. [Online]. Available: https://mosquitto.org/. [Accessed 13 01 2021].

[9] S. studio, “PYNQ Z2 FPGA Development board | Hardware Overview,” Seed Studio, 2019.

[10] Difilend, “PYNQ-Z1: Python Productivity for Zynq-7000 ARM/FPGA SoC,” Xilinx, [Online]. Available: https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/. [Accessed 13 01 2021].

[11] Xilinx, “ZYNQ-7000,” Xilinx, [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Accessed 13 02 2021].

[12] PYNQ, “PYNQ: Python productivity,” Xilinx, [Online]. Available: http://www.pynq.io/board.html. [Accessed 13 01 2021].

[13] Rs-online, “Development Kit Cora Z7 Zynq-7000 Dual Core for use with ARM Development, FPGA Development,” Digilent, [Online]. Available: https://benl.rs-online.com/web/p/fpga-development-tools/1840484/?cm_mmc=BE-PLA-DS3A-_-google-_-CSS_BE_NL_Raspberry_Pi_%26_Arduino_%26_Development_Tools_Whoop-_-(BE:Whoop!)+FPGA+Development+Tools-_-1840484&matchtype=&aud-827186183886:pla-312891339080&gclid. [Accessed 13 01 2010].

[14] D. P.Greorge, “Getting started with micropython on the ESP32,” micropython, [Online]. Available: https://docs.micropython.org/en/latest/esp32/tutorial/intro.html. [Accessed 13 01 2021].

[15] D. George, “MircoPython,” MircoPython, [Online]. Available: https://micropython.org/. [Accessed 13 01 2021].

[16] “PSoC® 6 WiFi-BT Pioneer Kit (CY8CKIT-062-WiFi-BT),” Cypress, 09 01 2021. [Online]. Available: https://www.cypress.com/documentation/development-kitsboards/psoc-6-wifi-bt-pioneer-kit-cy8ckit-062-wifi-bt. [Accessed 13 01 2021].

[17] L. Vukovic, “Python on PSoC 6,” Zerynth, [Online]. Available: https://www.zerynth.com/blog/cypress-and-zerynth-demo-python-on-psoc-6-microcontrollers-for-internet-of-things-and-blockchain-applications/. [Accessed 13 01 2021].

[18] “Raspberry Pi 4,” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [Accessed 13 01 2021].

7 Attachment

Met opmerkingen [Bart Stuk10]: Lijst van bijlagen, mag in totaal niet meer pagina's bevatten dat de tekst zelf. Er moet steeds actief verwezen worden naar een bijlage vanuit de tekst.

7.1 Publisher ZMQ Python

```
# Verzenden API gegevens ZMQ-packet

import zmq
import time

context = zmq.Context()
socket = context.socket(zmq.PUSH)
socket.connect("tcp://mc1337server.ddns.net:24241")

while True:
# Declareren van voorbeeld waardes die zogezegd van de api komen
# In de toekomst worden in deze variabelen de echte uitgelezen waardes van de api gestopt

#topic = ID (x,y)
x = 0
y = 1

#message
mode = 2
freq = 25.000 #in Hz
R1 = 255
G1 = 65
B1 = 38
W1 = 0
R2 = 44
G2 = 0
B2 = 120
W2 = 33
R3 = 0
G3 = 25
B3 = 255
W3 = 0
hoekAlpha = 20
hoekTheta = 170

# Send to reciever
socket.send_string("%u,%u>{%u,%f,[%u,%u,%u,%u],[%u,%u,%u,%u],[%u,%u,%u,%u],[%u,%u]}"
% (x, y, mode, freq, R1, G1, B1, W1, R2, G2, B2, W2, R3, G3, B3, W3, hoekAlpha,
hoekTheta))
time.sleep(1)
```

