ULB UNIVERSITÉ LIBRE DE BRUXELLES

BRUFACE
BRUSSELS FACULTY OF ENGINEERING

VUB VRIJE UNIVERSITEIT BRUSSEL

MA1 Project

IOT ENABLED REHABILITATION AND ASSISTIVE DEVICES USING SMART ACTUATOR MODULES

Julien van Delft

2020-2021

Prof. Dirk Lefeber, Prof. Bram Vanderborght, Dr. ir. Victor Grosu

Electromechanical Engineering

Contents

Abstract

Nowadays, the development of exoskeletons for medical or personal use is exploding. Indeed, they are able to help people recovering from difficult health situation or helping motion. But as exoskeleton users are often vulnerable, continuous communication with a health centre is useful to ensure patient permanent safety. This study focuses on an exoskeleton built from SMARCOS actuators that needs to be connected to the internet through cellular connectivity. It must use a reliable, low-latency and large coverage communication mode. To that end, the different mobile technologies from 1G to 5G including LPWAN will be compared. Similarly, the exchange of data needs to be built on an efficient and lightweight communication protocol to ensure good reception on low-power cellular modules. After comparisons between the cellular technologies especially LTE Cat 1, LTE Cat 4, NB-IoT and LTE-M and between MQTT, CoAP, AMQP and HTTP as communication protocols, the best-suited for our application are found to be respectively LTE Cat 4 and MQTT. A corresponding compatible cellular module will be bought on the market for implementation. Then, a Simulink subsystem for communication with the exoskeleton will be created. Finally, an MQTT server is going to be implemented on a Raspberry Pi. The results will lead to a fully operational IoT enabled system to connect an existing exoskeleton to the internet. However, more testing would be needed with the exoskeleton as it will not be tested due to the sanitary context.

Keywords: Exoskeleton, communication protocol, cellular connectivity, 4G, 5G, LTE Cat 4, NB-IoT, LTE-M, MQTT, CoAP, AMQP, HTTP, server, real-time monitoring, rehabilitation.

1 Motivation

One important problem that is growing and will continue to grow during the 21st century is the physical disability. First, as the life expectancy is increasing, the population gets older, increasing the assistance needs to move and achieve simple tasks. By looking at the figure 1, one can see that 1 person aged 65 years or more out of 2 has experienced difficulties carrying out personal care or household activities. This is problematic as in [1], it is stated that 1 person out of 5 in the European Union is more than 65 years old.

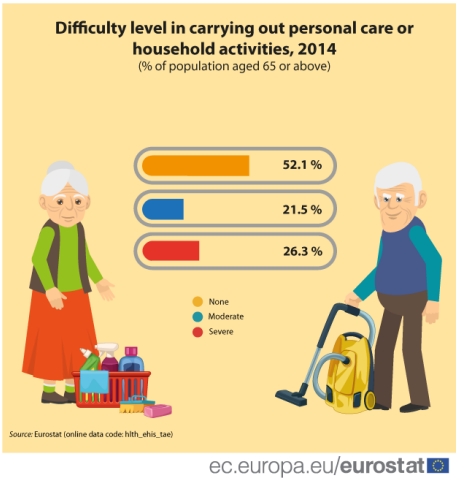

Figure 1: Elderly disability statistics [2].

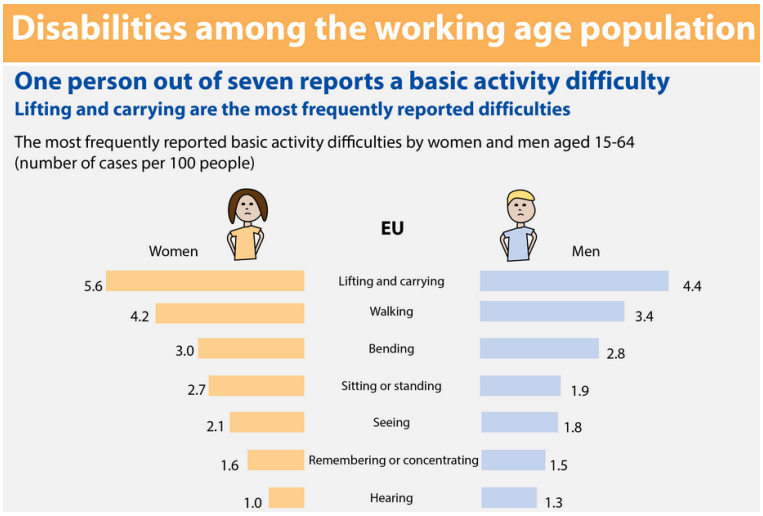

Figure 2: Working population disability statistics [3].

Secondly, a lot of people suffer from disabilities during their lifetime. This can be either caused by a medical condition (a stroke for example) or by an accident leading to cerebral or muscular dysfunction. In the figure 2, it is stated that at least one person out of seven in the working population is affected by this problem. The most common ones are linked to the motion of the muscle, for example, lifting a load or simply walking. To recover from the disabilities, a long rehabilitation must be undertaken which can be difficult for the patient.

To overcome those two problems, one current solution proposed is the development of exoskeletons. The latter can help elder people by making the motion and task easier. They are also widely used as a rehabilitation tool to assist the patient in their path to recovery. Indeed, with an exoskeleton, it is possible to adapt the load to focus and train the patient in the most efficient way.

2 Project description

As stated before, in the future, one can expect that lots of disabled people will be using exoskeletons to ensure safe and easy motion. However, it is useful that the exoskeleton is permanently connected to a central system or a supervisor to send help or important information whenever needed. The most obvious data communication one can think of is cellular communication to have 24/7 simple monitoring and global coverage over the country.

This project is developed around an exoskeleton, built from integrated smart actuators modules, which needs to be connected to the internet for remote monitoring and control. The mechanical system consists of two smart actuators SMARCOS as shown in figure 3 connected together. The connection will be held through the cellular network to allow connection everywhere to ensure the security of the patient. Research will be conducted on 4G and 5G technologies to find a way to achieve the monitoring in real-time. The main task being to transform the architecture to allow mobile communication from and to the system.

The project relates to 3 other projects linked to the same exoskeleton. The global objective is the creation of the whole system in figure 3, which includes a rehabilitation game with an interface for control by the doctor and the mechanical system to connect the exoskeleton to the patient's arm. This game would help the patients to recover while having fun and give them back some muscle functionalities. This report is focused on the IoT enabling through cellular connectivity (the lower middle part on figure 3).

The exoskeleton communicates with a computer through Matlab and Simulink. On this computer, the rehabilitation game is running (front end) and the exoskeleton acts as a joystick in the game to motivate the patient during the rehabilitation. The doctor can monitor the patient's game to analyse the data and send special parameters to focus on some muscles (back end). The exoskeleton is connected to a cellular communication module to send and receive special data that need urgent and real-time communication. This communication is especially important for further applications to allow real-time monitoring of the patient whenever and wherever the exoskeleton is used.

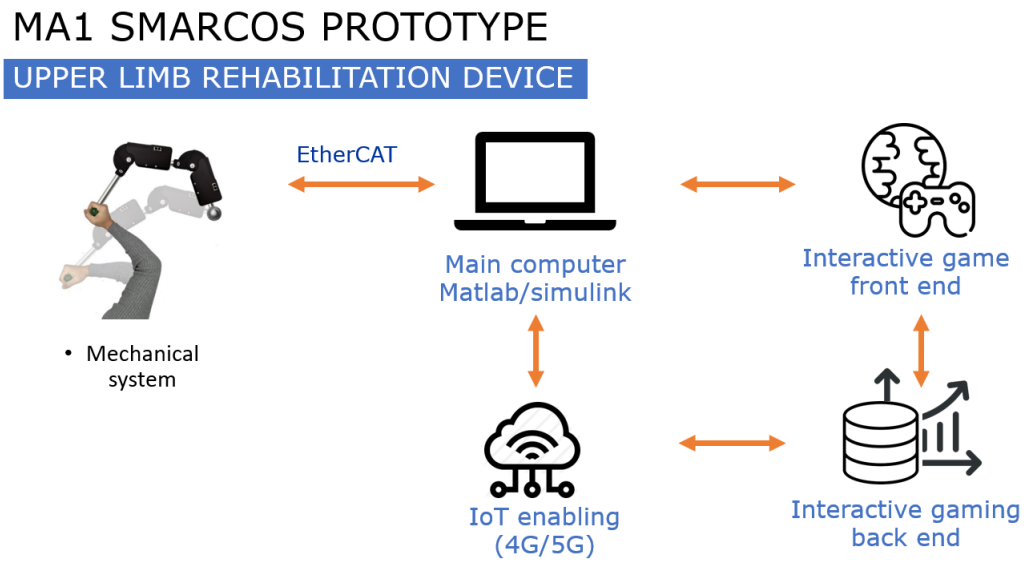

Figure 3: Architecture of the project including the exoskeleton, the game and the cellular connectivity on which this report is focused.

This report will be oriented to answer the following research questions:

1. What are the main differences between 4G and 5G connectivity?
2. Which technology is best suited for medical exoskeleton in Belgium?
3. What connectivity modules are best suited for remote control and monitoring?
4. What data transfer protocol is best suited for IoT cellular application?

To answer these questions, the different cellular technologies will be compared from 1G to 5G including the LPWAN to get a wide view of the possibilities. Similarly, the communication protocols are going to be discussed to find the lighter, more reliable and secure protocol that will be implemented on the cellular module. Naturally, those will be compared with particular attention to the requirements stated in the next section. A compatible board is going to be bought on the market after a comparison of the available solutions.
Concerning the implementation, the board will be coded, a Simulink subsystem will be created to communicate with the exoskeleton and a server will be implemented on a dedicated Raspberry Pi to have full control of the server and avoid the dependency on an external server.
Finally, the results will be analysed with, once again, particular attention to the latency.

3 Requirements list

The exoskeleton has 6 sensors sending 16 bits every ms. The total data rate needed for transferring all the sensor bits is at least 6 sensors $*$ 16 bits $*$ 1000 ms/s = 96 $kbps$. This value will be used to give an idea of the potential needed data rate of the exoskeleton. Even though in the end those data will not be sent using cellular communication, this value will be used for research purposes.

Due to an increase of IoT technologies that are exploding nowadays, we need to use a communication technology allowing a massive number of simultaneous connections. The exoskeleton is a wearable device. Therefore, the connectivity must be suited as wearable and easily switched between different antennas with the smallest delay possible.
The exoskeleton is being developed in Belgium and the patient needs to be able to use the exoskeleton almost everywhere in the country. As the device is mobile, it needs to be power-efficient as much as possible to avoid a problematic breakdown due to an empty battery.
The delay needs to be as low as possible because it needs to follow the motion of the user as quickly as possible to avoid important lags and ensure the safety of the user.

Requirements
Massive simultaneous connections
Suited as wearable
Available now at least in Belgium
Large coverage to ensure connectivity
Power efficiency
Small delay
Data rates minimum of 96 $kbps$
Small cost

Figure 4: Requirements list built for the SMARCOS exoskeleton.

Chapter 1

State of the art

1 Introduction

To have a clear understanding of the problem and the available solutions, the first chapter is devoted to theoretical considerations. First, the related works are introduced, in two parts, rehabilitation games with exoskeletons and cellular robotics systems. After that, the different mobile technologies will be compared from a historical point of view from 1G to 5G. The cellular connectivity will be compared with other communication means to explain the choice of cellular over LPWAN. Finally, to find the best-suited cellular communication, 4G, 5G, NB-IoT and LTE-M are going to be studied based on latency, availability and data rate. Then, the more coherent one with our requirements will be selected.
The communication needs to be built on an efficient, lightweight protocol to ensure reliability and fast sending of messages. To that end, MQTT, CoAP, AMQP and HTTP are going to be compared on bandwidth, latency, reliability and security, and the best-suited will be chosen.

2 Similar project

2.1 Rehabilitation games

There are not a lot of projects on the same subject. It is common to find some rehabilitation exoskeletons with rehabilitation games. However, most of the time, the exoskeleton stays in the rehabilitation or medical centre leading to sufficient WiFi connectivity. The study in [4] is very close to the global project developed by our team. They created an elbow exoskeleton for rehabilitation with a Virtual Reality developed game. Both the sensors in the exoskeleton and the game communicate through WiFi with a monitoring cloud. This cloud gives the exercises, stores the data, and allows the doctor to remotely monitor the patient. This is summarised in the figure 1.1.
It is very similar to our project, the only difference being with the game on a computer and not with a VR environment and the WiFi-enabled development board switched to a cellular one.

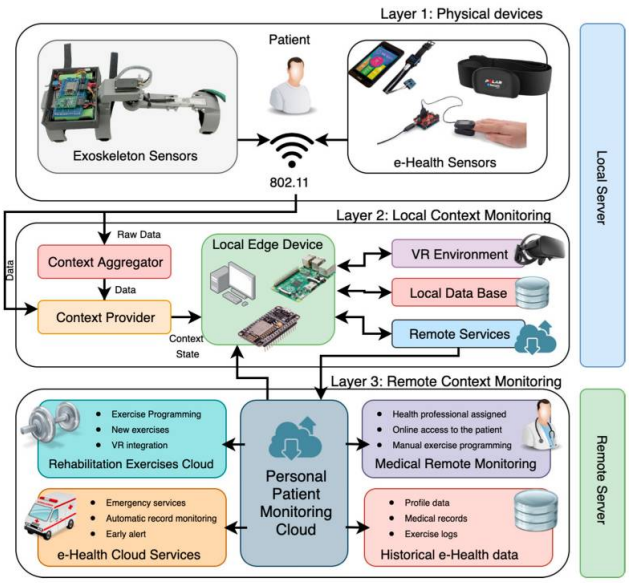

Figure 1.1: Global architecture of the *Connected elbow exoskeleton system for rehabilitation training based on virtual reality and context-aware* research [4].

2.2 Connected robotics system

Although in the research field, few examples of cellular enable IoT exoskeleton are found, those are common in the industry. Indeed, opposed to the rehabilitation centre, the workers using exoskeleton need to stay connected in huge warehouses and on work sites. In [5], the implementation of a smart factory is explained. The smart factory would still need human workers but upgraded by passive or active exoskeletons. Those exoskeletons are going to be IoT connected using WiFi if possible or LPWAN/cellular communication. The advantages of IoT connected exoskeleton are numerous. One can think of the optimisation of the workers' path. With a global perspective of all the workers' positions, it is easier to organise the warehouse and monitor the productivity. From a security point of view, it is necessary to ensure that help can be quickly given to any worker in difficulty after the exoskeleton has put itself in safe mode to protect the user. Also, the robots and machines could more easily communicate with the workers through the exoskeletons.

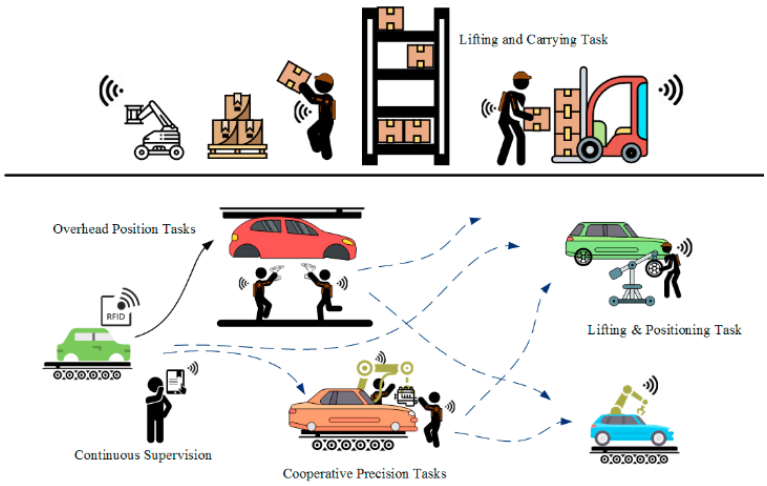

Figure 1.2: Overview of a smart factory deploying exoskeleton equipped workers and communication between machines and exoskeletons [5].