7.2 Subscriber ZMQ Python

```python
# ontvanger random zmq packet

import zmq
import time

ms_TO_WAIT_IN_POLL = 100

def cliShow(aText="no parameter was fed in"):
    print("{0:}:: {1:}".format(time.ctime(), aText))


#  Socket to talk to server
context = zmq.Context()
pull = context.socket(zmq.SUB)
print("Collecting updates from server...")
pull.connect("tcp://mc1337server.ddns.net:24242")

topicfilter = ""
pull.subscribe(topicfilter)
while True:
    msg = pull.recv()
    cliShow(msg)

pull.close()

context.term()
```

7.3 Publisher MQTT Python

```
# zender van ons packet

import paho.mqtt.client as mqtt # import the client1
import time

# use external broker
broker_address = "mc1337server.ddns.net"
broker_port = 24240
# create new instance
client = mqtt.Client("Team6 publisher")
# connect to broker
client.connect(broker_address, broker_port)


# topic = ID (x,y)
x = 0
y = 1

# message, dit worden in de toekomst ingevuld door de variabelen van de JSON-file
mode = 2
freq = 25.000  # in Hz
R1 = 255
G1 = 65
B1 = 38
W1 = 0
R2 = 44
G2 = 0
B2 = 120
W2 = 33
R3 = 0
G3 = 25
B3 = 255
W3 = 0
hoekAlpha = 20
hoekTheta = 170

# publish the message
while True:
client.publish("%u,%u" % (x, y),
"{%u,%f,[%u,%u,%u,%u],[%u,%u,%u,%u],[%u,%u,%u,%u],[%u,%u]}" % ( mode, freq, R1, G1, B1,
W1, R2, G2, B2, W2, R3, G3, B3, W3, hoekAlpha, hoekTheta))
print("MQTT-publisher heeft zijn bericht succesvol  verstuurt")
time.sleep(0.5)
```

7.4 Subscriber MQTT Python

```
# testcode voor het ontvangen van het mqtt packet

import paho.mqtt.client as mqtt # import the client1
import time

#######################################
def on_message(client, userdata, message):
    print("message received " ,str(message.payload.decode("utf-8")))
    print("message topic=",message.topic)
    print("message qos=",message.qos)
    print("message retain flag=",message.retain)
########################################

broker_address = "mc1337server.ddns.net"
broker_port    = 24240
print("creating new instance")
# create new instance
client = mqtt.Client("Team6 subscriber")
# attach function to callback
client.on_message = on_message
print("connecting to broker")
client.connect(broker_address, broker_port)
print("subscribing to topic")
client.subscribe("0,1")
client.loop_forever()
```

7.5 Publisher MQTT Zyrenth

```
import streams
from mqtt import mqtt
from wireless import wifi
from murata.lbee5kl1dx import lbee5kl1dx as wifi_driver

wifi_driver.auto_init()
streams.serial()

print("Establishing Link...")
try:

    wifi.link("NetwerkRalph",wifi.WIFI_WPA2,"Ralph1607")
except Exception as e:
    print("ooops, something wrong while linking :(", e)
    while True:
        sleep(1000)

try:
    # set the mqtt id to "zerynth-mqtt"
    client = mqtt.Client("Team6_PSOC",True)
    # and try to connect to "test.mosquitto.org"
    for retry in range(10):
        try:
            client.connect("mc1337server.ddns.net", 60,24240)
            break
        except Exception as e:
            print("connecting...")
    print("connected.")
    # topic = ID (x,y)
    x = 1
    y = 1
    # message, dit worden in de toekomst ingevuld door de variabelen van de JSON-file
    mode = 2
    freq = 25.000  # in Hz
    R1 = 255
    G1 = 65
    B1 = 38
    W1 = 0
    R2 = 44
    G2 = 0
    B2 = 120
    W2 = 33
    R3 = 0
    G3 = 25
    B3 = 255
    W3 = 0
    hoekAlpha = 20
    hoekTheta = 170

    while True:
        sleep(3000)

        client.publish("%u,%u" % (x, y),
"{%u,%f,[%u,%u,%u,%u],[%u,%u,%u,%u],[%u,%u,%u,%u],[%u,%u]}" % ( mode, freq, R1, G1, B1,
W1, R2, G2, B2, W2, R3, G3, B3, W3, hoekAlpha, hoekTheta))
        print("Verstuurd.")

except Exception as e:
    print(e)
```