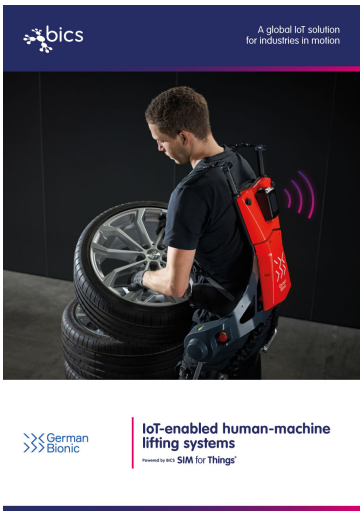

Figure 1.3: Example of cellular-connected IoT exoskeleton [6].

In the figure 1.3, an example of a commercialised exoskeleton IoT-enabled using cellular communication is shown. This has been recently released and it is foreseen that in the future, most human workers will be using this kind of IoT exoskeleton for the reasons stated previously.

Important information can also be taken from the cellular-connected robotics system such as the connected healthcare system presented in [7]. The system is based on an LTE network with wireless body area networks. The study describes the usage of cloud computing and of UDP-based protocol to measure physiological information and deliver real-time video contents. Indeed, even though this project does not imply an exoskeleton, the challenges and opportunities are very similar. The latency, reliability and security are crucial as they can impact the quality of patient healthcare.

3 Mobile technology

From [8], [9], [10], [11], [12] and [13], we can conclude that mobile communication technology is the result of a slow evolution process that started in the '50s with what is called the 0G, or "pre-cellular" and has been accelerating in the last 10 years.

The first real technology of mobile communication is the 1G that has been introduced in the '80s. It was completely restricted to voice service. At that time, the technology was analogue, but it was not generalised due to

the high cost of phones, their poor batteries and voice quality. The maximum speed was around 2 $Kbps$.

The 2G was implemented at the beginning of the '90s and has the advantage of digital communication. This also allowed data services such as SMS and MMS. One important key feature is the text and call encryption increasing the security of the network. During this generation, the use of phones has been widely spread and the number of users per frequency band increased due to significantly more efficient use of the radio frequency spectrum. The maximum data rate was about 15 $Kbps$.

The "second and a half generation" also called 2.5G has seen the *GPRS* (General Packet Radio Service) and the *EDGE* (Enhanced Data Rates for GSM Evolution, called 2.75G) emerging. The GPRS introduced a new packet-switching technology that was more efficient. The EDGE used the 2.5G technology and increased the data rates. The maximum data rates for 2.5G and 2.75G are respectively 75 $Kbps$ and 400 $Kbps$.

The 3G was released few years after the 2.75G in 2002 leading to an important increase in the data rate. From there, cellular technology could be used for demanding application such as video calling, mobile internet and mobile TV. The data rate was around 2 $Mbps$ up to 10 $Mbps$ for the latest 3G technology in 2006.

The 4G (also called LTE) was developed around 2010. Once again, the data rate increased and now the technology is completely IP (internet protocol) oriented. The classical circuit-switched telephone service is replaced by a full IP based communication. The data rate becomes 100 $Mbps$ for a moving device and up to 1 $Gbps$ for low-mobility communication ([14]).

The last technology that is currently being implemented around the globe is the 5G technology. The available frequency range is way bigger than before allowing more simultaneous connection. The data rate becomes very high from 1 to 10 $Gpbs$.

Each technology also decreased the latency (meaning the round-trip time taken by the data sent) on the network up to latency as low as 1 ms for the 5G. Concerning the real-time applications, the latency needs to be as low as possible with a maximal value from few dozens of ms to few hundreds depending on the applications.

Type of connectivity	Year	Data rate	Latency	Security	Specificity
1G	80's	2 $Kbps$	Very high	Very low	Voice only
2G	90's	15 $Kbps$	500 ms	Medium	Digital communication
2.5G	1999	75 $Kbps$	200 ms	Medium	New packed-switching technology
3G	2002	2 $Mbps$	100 ms	Medium	Increase of data rates (video, TV)
4G	2010	100 $Mbps$	50 ms	High	Completely IP oriented
5G	2019	1-10 $Gbps$	1 ms	High	Huge frequency range

Table 1.1: Comparison of the mobile technologies based on data rate, latency, security and characteristic showing the evolution and improvements at each generation.

The table 1.1 summarises the information developed in the beginning of this section. One may notice that the data rates, latency and security are considered for the first release of the technology. For example, the 4G is fixed to its first 100 $Mbps$ in 2010 and not the 1 $Gbps$ that can be achieved nowadays through (new) 4G or LTE-Advanced.

4 Comparison of communication means

In this section, different communication means are compared. The most obvious one can think of is of course WiFi. Although the data rate can be up to 2 Gbps for WiFi 6 ([15]) and latency as low as a few ms ([16]), WiFi has the drawback of the coverage. As seen in figure 1.4, the range of WiFi can be up to 100m only.

Moreover, the router that brings WiFi takes all connection of the network and has only limited available bandwidth. This means that the other users on the network will decrease the data rate and increase the latency for a given user. Cellular connectivity on the other hand presents the advantage of the coverage (up to km around the antenna). It is also reliable and easy to use as it is the most used large coverage network. However, cellular communication is more energy consuming than WiFi or LPWAN. Finally, in the LPWAN, one can think of Sigfox, LoRaWan and NB-IoT (which will be discussed later on). Those are low cost and large coverage up to dozens of km in rural areas ([17]). The different technologies are compared in the table below:

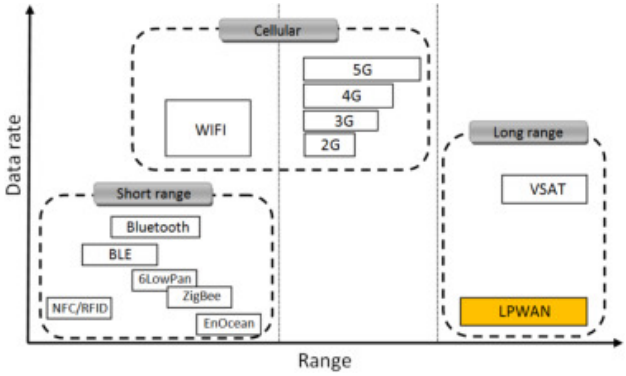

Figure 1.4: Comparison of the LPWAN with cellular and WiFi based on the data rate and the range [17].

Communication mean	Data rate	Latency	Coverage	Power consumption
WiFi	2 Gbps	10 ms	100 m	Medium
4G	100 Mbps	50 ms	few km	High
Sigfox	100 bps	Very high	up to 40 km	Very low
LoRaWAN	35 kbps	Very high	up to 20 km	Very low
NB-IoT	50 kbps	few seconds	up to 10 km	Low

Table 1.2: Comparison of communication means on data rate, latency, coverage and power consumption based on [17], [18] and what was stated before.

Due to the requirements on coverage and latency, the LPWAN network and WiFi seems not suited for the medical exoskeleton.

5 4G/LTE

The LTE evolution has been divided into two categories [19]. The high-speed cellular technologies that contain LTE Cat 3/4 to 12 with data rates around 100 *Mbps* up to 1 *Gbps* and very low latency. And on the other side, the LPWA (Low Power Wide Area) technologies allow smaller data rates, higher latency but better power efficiency. 5G takes both extremes of the graph, meaning that it is evenly suited for small IoT low-power devices than for very powerful devices such as smart live-streaming 4K city cameras. The current IoT applications focus often on the LPWA networks (LPWAN) as it is mostly composed of low-power sensors or microcontrollers. The LTE name regroups all the LPWAN and the 4G categories 3 and 4 and with a small misuse of language also the categories 6, 9 and 12. Although it would be more correct to divide those in regular LTE and LTE-Advanced, those two are often regrouped under the LTE term. This simplification will be held in this report as well for simplicity.

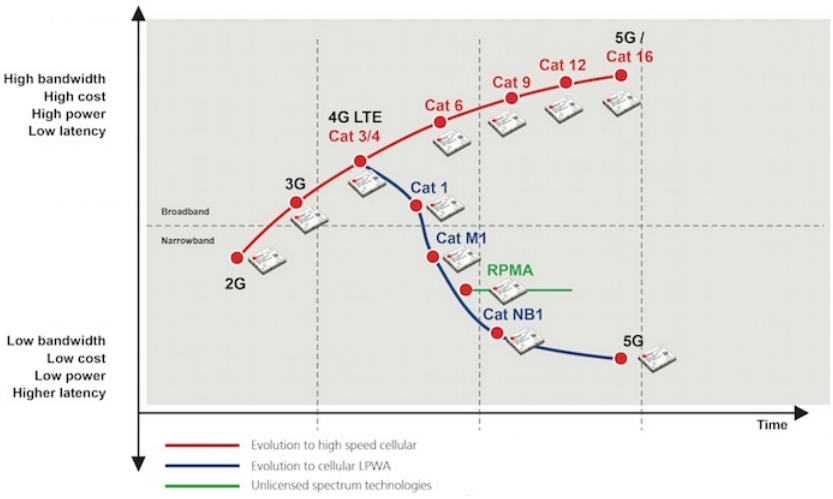

Figure 1.5: Evolution of the cellular technologies with time considering bandwidth, cost, power and latency [19].

5.1 Working Principle

LTE operates on a frequency spectrum that spreads from 700 MHz to 2.6 GHz. LTE has chosen to change from sending the data through classical GSM to an IP system. Therefore, it is way quicker, the resources are better allocated, the data rate increased a lot and so on. Another important point is that LTE allowed to divide very efficiently the frequency spectrum, which makes a different frequency for each carrier increasing the number of simultaneous connection per node. Each frequency got allocated a special number, the band (Bx where x is the number of the band). In Belgium B3, B7 and B20 are in use for LTE which corresponds to 1800, 2600 and 800 MHz [20].

Some bands share the same frequency but differ either due to the difference of frequency between the uplink and downlink or due to another duplex mode. The duplex mode is the way the download and upload are treated. There are two different modes that exist. The first is the FDD (Frequency-division duplexing) which consists of separating the two channels (upload and download) in two different frequencies. This allows to have both upload and download simultaneously without impact from one to the other. The second duplexing is the TDD (Time-division duplex) where the same frequency is shared between both uplink and downlink. This has the advantage of a more dynamic allocation of the resources as the ratio given to each can be changed depending on the needs. For example, if all devices are downloading, the antenna can allocate almost 100% to that intent leading to faster downloads.

5.2 Basic architecture of LTE communication

In [21], the whole architecture is explained. The full architecture is named Evolved Packet System (EPS) which is divided into the Evolved Packet Core (E-PC) and the physical network labelled UMTS Terrestrial Radio Access Network (E-UTRAN) as shown in figure 1.6. The E-PC is itself split into the Mobility Management Entity (MME), Service Gateway (S-GW) and Packet Data Network Gateway (P-GW). On the other hand, the E-UTRAN is only composed of the eNodeBs (evolved base stations). The user equipment (UEs) communicates through eNodeB. The different eNodeBs are connected together and to the MME and S-GW. The MME is responsible for the security and the dynamic allocation of the resource (choosing the less busy S-GW and P-GW accessible). The S-GW is responsible for passing the data to the P-GW which removes the dependence of the localisation of the user. The P-GW role consists of performing the routing to the internet using the IP address.

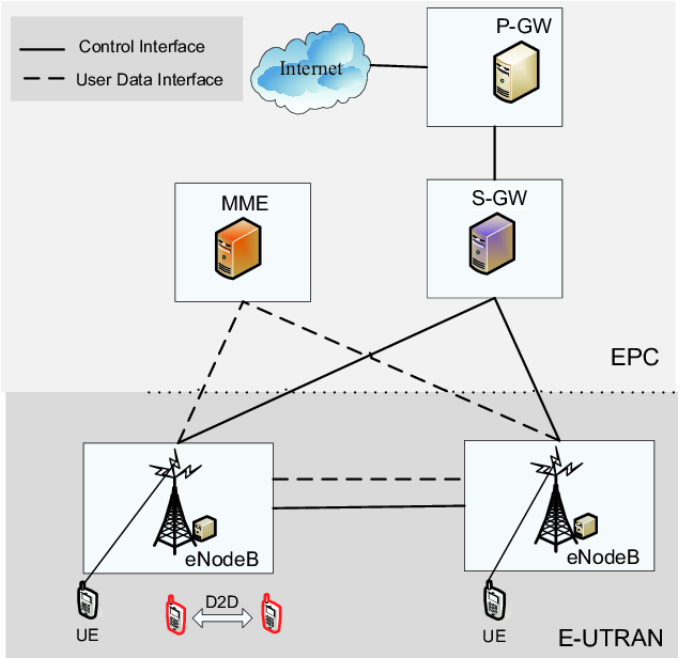

Figure 1.6: Basic LTE Architecture comprising the eNodeBs, MME, S-GW and P-GW to connect the UE to the Internet [21].

All categories of LTE communications follow this simple architecture with some complexities or simplifications for better data rates or LPWAN.

5.3 NB-IoT

NB-IoT (or LTE CAT NB1) is a rather new LPWAN (released in June 2016) and has the particularity to use a small frequency of 180kHz ([22] and [23]). This low frequency gives better coverage even in some isolated places (up to 35km). But the drawback is a higher latency (1 to 10 seconds) and a limited data rate (between 25 and 50 kbps). The protocol is also different, it does not use the IP protocol. The big advantage is that it can be

used in every LTE node with a simple digital update of the antenna. Therefore, it can add up to 200 000 devices per node on top of all the other LTE devices already connected without any interference as it uses a special frequency less used nowadays. This connectivity allows operating with an ultra-low consumption letting the system work for more than 10 years without any battery problem. This impressive power efficiency is also due to a clever sleeping process allowing the sensor to sleep and wake up only when it receives or sends something. As this technology is part of the LTE technologies, it is 5G ready as the 5G nodes will also deploy the NB-IoT connectivity. However, there is a mobility problem with the NB-IoT as it can take up to several minutes to connect to another antenna which is then not suited for wearable or mobile devices.

This connectivity is particularly crucial for future IoT applications as the number of connected devices is increasing exponentially (25 billion in 2020 [22]). The sensors that need only a small bandwidth can use NB-IoT to perform their tasks without overloading the antennas or interfering with the other applications.