7.6 Subscriber MQTT Zyrenth

```
import streams
from mqtt import mqtt
from wireless import wifi
from murata.lbee5kl1dx import lbee5kl1dx as wifi_driver

wifi_driver.auto_init()
streams.serial()

print("Establishing Link...")
try:

    wifi.link("NetwerkRalph",wifi.WIFI_WPA2,"Ralph1607")
except Exception as e:
    print("ooops, something wrong while linking :(", e)
    while True:
        sleep(1000)


def is_sample(data):
    if ('message' in data):
        return (data['message'].qos == 1 and data['message'].topic == "desktop/samples")
    return False

def print_sample(client,data):
    message = data['message']
    print("sample received: ", message.payload)

def print_other(client,data):
    message = data['message']
    print("topic: ", message.topic)
    print("payload received: ", message.payload)

def send_sample(obj):
    print("publishing: ", obj)
    client.publish("temp/random", str(obj))

def publish_to_self():
    client.publish("desktop/samples","hello! "+str(random(0,10)))

try:
    # set the mqtt id to "zerynth-mqtt"
    client = mqtt.Client("Team6_Subscriber_PSOC",True)
    # and try to connect to "test.mosquitto.org"
    for retry in range(10):
        try:
            client.connect("mc1337server.ddns.net", 60,24240)
            break
        except Exception as e:
            print("connecting...")
    print("connected.")
    # subscribe to channels
    client.subscribe([["AIdata/",1]])

    client.on(mqtt.PUBLISH,print_sample,is_sample)
    client.loop(print_other)

except Exception as e:
    print(e)
```

7.7 ZMQ broker

```
import time
import zmq

ms_TO_WAIT_IN_POLL = 100

def cliShow(aText="no parameter was fed in"):
    print("{0:}:: {1:}".format(time.ctime(), aText))

context = zmq.Context()

push = context.socket(zmq.PUB)  # Socket facing out - THE zmq.PUSH ARCHETYPE is a right-
enough one
push.bind("tcp://*:24242")  # push acquires all ports 5556 to be PUSH-served
push.setsockopt(zmq.LINGER, 0)  # .SET always explicitly, even if "defaults" promise

pull = context.socket(zmq.PULL)  # Socket facing in  - THE zmq.PULL ARCHETYPE is a right-
enough one
pull.bind("tcp://*:24241")  # pull acquires all ports 5555 to be PULL-served
pull.setsockopt(zmq.LINGER, 0)  # .SET always explicitly, even if "defaults" promise

try:
    while True:
        if (0 == pull.poll(ms_TO_WAIT_IN_POLL, zmq.POLLIN)):
            # cliShow("No message arrived yet")  # CLI - GUI STUB print()
            dud = 1
        else:
            msg = pull.recv(zmq.NOBLOCK)  # /NEVER/ USE A BLOCKING .recv()
            cliShow("recieved message...")  # CLI - GUI STUB print()
            push.send(msg)  # .send()
            cliShow("And sent it to all subscribers")  # CLI - GUI STUB print()

except KeyboardInterrupt:
    pass

except:
    pass
finally:
    pull.close()
    push.close()

    context.term()
```

7.8 Demo ZMQ broker

The video can be viewed by the following link:

https://github.com/11902804/MADDIGITAL-TEAM6/blob/main/demo_zmq_broker.mkv

7.9 Demo MQTT broker

The video can be viewed by the following link:

https://github.com/11902804/MADDIGITAL-TEAM6/blob/main/demo%20mqtt%20broker.mkv

7.10 Communication to team 5

The file can be viewed by the following link:

https://github.com/11902804/MADDIGITAL-TEAM6/blob/main/communication%20to%20team%205.docx

7.11 Demo final result

The video can be viewed by the following link:

https://youtu.be/zQFnieEKYRw

7.12 ZMQ documentation

The file can be viewed by the following link:

https://github.com/11902804/MADDIGITAL-TEAM6/blob/main/ZMQ-Documentatie.docx