5.4 LTE-M

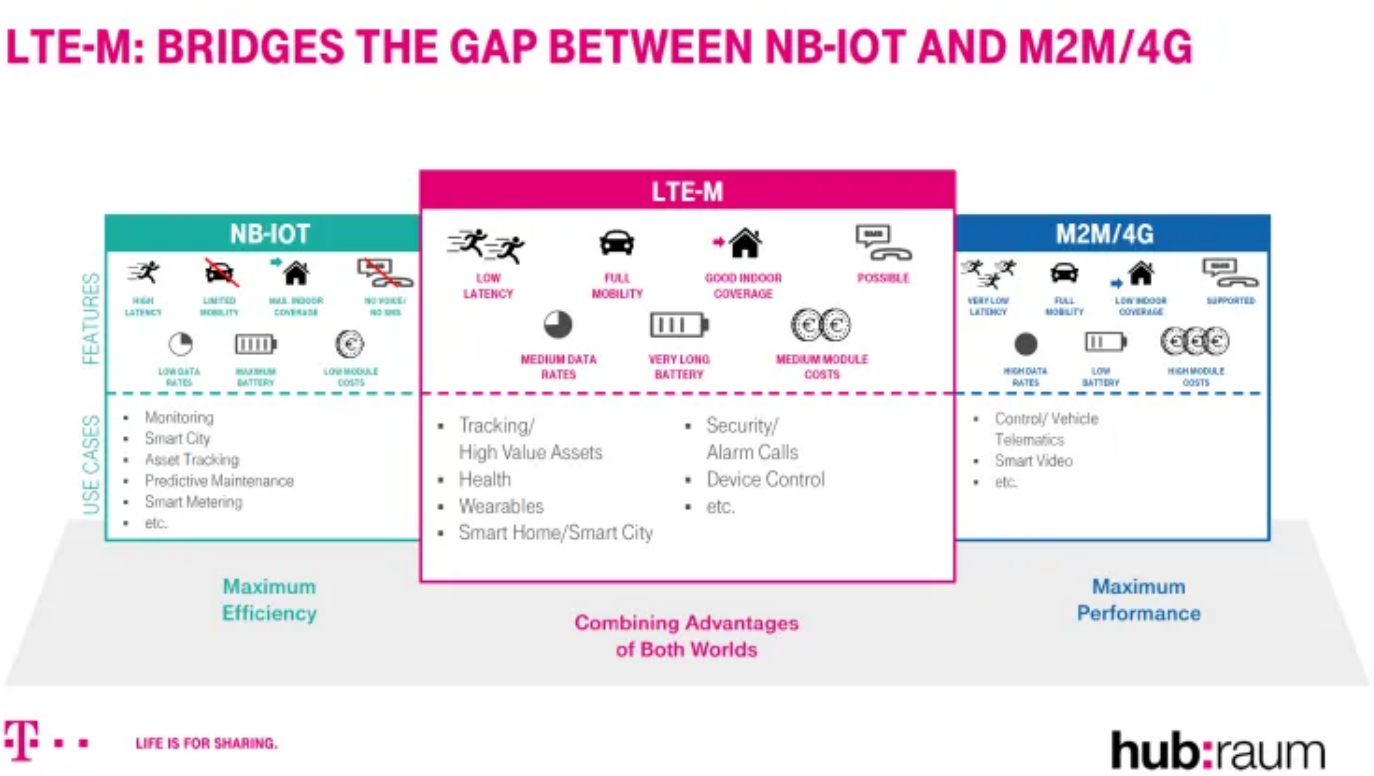

Figure 1.7: LTE-M compared to NB-IoT and 4G showing the advantages of LTE-M that combines qualities of NB-IoT and 4G [24].

As seen in [25], the LTE-M (or LTE CAT M) works similarly to the classic 4G LTE but it uses a clever sleeping process as the NB-IoT to increase the battery life. It uses the low 1.4 MHz frequency to have better coverage than regular LTE but lower than NB-IoT due to its smaller frequency. LTE-M is estimated to have a coverage of 4 times the LTE. This technology is suited for wearable devices as the switching between nodes is very quick (as few seconds as regular LTE). The latency is also relatively small with around a few dozens of milliseconds on average. The figure 1.7 summarises everything by comparing NB-IoT, LTE-M and regular 4G on several points including price, mobility, coverage with typical applications for each connectivity.

6 5G

Before, it was stated that the 4G/LTE has been an important breakthrough for the development of lots of personal systems (phones, computers, camera, and so on). But 4G/LTE has important limits. The most important

one is the big limitation in the number of users for a classical 4G node. Also, the latency for the 4G is around 15-20ms minimum which is not enough for instantaneous services and for the user not to feel the lag.

5G, on the other side, is not limited to personal communications and also allows devices (IoT), sensors, robots, vehicles and so on. Moreover, lots of mobile services with the 5G are expected to work with mobile cloud computing system. The very low latency of the 5G (around 1ms) allows to let clouds make the calculations and send it back to the device, reducing the needed calculation power and space. The same goes for virtual reality and hologram services that cannot emerge without 5G giving sufficient bandwidth. Same for IoT, smart houses, smart traffic control and autonomous cars, health care, disaster monitoring and so on. Lots of applications that need connection simultaneously in a small area can be connected through 5G without impacting all personal devices [26].

In figure 1.8, one can remark the similarity between the LTE and 5G architectures. However, the functional split is completely different. Every piece in the 5G architecture is more complex, has more tasks to perform and handles new functionalities. The most important feature is the decoupling of the computing power. The only physical resource close to the user equipment in 5G is the gNB. After the node, everything is done using operator cloud which allows faster computing, better resource allocation and faster communication. There is also one step less as the data is directly sent to the internet after the UPF (the equivalent of the P-GW) [28].

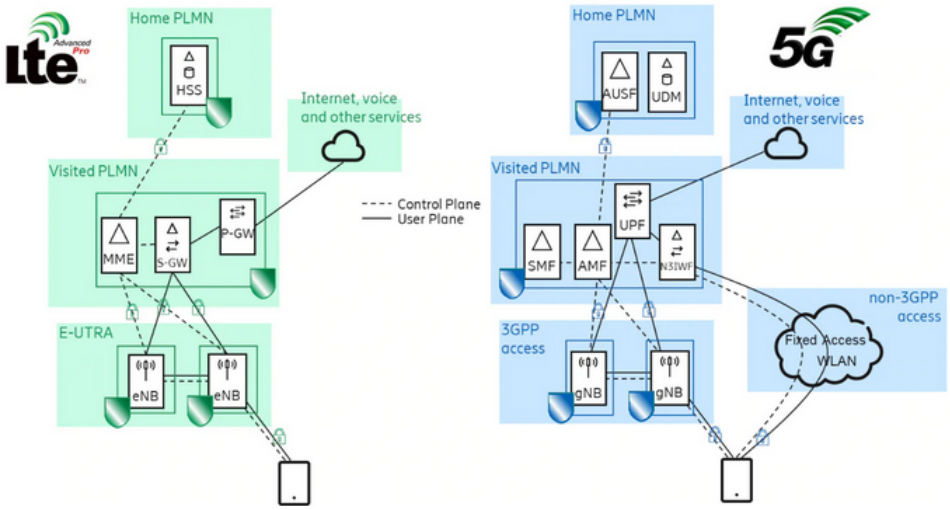

Figure 1.8: Comparison of the 4G and 5G architectures structurally similar but functionally different [27].

As seen in [29], one important difference between 4G and 5G is the frequency spectrum. The 5G network can have frequencies up to 30 GHz which allows again more connections per node. The 5G connectivity allows for ten times more connections per km^2 comparing to the actual 4G. The huge spectrum range leads to different types of connectivity as stated in figure 1.5. The IoT applications that need small bandwidth, low data rate and big coverage can use the smaller frequencies while the bandwidth-consuming devices will transfer data through the higher ones. The data rates are also much higher. The most important feature is the small latency as stated before. As the latency falls below 1ms, we get to a crucial point as lots of systems work with a 1000Hz rate (robotics for example).

7 Possible cellular technologies

Summarising [30], [31], [32], [33] and what was stated previously, the table 1.3 is created while focusing on the needed requirements. Those quantitative values need to be considered as average values. Indeed, the theoretical peak value can be much higher, but the real data rate and delay depend on the distance to the antenna, the current network load, the possible interference and so on. Although those values need to be taken with a pinch of salt, they are useful to find the best-suited technology for our application. The 1G, 2G and 3G are not considered as most European countries have announced that the 3G (and older) antennas are or will be taken down as shown in the article [34].

Connectivity	Simultaneous	Available	Coverage	Power	Delay	Data rate	Cost	Wearable
5G	High	No	Low	Low/High	1ms	1 *Gbps*	High	Yes
NB-IoT	200 000	Yes	High	Low	1-10s	25 *kbps*	Low	No
LTE-M	1 000 000	Yes	High	Low	15ms	300 *kbps*	Low-Medium	Yes
LTE Cat 1	High	Yes	High	Low-Medium	50ms	10 *Mbps*	Medium	Yes
LTE Cat 4	High	Yes	High	Medium	20ms	100 *Mbps*	Medium-High	Yes

Table 1.3: Comparison of cellular technologies based on averaged and qualitative values showing the non-suitability of 5G and NB-IoT.

Due to our wearable equipment tested in Belgium, 5G and NB-IoT can be directly erased from the discussion. From the table 1.3, LTE-M seems to be the best suited cellular connectivity. Although, the three communication types could be sufficient to fulfil our requirements.

However, now in Belgium, there is only one network provider that allows LTE-M connectivity which is Orange. The latter only proposes up to 30 MB/month of usage for the biggest subscribing contract with LTE-M [35]. By a small calculation, it is obvious that this amount of data is too small to ensure a full month of connection: $30MB/96kbps = 2500$ seconds or 42 minutes. Therefore, LTE-M will not be suited due to the lack of a better provider offer.

In conclusion, the best suited cellular technologies are LTE Cat 1 and LTE Cat 4 with both advantages and drawbacks on latency, price, power, or data rates.

8 Communication protocol

Important to note that to send data, the communication needs to be built on a communication protocol. The connectivity is ensured using 4G but the message needs to be wrapped up using a protocol to give the correct way to read to the receiver. For example, each protocol has its specific header and terminator (small ones allow for faster communication but less flexibility). Also, some protocols ask for acknowledgement for each message. The acknowledgement is sent by the receiver to let the sender know that it was well-received. This means that the sending time is twice higher due to the data that needs to go through the path twice.

Now, it is important to compare the different protocols available for the communication between the server and the exoskeleton while keeping in mind the requirements on speed, latency, and the simplicity of implementation. The most known and used protocols especially for IoT are *MQTT*, *CoAP* and *AMQP*. The most used protocols for all applications being *HTTP*, it will be introduced at the same time.

8.1 MQTT

The Message Queue Telemetry Transport (*MQTT*) is a publish-subscribe protocol created in 1999 by IBM [37] and became open-source and standardised in 2013. The protocol works over *TCP/IP* and is defined as the exchange of messages between three different parts: the publisher, subscriber and broker (or server). As seen in figure 1.9, the publishers publish to a specific topic (similar to a channel or a feed) to the broker. Afterwards, the broker will send the published message to all the subscribers of the topic. This is an advantage of the *MQTT* communication. Unlike *HTTP*, the end-user does not need to ask the server

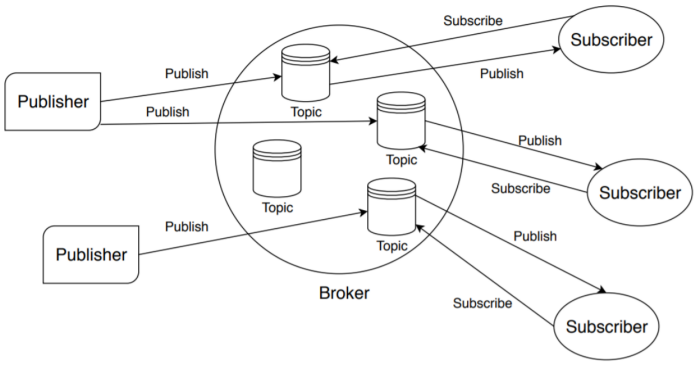

Figure 1.9: Typical architecture of MQTT protocol with the publisher, the broker that contains topics and the subscribers to those topics [36].

regularly to be informed of new message
reception by the server. Moreover, the sending of the message is made by the broker, which is often more powerful than the subscriber, leading to faster communication.

On the advantages of this protocol, one can enumerate lightweight, efficient, reliable and secure [38]. First, MQTT uses very small resources on the clients' side allowing connectivity on small microcontrollers and small sensors. Thus, it is power-efficient. The headers used in the message are small which optimises the bandwidth. Besides, the protocol is reliable. When sending a message, the publisher will precise the Quality of Service (QoS) with different security level: 0 means that the message will be received at most once without requiring a confirmation message from the receiver (called ACK). 1 ensures the reception of the message at least once by waiting for an ACK. Finally, 2 assures that the message will be received exactly once [39].
However, as the *MQTT* works over the *TCP/IP* protocol, whatever the QoS, the publisher will receive a TCP ACK. This allows ensuring the good reception of the package by the server.
Concerning security, to have the right to publish and subscribe to a topic, the client needs to identify itself through an authentication protocol with a username and a password. This way, only the authorised machines can publish and subscribe to the broker's topics. On top of that, *MQTT* is thought to work with *TLS* or *SSL* encryption giving the protection of the message.
One important drawback of *MQTT* is the impossibility to implement peer-to-peer communication. Indeed, every message needs to be sent to the broker which will send it to the subscribers. One may also think about the topic names in the message that add weight to the transmission. This leads to a slow-down of the communication especially in the case of very long topic names. Another disadvantage is the necessity for the subscriber to maintain the communication with the broker to be notified of the changes in the topic. This is due to the *TCP* protocol over which the *MQTT* is built. A clever way to overcome this issue is to simply use *UDP* instead of *TCP*. This is the definition of the *MQTT-SN* (MQTT for Sensor Nodes) that allows sleeping subscriber and uses only two-byte alias making the protocol lighter and faster [39].

This protocol is the preferred one for IoT application as it is easy to implement and it allows thousands of communications at the same time. It is also developed on low-power devices for which *HTTP* is too heavy.

8.2 CoAP

Constrained Application Protocol (*CoAP*) was developed by the Internet Engineering Task Force and ARM and standardised in 2014. It has been created, as its name states, for constrained devices in power and processing capabilities [39]. It is defined as a lighter version of the *HTTP* protocol and uses the same request-response communication model (*REST*). Similarly to the lighter version of *MQTT* briefly introduced in the previous section, it works under the *UDP* protocol as the transport layer. This leads to a tiny packet header of only 8 bytes. The *CoAP* adds only 4 bytes as the application header, allowing smaller messages and faster communication. This protocol is IP based (but compatible only with IPv6), granting easy access to the internet. As seen on figure 1.10, the protocol works very often with *HTTP*. Indeed, the low-power devices run on *CoAP* and the compatibility is ensured by the similar protocol stack and communication model between those two.

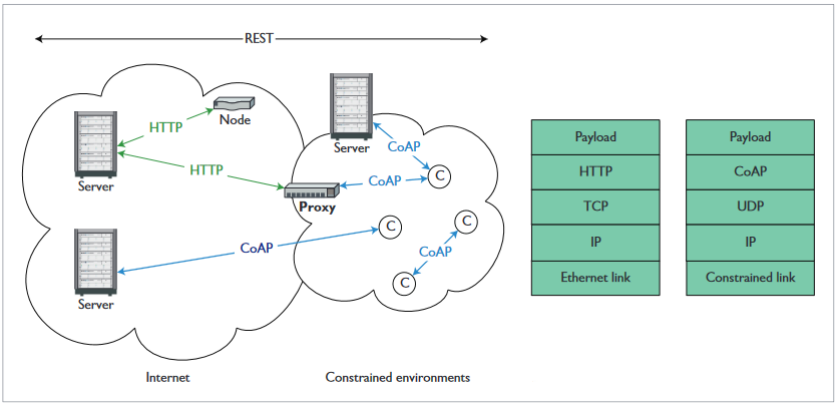

Figure 1.10: Typical Web architecture including both *HTTP* for servers and *CoAP* for low-power nodes [40].

It is a secure protocol due to the easy implementation of *DTLS* or *TLS* with *HTTP* and consequently with *CoAP*.

It is able to work both on reliable and non-reliable communication by the addition of a Confirmable message (*CON*). For each *CON* sent, the server will reply an Acknowledge package (*ACK*) to the sender.
In order to reduce the communication between the server and the client, the *ACK* can be included in the response message of another message (namely *Piggybacked Response*). This means that the server only needs to add 2 bytes to a message that would be sent to the client anyway to confirm the reception of the previous one.
Thanks to the *UDP* transport layer, *CoAP* is lightweight and consumes very low bandwidth.
Peer-to-peer communication does not need a server between the clients which results in lower latency.

This protocol is very often used when the application contains parts with *HTTP* and some low-power devices that need to communicate.

8.3 AMQP

Advanced Message Queuing Protocol (*AMQP*) is another publish-subscribe protocol first created in 2003. It is close to *MQTT* but this one is not a lightweight protocol. As a matter of fact, it has a lot of complexity in the architecture allowing a more powerful communication. It is not limited to small messages as *MQTT*. The size of the messages is not defined, it can be from kBytes to GBytes [41].

As shown in the figure 1.11, there are three elements that establish the transfer of the messages with *AMQP*: The Exchange, Queue and Binding. The Exchange represents the broker which will send the messages to queues depending on their priorities. Binding gives the definition of the priority with special methods including direct, topic and fanout [37].
Similarly to *MQTT*, the security of the transmission is ensured by the Transport Layer Security (*TLS*).

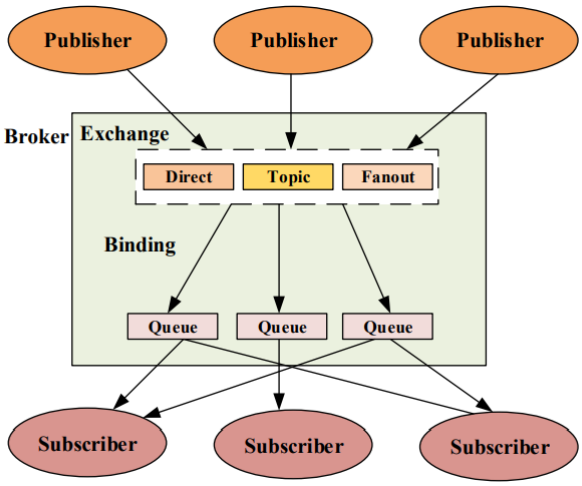

Figure 1.11: Typical AMQP architecture containing publishers, Exchange, Queue and Binding that handle messages and give priorities [37].

The important drawback of this protocol is its weight. It takes a lot of memory, bandwidth, and power to communicate. Therefore, it is not suited for low-power devices. It is often used with Autonomous Computing or Cloud Computing [41].

8.4 HTTP

Hypertext Transfer Protocol (*HTTP*) is the most used protocol especially for the web applications. It works over a simple request and response model, using the TCP transport layer [42]. This protocol is rather old as it has been launched in 1990 [43].
Each request is sent to the server by the means of proxies that will direct the request and return the answer from the server. Normally, the server will never start a request itself but rather only answer to clients' requests.

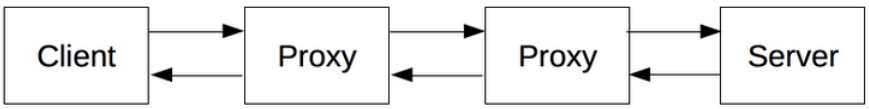

Figure 1.12: Typical HTTP request-response between a client connected to a server through proxies [44].

One of the specificities of *HTTP* is its simplicity. It is even human-readable, meaning easy debugging. Being an old very used protocol, it has been extended with lots of new features as time is flying.

Concerning the IoT application, as explained before, *HTTP* is rather unsuited for low-power devices, which are most often used for IoT. It can be too heavy and *MQTT* or *CoAP* will be almost always preferred.

8.5 Comparison

Due to the requirements, the most critical parameters of the protocol are latency, security, required bandwidth, power efficiency and the current preferences in IoT applications ([45]).

Latency:

The latency with MQTT changes with the QoS, the lowest with QoS 0 and the highest for QoS 2. From [46] and [47], we can estimate the average latency between 0 to 50ms (and up to 100ms for rare cases) depending on the application, the connection type between the broker and the client and the QoS.
Compared to MQTT, CoAP appears to have always in average twice lower latency except for cellular network on which CoAP and MQTT QoS 0 have comparable latency [45].
Similar results to MQTT can be found with AMQP. MQTT has lower latency for smaller message sizes but as the size increases, MQTT tends to AMQP latency.
HTTP is the slowest protocol with an average latency of 100-200ms with the latency increasing with the distance to the server [48].

Security:

The security part is not problematic as all protocol can be easily secure with *TLS* or *SSL*. On this point, all protocols seem to be equivalent. However, AMQP presents an advantage as the security is directly embedded in the protocol.

Required bandwidth:

As the exoskeleton needs to stay connected and to be able to send data in most constrained environments, the required bandwidth is a crucial parameter. Indeed, the lower the bandwidth, the easier the exoskeleton can communicate even on an unstable network. It is also important because the resources from the antenna that will pass the message are limited and IoT applications will grow exponentially in the next years. To avoid any limitations, the smaller the bandwidth, the better it will be for the application.

The bandwidth is linked to the header and everything that wraps up the message. The header sizes for the MQTT, CoAP and AMQP are respectively (at least) 2, 4 and 8 bytes. Nevertheless, from [45] and [39], CoAP appears to needs slightly lower bandwidth than MQTT for almost every configurations except for networks where latency is high. This is explained by the fact that UDP needs smaller bandwidth than TCP. Concerning the others, AMQP needs higher bandwidth than both MQTT and CoAP because of its bigger header and the needed information that wrap the message (exchange, binding, and so on). Finally, the biggest bandwidth protocol is HTTP which uses several times the CoAP bandwidth to send a message.

However, it is important to note that CoAP due to UDP protocol, is more prompt to message loss in busy networks. As opposed to TCP, UDP does not ask for an ACK message for each packet meaning that it does not detect the message losses. If the reliability is considered, the ranking from best to worst performance becomes MQTT, AMQP, CoAP and HTTP.

Power Consumption:

The exoskeleton needs to use the least energy to communicate as it needs to work on a battery. Once again using [45] that resumes a lot of articles on the topic, the lowest consumer is CoAP with slightly better results than MQTT. AMQP taking third place and HTTP being once again the worse protocol energy-efficient wise. This is not surprising as the power consumption is associated with bandwidth usage leading to the same ranking

found in the previous point.

Current applications:

Finally, it is interesting to observe what protocol is the most used for IoT applications. It will guide the search as it leads to more resources, hardware, and software to achieve the project. If we consider the survey conducted by Eclipse in 2018 [49], over 60% of the developers preferred MQTT, followed closely by HTTP with 55% and CoAP and AMQP with around 30 and 25%. MQTT is also seen to be an increasing protocol with the multiplication of IoT applications.

Conclusion:

Characteristics	Best protocol			Worst protocol
Latency over a mobile network	MQTT (QoS 0)	CoAP	AMQP	HTTP
Security	All			
Bandwidth	CoAP	MQTT	AMQP	HTTP
Reliability	MQTT	AMQP	CoAP	HTTP
Power Efficiency	CoAP	MQTT	AMQP	HTTP
Developers preference	MQTT	HTTP	CoAP	AMQP

Table 1.4: Comparisons of the protocols on key required parameters such as latency, security or bandwidth inspired by [45] showing the motivations of MQTT choice.

The project needs real-time monitoring and is connected through a mobile network. To avoid some latency publish-subscribe connectivity is better to only receive a value when it changes. For security reason, the protocol must be reliable to protect the patient. The most used protocol will be the easiest to implement and with the time constraints inherent to this project, it will be preferred. For all those reasons, the best-suited protocol for the current application is the MQTT.

9 Conclusion

After comparison with WiFi, LPWAN (more precisely LoRaWAN, Sigfox and NB-IoT) and cellular connection, the latter is preferred to have a wide coverage with low latency. NB-IoT is a LPWAN, especially devoted to IoT application for low-power devices that do not need low latency or high data rate. Similarly, LTE-M is another IoT oriented communication mean. This combines the advantages of both NB-IoT on power efficiency and cost and 4G on high data rate, low latency and full mobility. Finally, 5G is a new emerging technology that allows latency as low as 1 ms, a very high data rate and connection density. Comparing those three, LTE Cat 1 and 4, one can see that 5G and NB-IoT are declared unsuited due to our requirements. Unfortunately, because of a lack of network provider in Belgium, LTE-M must also be eliminated. Both LTE Cat 1 and 4 are then chosen and a compatible cellular module will be searched.
Concerning the protocol that needs to be implemented on the cellular board, MQTT, CoAP, HTTP and AMQP have been compared. MQTT is a lightweight publish-subscribe protocol, most used in IoT applications. CoAP is the competitor as it also focuses on low-power constraint devices. It is a lighter version of HTTP, built on UDP protocol, based on a request-response communication model. AMQP is a heavier, more complex publish-subscribe protocol. And HTTP is a well-known request-response protocol. Based on latency, security, bandwidth, reliability, power efficiency and the current usage of the developers, MQTT is chosen. Indeed, it allows for the smallest latency over mobile network, while ensuring reliability.

Chapter 2

Implementation

1 Introduction

This chapter is devoted to the system implementation. To that end, the cellular module must be chosen by looking at the MQTT and LTE Cat 1 or 4 compatible boards available on the market. Then, the different parts are implemented. The full project needs communication between Matlab and the cellular module as the exoskeleton is connected to Matlab. To that end, a Simulink system will be implemented. A server needs to be built to allow communication with the module. Finally, an interface is built for quick monitoring and easy message sending by the doctor.
First, the coding of the board is made. The code is implemented to connect to an MQTT server, send data and subscribe to topics. Then, the Simulink module is created to send a message from Matlab to the board by Serial communication. The MQTT server and the linked interface are implemented as well as an FTP server to centralised all servers for the global project at the same place.

2 LTE boards

The best-case scenario would be to build a full standalone micro-controller with a built-in cellular chip with only the needed components for our application to keep the power consumption as low as possible. However, due to the limited development time of 6 months, the connected board will be preferably an already built standalone board or an Arduino or Raspberry pi shield for simplicity. Moreover, as the exoskeleton is working through Matlab, it is better to use Arduino or Raspberry pi for easy communication with Matlab. After research on the material available on the market, three different boards compatible with LTE Cat 1 or 4 and MQTT are found. The latter are compared on their cost, their power consumption, their possible data rates and, once again with the time constraints, on the amount of support available online. The support and documentation available are very important for the learning curve. Indeed, the more support and help can be found online, the less time will be needed to understand the working of the board leading to faster final implementation.

Boards	Cost	Power	Data rates	Online supports
SIM7600CE-T 4G(LTE) Shield for Arduino [50]	60.36 €	Low	100 *Mbps*	High
Shield per Raspberry Pi - SIM7600E-H [51]	70.00 €	Medium	100 *Mbps*	High
Orange Pi 4G-IOT 1G Cortex-A53 8GB [52]	42.25€	High	100 *Mbps*	Low

Table 2.1: Comparison of the cellular boards based on power, data rates, cost and online supports for faster implementation.

The first board on the table 2.1 is an Arduino shield, power-efficient and relatively expensive. The Arduino is less powerful than a Raspberry Pi or an Orange Pi leading to easier development but potential limitations. The two shields are working on the same SIM7600 chip meaning that parts of the tutorials and supports are common for both. The Raspberry Pi is more expensive and more powerful. Also, on the price of the Raspberry Pi shield, a regular Raspberry Pi price needs to be added (between 20 and 40 €). Finally, the Orange Pi is surprisingly inexpensive as it costs half the price of the others for a full standalone LTE, WiFi and Bluetooth compatible board. However, as the board is rather new, few tutorials are available online and it consumes a lot of power.

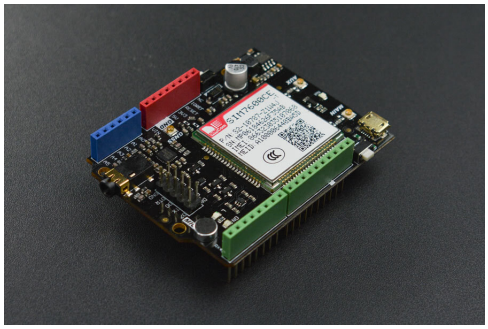

Figure 2.1: The chosen Arduino shield using the SIM7600 cellular chip.

For simpler coding, lower power consumption and cost, the Arduino shield, presented in figure 2.1, is chosen to be the best suited for this project.

3 Project architecture

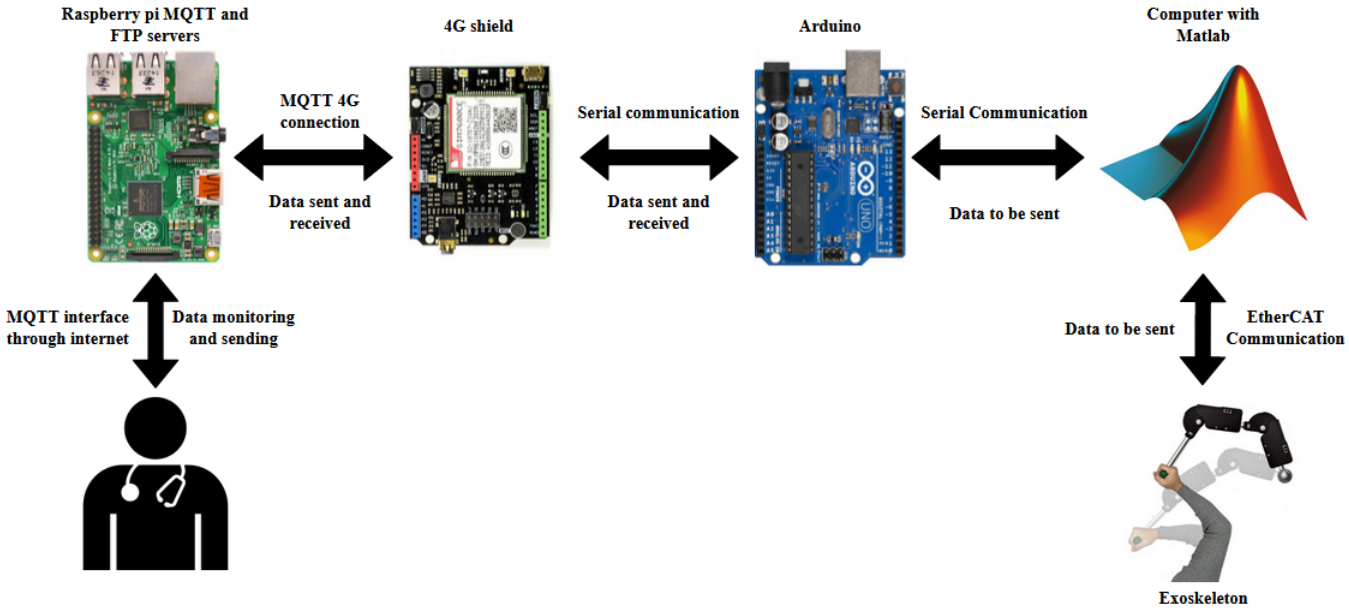

Figure 2.2: Global organisation of the data flow with information coming from exoskeleton to the cellular chip through Matlab and two Serial communications and sent through 4G to the server built on the Raspberry Pi.

In order to make communication effective with the exoskeleton through the cellular network, three different things are mandatory. The first one is Arduino coding. The cellular shield needing an Arduino, the communication always passes through the Arduino. The two are connected using a virtual Serial communication between two pins. Also, the Arduino takes the data from the exoskeleton via its serial port and transfers it to the cellular module after processing.
Then, as the entire exoskeleton is working on Matlab and Simulink, a module needs to be created to easily transfer information between the exoskeleton and the Arduino. The data that needs to be sent through 4G is given by Matlab to the Arduino through the module and the serial port.
Finally, an MQTT server is also needed to make communication possible. Moreover, an interface to allow easy monitoring by the doctor is implemented. Those are created using a Raspberry Pi 2 that has the advantage of being easily plugged in and out, avoiding the necessity of an external server.

4 Arduino coding

The code of the Arduino is using the Botletics SIM7000-LTE-Shield library [53], itself a modified version of the Adafruit_FONA [54]. Important to note that this library is not made for the same shield as the one chosen for this project. Indeed, the SIM7000 is not the same chip as the SIM7600 taken. But, as those chips are from the same family, part of the functions does work on the shield. For the rest, it is slightly more complicated as each command needs to be sent separately. For example, to connect to an MQTT server, the Arduino must send to the chip the following commands:

1. AT+CMQTTSTART
2. AT+CMQTTACCQ = 0, name
3. AT+CMQTTCONNECT = 0, server, 90, 1, username, key

The first command stating that the chip must enable MQTT connection. The second one is attributing the name with which the server is seeing the shield. And finally, the last line giving the server to which the shield is connected, the username and the key for the identification and the authorisation to post on the server.
To send manually the command, the library function *sendCheckReply* is used to ensure that the chip correctly received the command. The list of available commands implemented on the chip is given in the GitHub of *DFRobot* which produces the shields [55]. The code being commented and rather long, it will only be discussed briefly.

First, the different constants are defined, including the server address, the password, the pins and the needed global variable for the code.
After that, in the setup, the shield is turned on, the sim card is unlocked, it is connected to the cellular network, then to the MQTT server and subscribed to the topics.
In the loop, the Arduino is checking if it receives anything either from the game through Matlab or from the MQTT server via the subscription. The received information is processed in separated functions.

Now, each function in the code will be quickly explained to have a clear understanding of the code.

1. netStatus(): It simply returns the strength of the network. This function has been taken from an example of the Botletics library [53].
2. MQTTStart(Name, server, username, key): This function is the example seen before, it enables the MQTT connection on the shield and then connects the shield to the server.
3. MQTTDisconnect(): Rather straightforward, it disconnects the board from the MQTT server to which it has been connected before.
4. MQTTSend(topic, message): The aim of this function is to send a message to a specific topic from the server. To do so, the shield must receive first the topic, then the message and finally the command to publish everything. The message is sent with a QoS of 1 to ensure a good reception by the server.
5. MQTTSub(): It subscribes to all specified topics on the server. After that, it sends a small message to the topics and checks if it receives the messages from the server. If so, it means that the shield is well subscribed to the topics.
6. MQTTReceived(): This is one of the two functions that runs permanently in the loop. As its name states, it checks if the shield received anything from the server. The condition is the reception of "+CMQTTRXSTART" from the shield. If the Arduino is reading this command, it means that a transmission has been made. It will sort the serial buffer to extract the topic and the message. Depending on the topic, different actions are taken.
7. SerialRecieved(): Similarly to the previous one, this function is running at each loop, emptying the serial buffer and processing the message if one has been sent by the game. The flag stating that a new message has been received is the header packing the message in the Matlab model. Then once again, the message is extracted and processed.

8. FTPConnect(server, port, username, password): This function connects the shield to an FTP server using the username and key provided.
9. FTPDisconnect(): It disconnects the shield from the FTP server.
10. FTPGet(dir, fileName): It gets the file at the specified directory on the FTP server.

The three final functions are thought to work with File Transfer Protocol. Those are unused in this project but the game part using FTP to communicate with the database, they have been implemented just in case. Also, FTP communication needs more time and a powerful processor. Using FTP with the Arduino could delay the MQTT or serial communication of seconds which could block the real-time operation.

5 Matlab and Simulink

The Simulink module has been created using the *Instrument Control Toolbox*. Although an official Arduino Toolbox exists on Simulink and Matlab, it does not allow the use of standalone code on the Arduino. It is built to create simple code using Matlab and Simulink but it controls everything. As in this case, only some information needs to be passed through the serial port, the Arduino Toolbox would not be time-efficient. This explains the choice of the *Instrument Control Toolbox* that is not specifically linked to a micro-controller and which allows to easily transfer data from and to the controller [56].

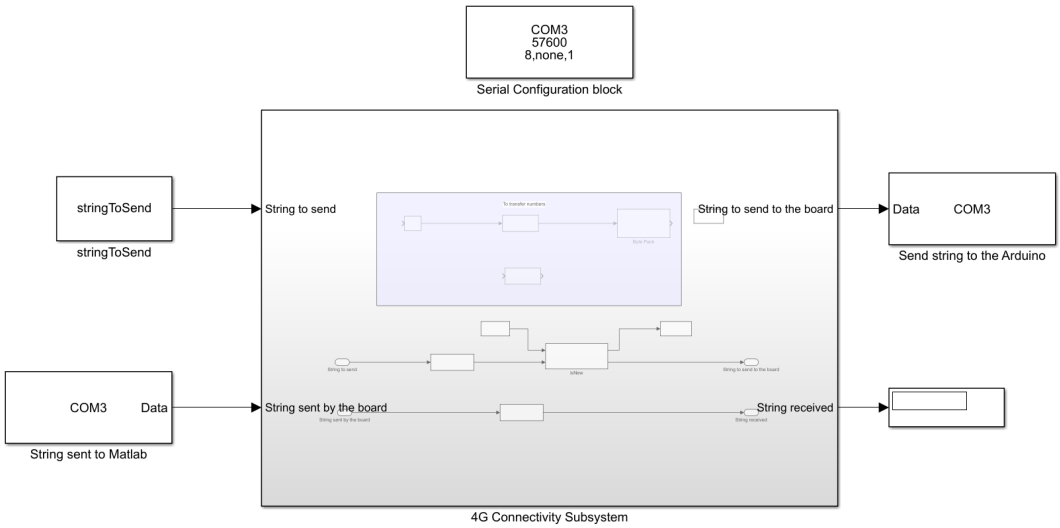

Figure 2.3: The Simulink module allowing to send and receive data through Serial communication using a variable input that allow to send string messages to the board.

The Serial configuration block on the top is used to configure the serial communication by specifying the port, the Baud rate, and the data bits. The Baud rate of the Arduino has been chosen to be 57600 Hz as it is the same as the Baud rate between the shield and the Arduino. This value has been chosen by trial and error. The maximum value is 115200 and the smallest around 9600. This 57600 Hz has been found to be the fastest without any loss of information. Indeed, the communication between the Arduino and the shield is made through the *SoftwareSerial* library that creates a virtual serial port. This library is powerful but also known as not very reliable at a high Baud rate.
The data bits are put to 8 as the data transferred are either numbers or characters. As the ASCII characters are defined by 8 bits [57] and it allows to transfer big numbers, 8 is sufficient in our

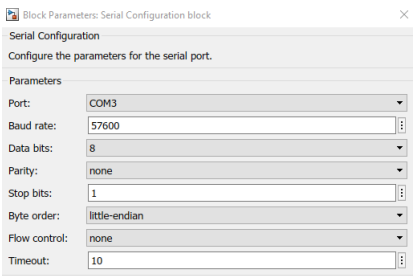

Figure 2.4: Parameters of the configuration block with a baud rate of 57600Hz and 8 bits per data.

case.

As explained in the previous section, the beginning and end of each message are composed of a header and a terminator. In this case, the headers have been selected to 'new' for transmission from Matlab and 'S' from the Arduino. This will indicate to both Matlab and the Arduino that the transmission is beginning. To indicate the end of the messages, in both directions the '\n' character is used. The latter is simply an ASCII newline character. Actually, it is not mandatory to use terminators and headers but the advantage is the clear limitation of the messages. Also, it prevents the message to be cut prematurely.

The value sent to the Arduino can be a string of any length as the size is not predefined in this case. The ex-oskeleton can then send whatever information needed to be quickly transferred to the server and to the doctor. On the other side, the Arduino is sending back the information it receives to prove the good transmission. A display has been added to the module to show the received message from the Arduino.

In the subsystem, a function is implemented to send the *stringToSend* to the Arduino only when it changes and only once. Otherwise, it is replaced by a null character. To send the string and receive it, the string to send is transformed into ASCII values so that the Arduino can understand the character. Then, if Matlab receives a message from the Arduino, the ASCII values received are changed into a simple string so that it can be read on the display.
A simple function has been created *setStringToSend* that allows to easily change the value of the *stringToSend* variable sent to the Arduino during the Simulink simulation using the *set_param* function from Matlab.

6 MQTT server and interface

6.1 MQTT server

As MQTT is a very light protocol, a common way to build a server is to use a Raspberry Pi. The advantages are numerous, no external server is required, it is relatively cheap, it allows for complete flexibility and it is low power giving the possibility to run it 24/7. As the server could run on every type of Raspberry Pi (from zero to 4), the server has been implemented on a Raspberry Pi 2 already available at the lab to avoid any further expenses.

After looking at [58], one can remark that all MQTT brokers have comparable performances. The choice will then be driven by the simplicity of utilisation to ensure fast and easy construction of the server. As it is the most used open-source MQTT broker, the *Eclipse Mosquitto* broker seems to be best suited for the project. Also, it is directly implemented on Linux and consequently on the Raspberry Pi.

Following the tutorials in [59], [60] and [61], the Mosquitto broker is easily implemented on the Raspberry Pi. The Mosquitto broker runs on standalone mode meaning that it starts automatically as soon as the Raspberry Pi boots. The configuration file (mosquitto.conf) has been modified to listen to the 1883 port (the default port for MQTT), the MQTT connection is secured using a password set to *test* and a username which is *jvdelft*. Interesting to note that it is simple to add several passwords and usernames, for example, to allow every doctor to have his own personal username and password.

6.2 Interface

After looking at the choices that are available concerning the dashboards to allow easy monitoring by the doctor, one can see that most of them are not free or only linked to a specified server [62]. However, one simple commonly used platform for flow-based programming is Node-RED. This is an easy open-source programming tool for connecting together various things such as hardware devices, APIs and online services [63]. On the advantages, Node-RED can run independently on the Raspberry Pi regrouping together the server and the interface, the flow editing (connection of the different nodes together) is browser-based meaning that new nodes can be created from anywhere by using the IP address of the Raspberry Pi and it is fully flexible with various modules including a dashboard (*node-red-dashboard*) with classical MQTT elements such as button, slider,

text input and so on.

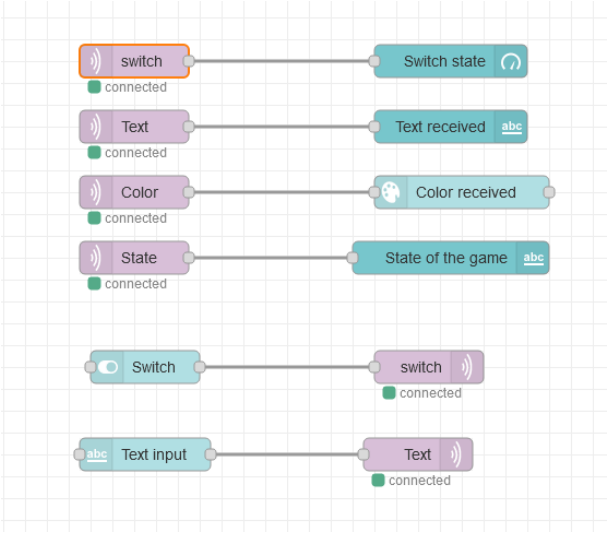

Figure 2.5: Example of development of several nodes sending and receiving data with Node-RED.

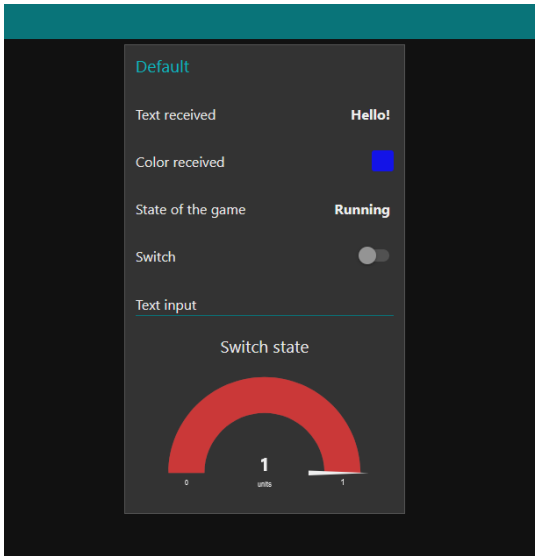

Figure 2.6: Corresponding dashboard created with the flow chart on figure 2.5 with interactive inputs that can be used to send data to the server.

One drawback of Node-RED is that it does not allow for several users at the same moment. If time would allow, the best choice would be to fully develop a dashboard interface software from scratch to allow several doctors with private sessions to monitor different patients. But, in this context with the time constraint, this simple interface will be enough to ensure the good functionality of the project.

6.3 FTP server

An FTP server has also been implemented on the Raspberry Pi so that all servers linked to the exoskeleton are regrouped in one place. The File Transfer Protocol is used by the game to store all the data from the patient to authorise the doctor to monitor each session recorded and see the evolution of the patient. To have more detail on this specific type of connection, one can check the reports of Amaury Deschuymere and Sajl Ghani that treat this subject.

To create a simple FTP server, VsFTPd is installed on the Raspberry Pi. This is an interesting flexible server that ensures the security and privacy of the file using once again a username and a password. Indeed, the Very Secured FTP Daemon is restrictive, it prevents the anonymous from accessing the file and can lock the identified users in a special file without the possibility to go to the parent file. The port used for this server is the regular 22 which is the default one used for secured FTP. The username and the password are the same as the Raspberry Pi so *pi* and *password* respectively. The files can be accessed using external software such as *FileZilla*.

7 Conclusion

The DFRobot SIM7600 Arduino shield has been chosen as it is the most documented and cheapest MQTT and LTE Cat 4 compatible module. An Arduino Uno has been coded to allow for sending data received through Matlab to the MQTT server and for processing the information received by MQTT subscription. The Simulink module has been implemented to send and receive strings of at most 10 characters. A variable string is used as input to change the message to send during the simulation. The servers MQTT and FTP have been created on an available Raspberry Pi for simplicity, using respectively Mosquitto MQTT broker and VsFTPd. Moreover, a Node-RED interface has been implemented linked to the broker and remotely accessible to monitor and to interact with the exoskeleton easily.

Chapter 3

Results

1 Introduction

The results of the implementation are shown in this chapter. All results are measured and analysed with particular attention to the latency as this parameter is important and simple to measure. The first tests were made with the public IO-Adafruit broker which allowed to try sending and subscription. Then, the sending of strings and numbers between Matlab and the Arduino are checked. Finally, the server parts are tested, including the sending and reception between the Node-RED interface and the broker, the sending to the server topics and the subscription to these topics. Moreover, the latency of the server only is measured using a latency measurement program. The final results are shown by combining all parts and checking the good sending and reception of messages in both directions.

2 First tests with IO - Adafruit

In the beginning, the Raspberry Pi servers were not implemented. Therefore, a public MQTT broker has been used. For testing purposes, a free limited server is chosen. One well-known broker is *IO-Adafruit* (accessed through the site `https://io.adafruit.com`) which is the IO branch of the *Adafruit* company. This company is very powerful in the electronics and IoT fields. This broker supports several connection types including HTTP and naturally MQTT. Although the free version only allows for 30 messages/minute, it will be enough for testing. To connect to the broker, a predefined username and a special key given by the server needs to be sent. In our case, the username is defined as *jvdelft* as the one for the Raspberry Pi server and the private key is composed of 32 characters.

The big advantage of the *Adafruit* broker lies in the possibility to create several feeds, group them and display all of them in a dashboard included in the broker. Each feed acts as a topic of the MQTT server. If we group them, it is possible to update all topics at the same time using a simple JSON format. For example, if we have three feeds, Temp, Time and Position, regrouped in one group called Measurement, the feeds can be updated using the following method [64]: The shield must send to the topic ”jvdelft/groups/Measurement” a message as: ”feeds”:{”Temp”:”value 1”,”Time”:”value 2”,”Position”:”value 3”}.

In the testing, 4 feeds have been created, *xPosition*, *yPosition*, *zPosition* and *subscribe*. They are all grouped into the *Default* group on the *Adafruit* broker. The three first are displayed together in a graph as seen in figure 3.1. The fourth one has been used to test the *Subscribe* function of the MQTT connection. Some classical blocks for the MQTT connectivity such as button and slider are also implemented.

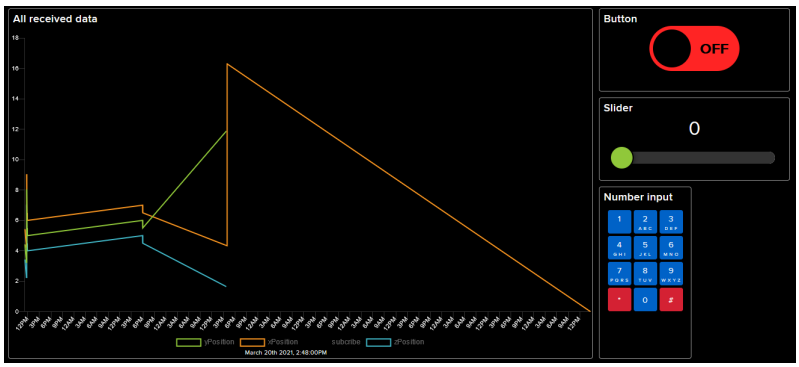

Figure 3.1: The Adafruit dashboard built for testing showing all data received by the server and displaying button, range and number input to interact with the server.

Concerning the testing, one can see in the figure that data were well received. The Arduino was able to send some predefined numbers and text to a particular feed or to several feeds through the group. The subscribing function was also working. After subscribing to the *subscribe* topic, the Arduino received the data sent from the server.

However, a particular latency was experienced to send a message. It was very often around 300 to 400 milliseconds which could be big for the real-time application. But this could be linked to an inherent limitation of the free version of the broker.

In conclusion, the testing was successful, the Arduino was able to send and received data to and from the broker. The broker is relatively well documented and easy to use which was helpful for a first touch with the MQTT brokers. Ultimately, more testing is needed with the final Raspberry Pi broker to check the latency.

3 Matlab communication

3.1 Number transfer

First, the sending of numbers is tested. To do so, a sine wave is sent to the Arduino. The sine wave is discretized using a zero-order holder for a period of 0.3 seconds. After that, the signal is transformed into a single and packed into a 4-byte pack as each received float is defined by 4 bytes in the implemented Arduino code. The Arduino receives the message and resends it back to Matlab to check the accuracy of the reception. The Arduino checks the new reception every 0.3 seconds.

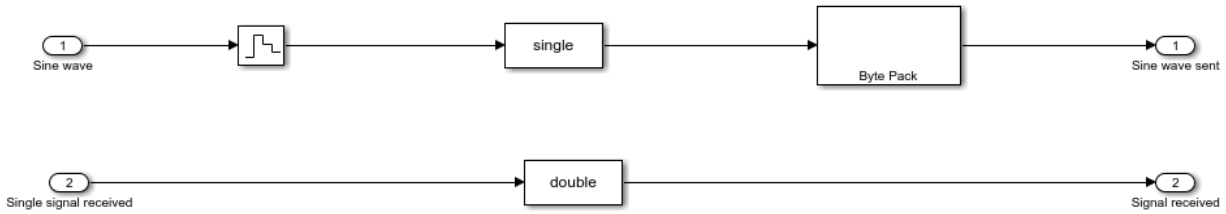

Figure 3.2: Processing of the number to be sent by changing it into single and packing it into 4-byte packs.

The two signals are compared in a scope. The one sent by the Arduino is delayed by 0.3 seconds due to the zero-order holder. One can see that the signal seems to be well received. To choose a particular number to send, a variable can be created and put at the input and a simple function can be used to change the variable during the simulation. Although this project needs the sending and receiving of strings by Matlab, this method for numbers has been implemented for further applications.

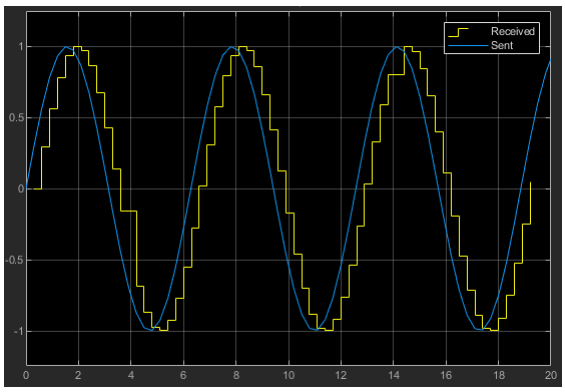

Figure 3.3: Comparison of the sent (blue) and received (yellow) sine wave proving the good reception.

3.2 String transfer

As stated in section 5 of the previous chapter, the Simulink subsystem is used to transfer strings between Matlab and the Arduino. An important drawback of Simulink is that the messages need to be of fixed length. This length is put to 10 in this project to have a sufficient length for most messages. This length is specified in the string to ASCII block and in the Serial receive block. From Matlab, any message of length from 0 to 10 characters can be sent to the Arduino. However, on the other side, Matlab needs to receive a fixed-length message of 10 characters to process it.

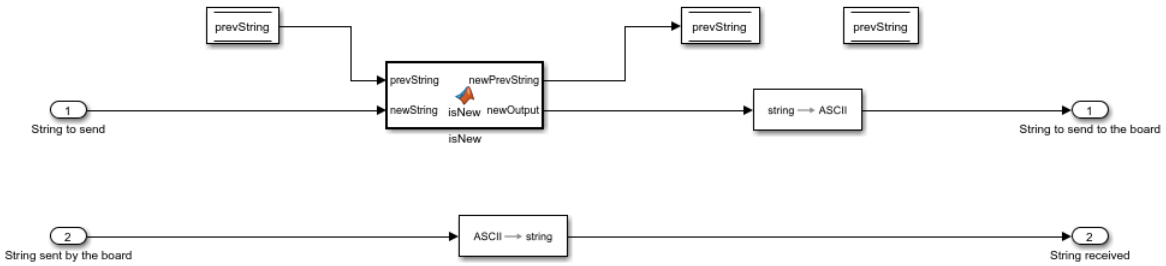

Figure 3.4: Processing of the string to be sent by checking if the message has not already been sent and in that case transforming it into an ASCII vector.

A display is set at the output of the subsystem block to see the received message. In the Arduino, a code is implemented to take all characters and add the space character at the end to allow for a fixed 10 char length. For example, if the Arduino receives the message "*Hello*", it will send back to Matlab the message "*Hello* " so that Matlab is able to receive it. Through the display, it is seen that the message is well received as shown in figure 3.5.

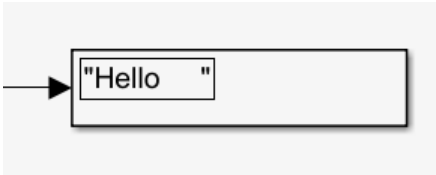

Figure 3.5: Response of the Arduino after the "*Hello*" message received by Matlab.

After sending few messages through the *setStringToSend(string)* function, Matlab seems to send and receive messages efficiently. To have a clear view of the time taken between the sending and the receiving after processing of the Arduino, two figures are made. The first one (figure 3.6) shows the norm of the char vector sent and received with respect to the simulation time. The size of the message is varying with messages from "Hi" to "HelloWorld". The normed is used simply to show the sending of a new message. With the Matlab function implemented, one can see that the message is sent to the Arduino only once. Regarding the reception, the Arduino sends the received message to Simulink and Simulink will simply keep

the received value in the display. This explains the difference between the two signals in figure 3.6.

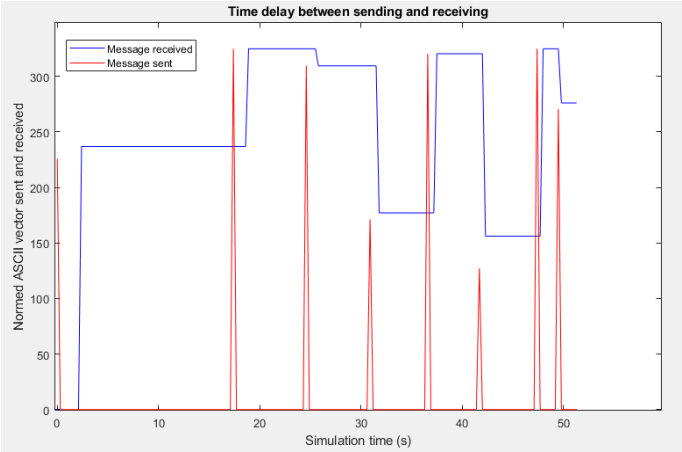

Figure 3.6: Evolution of the normed vectors sent and received used to see the sending of new messages.

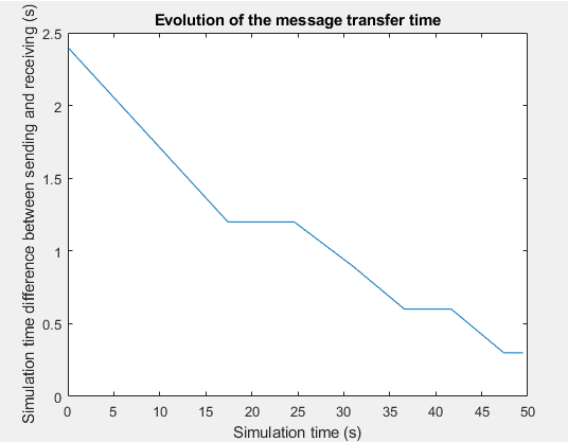

Figure 3.7: Evolution of the time taken by message transfer which rapidly decreases until its minimum value after board starting time.

By comparing the time between the sending peak and the reception, the evolution of the time taken can be shown. As stated in the figure 3.7, the needed time is rapidly decreasing as the Arduino is starting. This may be linked to the starting process of the Arduino and the shield. However, the time shown in the figures is the simulation time which is not representative of real-time. Indeed, the simulation is depending on the sending and receiving from the Arduino. This means that at some point, the simulation time increases faster than the real-time and at other points slower. Nevertheless, on average, the two times are rather similar as the sampling time in Matlab is chosen to 0.3s to correspond with the Arduino communication. This leads to a minimum time difference of less than 0.3 seconds between the reception and the sending of the message. One can easily see that this minimum is reached at the end of the figure 3.7 after 45 seconds of simulation.

In conclusion, the implemented Simulink subsystem can send and receive both numbers and strings. The communication is fast with the optimum simulation time reached when everything is started. The message can be set using the *setStringToSend(message)* during the simulation. However, Simulink is introducing a limitation on the reception as the message needs to be 10 characters long to avoid the blocking of the simulation time and the misunderstanding of the message. This can be restricting for the communication. Also, Simulink is introducing a small delay to send the message to the Arduino.

4 MQTT server and interface

4.1 Sending to server

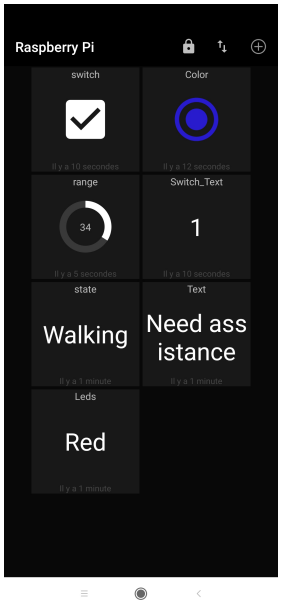

Figure 3.8: Sent messages using the *MQTT Dash* application on different server topics.

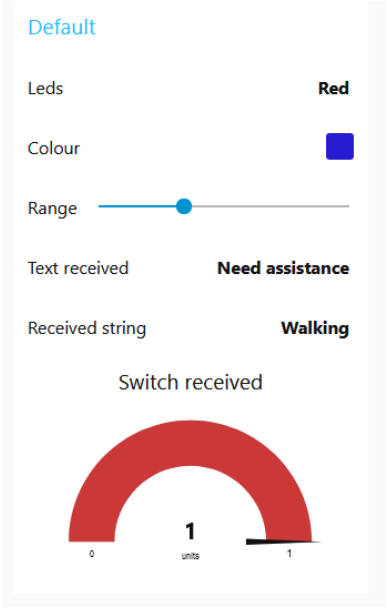

Figure 3.9: Received messages displayed on the Node-RED dashboard on the Raspberry Pi proving the right reception by the server.

The MQTT server is created using Mosquitto on the Raspberry Pi and is running permanently in the background. To test the server, a simple application is found on the internet for mobile phones that allows sending different types of messages, from switch to colour or text. This application is called *MQTT Dash* on Google Play. To test the reception of the server, the Node-RED dashboard previously created is used. The server is well receiving the messages. The reception seems to be instantaneous. The messages can be of any type and the server has no problem to receive and display them in the Node-RED dashboard as seen on the following figures.

Similarly, the server is well receiving the messages that the Arduino is sending. Regarding the latency, the message is experiencing a latency of around 170 ms and it is not really changing with the message size as seen in the figure 3.10. However, this time also considers the time taken by the 4G shield and the Arduino to create the message (including the header, terminator, and so on). Surprisingly, one can see on the figure 3.11 that most of the time is not taken by the sending of the message itself but by its formation.

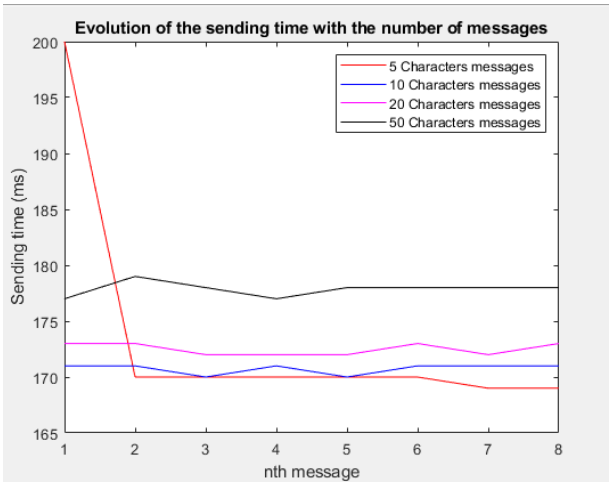

Figure 3.10: Evolution of the time taken by the sending of messages displaying the difference in sending time between various sized messages.

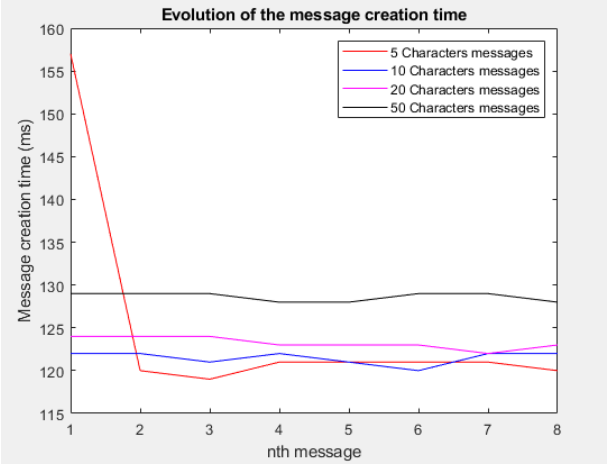

Figure 3.11: Evolution of the time taken by the creation of the messages to send proving the huge impact of the message formation on the overall sending time.

Characters in the message	Averaged sending time (ms)	Averaged creation time (ms)
5	169.71	120.43
6	169.63	120.75
7	169.5	120.88
8	170.13	120.88
9	170.13	121.38
10	170.75	121.5
20	172.5	123.25
50	177.88	128.63

Table 3.1: Averaged time for sending and forming messages showing that around 70% of the time is taken by message creations.

As seen on table 3.1, most of the time is taken by the formation of the message and this formation time is increasing with the size of the message. More testing would be needed to see whether this time could be reduced by testing the shield with a more powerful Arduino board (such as a Mega) or with a Raspberry Pi, or the shield itself is the limiting parameter.
Nevertheless, the latency being less than 200 ms is sufficient to fulfil our requirements. This is enough to quickly communicate and even ensure the security of the patient in case of emergency.

4.2 Subscription and reception from the server

On the other side, to test the subscription part of the server, the same manipulation is used. The Node-RED interface or the phone application is used to send messages to the server topics on which the Arduino is subscribed. For testing purposes, the Arduino is subscribed to two different topics. The first one that will control its built-in LED. And the other called *Led* which is used to turn on and off several LEDs. A small set-up has been created with 8 LEDs, 5 blue in line, and then one red, one blue and one green. By sending a number to the topic between 1 and 8, the Arduino will receive the message and process it to turn on the number of LEDs specified. Similarly, if a colour is sent (Green, Red or Blue), only the corresponding LED will be turned on. For example, if we send ”6”, the first 6 LEDs starting on the right will turn on so the 5 LEDs and the green one. Another example, if the Arduino receives ”Blue”, it will only turn the second LED starting on the left. To turn off the LEDs after turning on, we can simply send ”stop”. The received message is also sent to another topic by the Arduino to show the good reception of the message. As a picture is worth a thousand words, the following link leads to a testing video of the LEDs system: `https://youtu.be/ZdoZL-N9F24`.

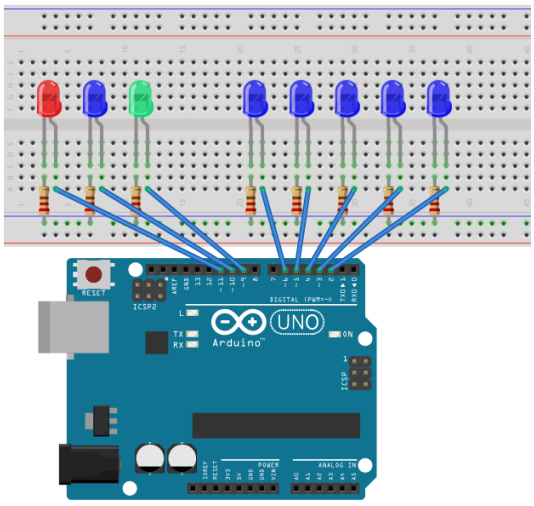

Figure 3.12: Experimental set-up used to check the subscribing of the shield by turning on and off specified LEDs.

After testing, one can see that the connection seems to work efficiently. The latency is visually very low. Indeed, The sending is made by the powerful Raspberry Pi. The only time taken is needed to process the messages and most of the latency is not taking Arduino computing time. To compute this latency, the time between the reception of the message by the Arduino and the message being processed is taken for various size of messages. One can observe that the processing time is linearly increasing with the number of characters in the message. As seen in the table 3.2, this time is from 10 ms to 20 ms which is rather low compared to the 200 ms for sending.

Number of characters	Processing time (ms)
5	9
10	10
20	12
30	14
40	16
50	18

Table 3.2: Processing time for various message sizes showing the small impact of subscription on computing time.

4.3 Latency of the server

Regarding the latency of the server only (and not the cellular connection or the Arduino processing of the message), a test is performed. Using the *mqtt-bim-latency* latency measure tool program found in [65], the latency can be computed for both the publishers to send messages and subscribers to receive messages from the server. The test is performed with 10 clients and 10 subscribers. Each client is sending 1000 messages that one subscriber receives. Those numbers are chosen relatively high to have representative data. To have a closer test to the real situation of this project, the size of each message is set to 10 bytes. Naturally, the public IP address of the Raspberry Pi is used, the local IP would lead to faster communication but the Arduino is not using the local network. Finally, to be able to measure the time taken, the sending and the subscription are set with QoS of 1. The line of code is then:
mqtt-bm-latency –broker tcp://xx.xxx.xxx.xxx:1883 –username jvdelft –password test –count 1000 –size 10 –clients 10 –format text

This gives after few seconds of running code the figure 3.13. One can see that the average latency is 12.785 ms which is really low. Another important conclusion that can be drawn from those results is that all messages are well received whatever the latency. Indeed, the latency seems to have a very high maximal value but it is totally normal to have such latency for rare cases that happen for one or two messages out of the dozens.
Similarly for the subscribers, they receive the messages from the server with an average latency of 16.734 ms. From the previous section, it is seen that the total latency for the subscription part (the sending of the server and the processing of the reception by the Arduino) is less than 40 ms.

```
================ TOTAL PUBLISHER (10) ================
Total Publish Success Ratio:   100.000% (10000/10000)
Total Runtime (sec):           13.047
Average Runtime (sec):         12.810
Pub time min (ms):             0.000
Pub time max (ms):             469.718
Pub time mean mean (ms):       12.785
Pub time mean std (ms):        0.242
Average Bandwidth (msg/sec):   78.089
Total Bandwidth (msg/sec):     780.891

================ TOTAL SUBSCRIBER (10) ================
Total Forward Success Ratio:      100.000% (10000/10000)
Forward latency min (ms):         0.000
Forward latency max (ms):         477.534
Forward latency mean std (ms):    2.214
Total Mean forward latency (ms):  16.734
```

Figure 3.13: Results of the latency test proving the high reliability and low latency of the server.

In conclusion, in both directions, the server does not introduce an important delay. Especially compared to the Matlab subsystem and the time needed to create the message, we can conclude that the impact of the server over the communication latency is relatively small.

5 FTP server

This part is very briefly describing the results concerning the FTP server. For more details, please refer once again to the reports of Amaury Deschumere and Sajl Ghani. The server has been tested with the game data. At the end of each session, the game is saving all the data in a file that is sent to the server. This functionality is also working. In this part, there is no big constraint regarding latency, data rate or whatsoever. No big advantages and drawbacks are considered and the final conclusion is that the server is able to send and receive data.

6 Final results

All parts are working independently. The final tests are held by combining all pieces together. At the same time, some messages are sent through Matlab to the Arduino and others are sent to the server on which the Arduino is subscribed. The Arduino must process each information, send and receive messages at the same time while playing on the LEDs to show the good reception of the messages.

As the subscription part is not introducing a lot of processing time, the final delay to send a message is around 300 ms, close to the time found with only the sending. This time includes the sending to the Arduino through the Simulink model, the processing of the received message due to the subscription, the processing due to the message received from the Simulink and the sending of the latter to the MQTT server. Overall the latency is relatively low which is enough to ensure the security of the patient. The Arduino is able to send and receive all messages quickly. The Simulink subsystem is able to send whatever messages from 0 to 10 characters or even

directly numbers and the size of the message can be easily increased by modifying slightly the Simulink model and the Arduino code. And the doctor can easily communicate with the Arduino remotely using the Node-RED interface created for that purpose.

However, it happens from time to time that if a message is received by the subscriptions when the message from Matlab is processed and sent to the server, the message is not seen by the Arduino. This means that some messages can be not received by the Arduino when it is busy sending messages. As stated before, it could be interesting to test with a more powerful micro-controller to see if this problem is inherent to the power limitation of the Arduino.

Due to the covid-19 situation, it was not possible to test the entire system with the exoskeleton. One thing that needs to be done for further implementation is to connect the real-time system of the exoskeleton with the created subsystem used for communication. Also, all functionalities were implemented with arbitrary values (size of messages, topics on the server, actions for received message by subscription and so on), one needs to consider the exact needs of the exoskeleton for a certain situation and adapt the communication system to that end.

7 Conclusion

The aim of this project was to research and implement cellular IoT connectivity on an exoskeleton.

First, research was conducted in order to find the best-suited cellular connectivity and communication protocol. Concerning cellular connectivity, between 5G, LTE Cat 1, LTE Cat 4, NB-IoT and LTE-M, due to the low latency and availability requirements, LTE Cat 1 and 4 were preferred. Similarly, after a comparison between two publish-subscribe protocols (MQTT and AMQP) and two request-response protocols (CoAP and HTTP), MQTT was chosen because of its low latency, high reliability and current usage in the IoT field. The MQTT and LTE Cat 4 compatible DFRobot SIM7600 Arduino shield has been bought on the market due to the available supports, cost and power efficiency.

Then, to achieve the project, three different things needed to be implemented: the Simulink module, the Arduino and the servers. A Simulink subsystem has been created using Serial communication blocks to send and receive strings of 10 characters length and numbers between the Arduino and Matlab. The Arduino code has been implemented to send the received messages from Matlab to the server through MQTT and 4G. It is also able to process the information received by subscription on the server topics. The MQTT and FTP servers were built using Mosquitto and VsFTPd on a Raspberry Pi with a linked Node-RED interface to allow remote monitoring by the doctor.

Everything put together led to a fully IoT enabled 4G module that can be used for monitoring of an exoskeleton assisted patient that needs real-time healthcare. The overall latency is around 300 ms between the input of the message in Matlab to the reception on the MQTT server, which is sufficiently fast. However, due to the actual context, real testing on the exoskeleton was not performed and the final link between communication and exoskeleton systems still needs to be made.

8 Acknowledgements

I want to thank my promoters Prof. Bram Vanderboght and Prof. Dirk Lefeber for the nice subject proposal and their advice during the mid-term presentation. Furthermore, I am grateful to Prof. Bram Vanderboght for commenting and guiding my final report improving its readability.

I would also like to express my gratitude to Dr. Ir. Victor Grosu for his help and guidance during the entire project. Especially due to this particular covid context, his help and motivation came along at just the right time. Moreover, I want to thank him for the kindly given Raspberry Pi that was needed to create the servers.

References

[1] Eurostat. *Population structure and ageing*. https://ec.europa.eu/eurostat/statistics-explained/index.php/Population_structure_and_ageing. Aug. 2020. (Visited on Apr. 25, 2021).

[2] Eurostat. *Disability statistics - elderly needs for help or assistance*. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Disability_statistics_-_elderly_needs_for_help_or_assistance. June 2019. (Visited on Apr. 25, 2021).

[3] Eurostat. *Disability statistics*. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Disability_statistics. Aug. 2019. (Visited on Apr. 25, 2021).

[4] Daniel H de la Iglesia et al. "Connected Elbow Exoskeleton System for Rehabilitation Training Based on Virtual Reality and Context-Aware". eng. In: *Sensors (Basel, Switzerland)* 20.3 (2020), pp. 858–. ISSN: 1424-8220.

[5] Enrique Bances et al. "Exoskeletons Towards Industrie 4.0: Benefits and Challenges of the IoT Communication Architecture". In: *Procedia Manufacturing* 42 (2020). International Conference on Industry 4.0 and Smart Manufacturing (ISM 2019), pp. 49–56. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2020.02.087.

[6] BICS. *Improve efficiency and throughput with seamless connectivity for intelligent exoskeletons*. https://bics.com/use-case/connected-exoskeleton-use-case/. 2020. (Visited on Mar. 15, 2021).

[7] Yujun Ma et al. "Robot and cloud-assisted multi-modal healthcare system". eng. In: *Cluster computing* 18.3 (2015), pp. 1295–1306. ISSN: 1386-7857.

[8] Erik Dahlman, Stefan Parkvall, and Johan Sköld. "Chapter 1 - Background of LTE". In: *4G LTE/LTE-Advanced for Mobile Broadband*. Ed. by Erik Dahlman, Stefan Parkvall, and Johan Sköld. Oxford: Academic Press, 2011, pp. 1–13. ISBN: 978-0-12-385489-6. DOI: https://doi.org/10.1016/B978-0-12-385489-6.00001-1.

[9] Indika Balapuwaduge and Frank Li. "Cellular Networks: An Evolution from 1G to 4G". In: July 2018. ISBN: 978-3-319-32903-1.

[10] Ken's Tech Tips. *Download Speeds: What Do 2G, 3G, 4G 5G Actually Mean?* https://kenstechtips.com/index.php/download-speeds-2g-3g-and-4g-actual-meaning. Nov. 2018. (Visited on Feb. 1, 2021).

[11] 5gNotes. *History of Cellular Mobile phones and technology*. https://5gnotes.com/history-of-cellular-mobile-phones-and-technology/. 2015. (Visited on Feb. 1, 2021).

[12] Adam Fendelman. *1G, 2G, 3G, 4G, 5G Explained*. https://www.lifewire.com/1g-vs-2g-vs-2-5g-vs-3g-vs-4g-578681. June 2020. (Visited on Feb. 1, 2021).

[13] Chinmay Chakraborty and Joel J. C. P Rodrigues. "A Comprehensive Review on Device-to-Device Communication Paradigm: Trends, Challenges and Applications". eng. In: *Wireless personal communications* 114.1 (2020), pp. 185–207. ISSN: 0929-6212.

[14] Yongwan Park and Fumiyuki Adachi. "Enhanced Radio Access Technologies for Next GenerationMobile Communication". In: 2007. Chap. Overview of Mobile Communication. ISBN: 978-1-4020-5532-4. DOI: 10.1007/1-4020-5532-3.

[15] Intel. *Different Wi-Fi Protocols and Data Rates*. https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html. Mar. 2021. (Visited on May 1, 2021).

[16] Toni Adame, Marc Carrascosa, and Boris Bellalta. "Time-Sensitive Networking in IEEE 802.11be: On the Way to Low-latency WiFi 7". eng. In: (2019). arXiv: 1912.06086 [cs.NI].

[17] Kais Mekki et al. "A comparative study of LPWAN technologies for large-scale IoT deployment". eng. In: *ICT express* 5.1 (2019), pp. 1–7. ISSN: 2405-9595. DOI: doi.org/10.1016/j.icte.2017.12.005.

[18] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. "A survey on LPWA technology: LoRa and NB-IoT". In: *ICT Express* 3.1 (2017), pp. 14–21. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2017.03.004.

[19] ublox. *LTE Cat 6*. https://www.u-blox.com/en/technologies/lte-cat-6. 2018. (Visited on Mar. 15, 2021).

[20] GSMArena. *Network coverage in BELGIUM*. https://www.gsmarena.com/network-bands.php3?sCountry=BELGIUM. 2020. (Visited on Feb. 10, 2021).

[21] Mohammad Zulhasnine, Changcheng Huang, and Anand Srinivasan. "Efficient Resource Allocation for Device-to-Device Communication Underlaying LTE Network". In: *6th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications,* (Nov. 2010), pp. 368–375. DOI: 10.1109/WIMOB.2010.5645039.

[22] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. "A survey on LPWA technology: LoRa and NB-IoT". eng. In: *ICT express* 3.1 (2017), pp. 14–21. ISSN: 2405-9595.

[23] Hassan Malik et al. "Radio Resource Management Scheme in NB-IoT Systems". eng. In: *IEEE access* 6 (2018), pp. 15051–15064. ISSN: 2169-3536.

[24] GSMA. *Exclusive Interview: High Potential Prototyping*. https://www.gsma.com/iot/exclusive-interview-high-potential-prototyping/. 2018. (Visited on Jan. 17, 2021).

[25] Jia-Ming Liang, Po-Yen Chang, and Jen-Jee Chen. "Energy-efficient scheduling scheme with spatial and temporal aggregation for small and massive transmissions in LTE-M networks". In: *Pervasive and Mobile Computing* 52 (2019), pp. 29–45. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2018.11.002.

[26] Howon Lee Heejung Yu and Hongboem Jeon. "What is 5G? Emerging 5G Mobile Services and Network Requirements". In: *Sustainability* (Oct. 2017). DOI: 10.3390/su9101848.

[27] Noamen Ben Henda, Monica Wifvesson, and Christine Jost. *An overview of the 3GPP 5G security standard*. https://www.ericsson.com/en/blog/2019/7/3gpp-5g-security-overview. 2019. (Visited on Feb. 1, 2021).

[28] Nadine Akkari and Nikos Dimitriou. "Mobility management solutions for 5G networks: Architecture and services". In: *Computer Networks* 169 (2020), p. 107082. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2019.107082.

[29] Qorvo. *Getting to 5G: Comparing 4G and 5G System Requirements*. https://www.qorvo.com/design-hub/blog/getting-to-5g-comparing-4g-and-5g-system-requirements. Sept. 2017. (Visited on Jan. 17, 2021).

[30] Intel Network Academy. *Overview and Comparison of Cellular Technologies*. https://www.coursera.org/lecture/network-transformation-101/overview-and-comparison-of-cellular-technologies-MCxc8. 2020. (Visited on Mar. 15, 2021).

[31] Siretta. *3G vs LTE Cat 1*. https://www.siretta.com/2019/04/3g-vs-cat-1/. 2019. (Visited on Mar. 15, 2021).

[32] European Commission. *Comparison of technologies*. https://ec.europa.eu/digital-single-market/en/comparison-technologies. 2020. (Visited on Mar. 15, 2021).

[33] Olof Liberg et al. "Chapter 9. The competitive Internet of Things technology landscape". In: *Cellular Internet of Things*. Oxford: Academic Press, 2017. ISBN: 978-0-12-812459-8.

[34] Westbase.io. *3G Sunset in Europe: Planning Your Next Steps*. https://www.westbase.io/3g-sunset-in-europe-planning-your-next-steps/. 2019. (Visited on Mar. 1, 2021).

[35] Orange. *IoT - How can we help you?* https://business.orange.be/en/it-solutions/internet-things. 2020. (Visited on Feb. 1, 2021).

[36] Francesco Buccafurri, Vincenzo De Angelis, and Roberto Nardone. "Securing MQTT by Blockchain-Based OTP Authentication". English. In: *Sensors* 20.7 (2020), p. 2002. DOI: 10.3390/s20072002.

[37] Lilia Tightiz and Hyosik Yang. "A Comprehensive Review on IoT Protocols' Features in Smart Grid Communication". In: *Energies* 13.11 (2020), p. 2762.

[38] MQTT.org. *MQTT: The Standard for IoT Messaging*. https://mqtt.org/. 2020. (Visited on Apr. 6, 2021).

[39] Muhammad Harith Amaran et al. "A Comparison of Lightweight Communication Protocols in Robotic Applications". In: *Procedia Computer Science* 76 (2015). 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IEEE IRIS2015), pp. 400–405. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.12.318.

[40] Carsten Bormann, Angelo P. Castellani, and Zach Shelby. "CoAP: An Application Protocol for Billions of Tiny Internet Nodes". In: *IEEE Internet Computing* 16.2 (Mar. 2012), pp. 62–67.

[41] Jorge E. Luzuriaga et al. "Testing AMQP Protocol on Unstable and Mobile Networks". In: *Internet and Distributed Computing Systems*. Ed. by Giancarlo Fortino et al. Cham: Springer International Publishing, 2014, pp. 250–260. ISBN: 978-3-319-11692-1.

[42] Min Che and Ming Fu Tuo. "Server Program Analysis Based on HTTP Protocol". eng. In: *MATEC web of conferences* 63 (2016), pp. 5023–. ISSN: 2261-236X.

[43] Ping Zeng et al. "A multi-pattern hash-binary hybrid algorithm for URL matching in the HTTP protocol". eng. In: *PloS one* 12.4 (2017), e0175500–e0175500. ISSN: 1932-6203.

[44] MDN Web Docs mozilla. *An overview of HTTP*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. 2021. (Visited on Apr. 8, 2021).

[45] Dimitrios Glaroudis, Athanasios Iossifides, and Periklis Chatzimisios. "Survey, comparison and research challenges of IoT application protocols for smart farming". In: *Computer Networks* 168 (2020), p. 107037. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2019.107037.

[46] Veeramanikandan M. and Suresh Sankaranarayanan. "Publish/subscribe based multi-tier edge computational model in Internet of Things for latency reduction". In: *Journal of Parallel and Distributed Computing* 127 (2019), pp. 18–27. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.01.004.

[47] Amartya Mukherjee, Nilanjan Dey, and Debashis De. "EdgeDrone: QoS aware MQTT middleware for mobile edge computing in opportunistic Internet of Drone Things". In: *Computer Communications* 152 (2020), pp. 93–108. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.01.039.

[48] Mark Obren et al. "The tyranny of distance prevails: HTTP protocol latency and returns to fast fibre internet access network deployment in remote economies". eng. In: *The Annals of regional science* 52.1 (2014), pp. 65–85. ISSN: 0570-1864.

[49] Eclipse Foundation. *IoT Developer Survey Results*. https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2018.pdf. 2018. (Visited on Mar. 15, 2021).

[50] DFRobot. *SIM7600CE-T 4G(LTE) Shield for Arduino*. https://www.dfrobot.com/product-1834.html. 2021. (Visited on Feb. 1, 2021).

[51] StoreOpenElectronics. *Shield GSM/GPS/GPRS per Raspberry Pi - SIM7600E-H*. https://store.open-electronics.org/Shield-GSM-GPS-GPRS-Raspberry-Pi-SIM7600E-H. 2021. (Visited on Feb. 1, 2021).

[52] Orange Pi. *Orange Pi 4G-IOT 1G Cortex-A53 8GB EMMC Support SIM Card Bluetooth Android 6.0 Single Board*. https://www.aliexpress.com/item/1005001780069819.html?spm=a2g03.12010612.8148356.3.1ca91a9cQOCTGp. 2021. (Visited on Feb. 1, 2021).

[53] Timothy Woo. *GitHub - botletics/SIM7000-LTE-Shield*. https://github.com/botletics/SIM7000-LTE-Shield. Jan. 2021. (Visited on Mar. 15, 2021).

[54] Adafruit. *GitHub - adafruit/Adafruit_FONA*. https://github.com/adafruit/Adafruit_FONA. May 2020. (Visited on Mar. 20, 2021).

[55] DFRobot. *GitHub - DFRobot/TEL0124/Document*. https://github.com/Strictus/DFRobot/tree/master/TEL0124/Document. Mar. 2019. (Visited on Mar. 15, 2021).

[56] leomariga. *GitHub - Simulink and Arduino connection*. https://github.com/leomariga/Simulink-Arduino-Serial#configuring-your-serial. 2019. (Visited on Apr. 10, 2021).

[57] Arduino. *char - Arduino Reference*. https://www.arduino.cc/reference/en/language/variables/data-types/char/. 2021. (Visited on Apr. 3, 2021).

[58] Biswajeeban Mishra. "Performance Evaluation of MQTT Broker Servers". In: July 2018, pp. 599–609. ISBN: 978-3-319-95170-6. DOI: 10.1007/978-3-319-95171-3_47.

[59] Aditya Iyer. *Mosquitto MQTT broker on Raspberry Pi*. https://iot4beginners.com/mosquitto-mqtt-broker-on-raspberry-pi/. June 2020. (Visited on Apr. 17, 2021).

[60] Brian Boucheron. *How to Install and Secure the Mosquitto MQTT Messaging Broker on Ubuntu 16.04*. https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-the-mosquitto-mqtt-messaging-broker-on-ubuntu-16-04. Dec. 2016. (Visited on Apr. 17, 2021).

[61] Janne Kemppainen. *Getting Started With Mosquitto on a Raspberry Pi*. https://pakstech.com/blog/raspberry-pi-mosquitto-getting-started/. Apr. 2019. (Visited on Apr. 17, 2021).

[62] Steve. *Guide to IOT Dashboards and Platforms*. http://www.steves-internet-guide.com/iot-mqtt-dashboards/. Mar. 2021. (Visited on Apr. 28, 2021).

[63] OpenJS Foundation. *Node-RED, Low-code programming for event-driven applications*. https://nodered.org/. 2021. (Visited on Apr. 28, 2021).

[64] Adafruit. *Adafruit IO MQTT API*. https://io.adafruit.com/api/docs/mqtt.html#adafruit-io-mqtt-api. 2020. (Visited on Apr. 30, 2021).

[65] jianhui. *MQTT broker latency measure tool*. https://github.com/hui6075/mqtt-bm-latency. 2020. (Visited on May 11, 2021).

Appendix A

Arduino code

```
#include <Adafruit_FONA.h>
#define SIM7600
#define FONA_PWRKEY 12                                  //Pins for the serial communication and the power.
#define FONA_RST 20
#define FONA_TX 7
#define FONA_RX 8
                                                        //Declaring all the variables and the servers.
#define MQTT_SERVER      "tcp://78.129.106.132"
#define MQTT_SERVERPORT  1883
#define MQTT_USERNAME    "jvdelft"
#define MQTT_KEY         "test"
#define MQTT_TOPICSTATE  "State"
#define MQTT_TOPICSUB    "switch"
#define MQTT_TOPICSUB2   "Led"
char* Subtopics[2] = {MQTT_TOPICSUB, MQTT_TOPICSUB2};
#define FTP_SERVER      "195.144.107.198"
#define FTP_USERNAME    "demo"
#define FTP_PASSWORD    "password"
#define FTP_PORT        21

#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))      //Function to get the number of elements in an array.

int LED1 = 11;
int LED2 = 10;                                          //All the leds for testing
int LED3 = 9;
int LED4 = 6;
int LED5 = 5;
int LED6 = 4;
int LED7 = 3;
int LED8 = 2;

int Leds[8] = {LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8};

char replybuffer[50];                   //Char array to store the received messages
char Switch;
#include <SoftwareSerial.h>
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);                       //Initating the serial communication between the Arduino
SoftwareSerial *fonaSerial = &fonaSS;                                           //and the 4G shield.
Adafruit_FONA_LTE fona = Adafruit_FONA_LTE();

char prevMessage[10] = "          ";            //Char array to store the last message received by Matlab.

/*
typedef union{
```

```
45   float number;
46   uint8_t bytes[4];
47 } FLOATUNION_t;
48
49 FLOATUNION_t myValue1;
50 FLOATUNION_t myValue2;
51 FLOATUNION_t myValue3;                          //Functions to transfer numbers with
       Matlab.
52
53 float getFloat(){
54   int cont = 0;
55   FLOATUNION_t f;
56   while(cont < 4){
57     f.bytes[cont] = Serial.read();
58     cont++;
59   }
60   return f.number;
61 }*/
62
63 void setup() {
64   Serial.begin(57600);
65   pinMode(FONA_PWRKEY, OUTPUT);           //Starting everything, turning on the
       shield.
66   pinMode(13, OUTPUT);
67   for(int i = 0; i<sizeof(Leds); i++){
68     pinMode(Leds[i], OUTPUT);
69   }
70   fona.powerOn(FONA_PWRKEY);
71   fonaSS.begin(115200);
72   fonaSS.println("AT+IPR=57600");           //Testing the communication by switching
       from the
73   delay(300);                               //default baudrate to 57600.
74   fonaSS.begin(57600);
75   fonaSS.println("AT+CPIN=0000");           //Unlocking the sim card with the pin
       code.
76   if(!fona.begin(fonaSS)){
77     Serial.println(F("Could not find FONA"));         //Starting the shield
78     while (1);
79   }
80   Serial.println(F("FONA is OK"));
81   fona.setFunctionality(1);
82   fona.setNetworkSettings(F("internet.proximus.be"));       //Set the APN for
       proximus network
83   while (!netStatus()){
84     Serial.println(F("Failed to connect to cell network, retrying..."));    //
       Connecting to the network.
85     delay(2000);
86     }
87     // read the RSSI
88   uint8_t n = fona.getRSSI();
89   int8_t r;
90
91   Serial.print(F("RSSI = ")); Serial.print(n); Serial.print(": ");          //
       Checking the strength of the network.
92   if (n == 0) r = -115;
93   if (n == 1) r = -111;
94   if (n == 31) r = -52;
95   if ((n >= 2) && (n <= 30)) {
96     r = map(n, 2, 30, -110, -54);
97   }
98   Serial.print(r); Serial.println(F(" dBm"));
99   fonaSS.println(F("AT+CGSOCKCONT=1,\"IP\",\"internet.proximus.be\""));           //
       Enabling the data transfer
100  delay(250);
101  fonaSS.println("AT+NETOPEN");
102  MQTTStart(MQTT_USERNAME,MQTT_SERVER, MQTT_USERNAME, MQTT_KEY);               //
```

```
    Starting of the MQTT connection and connecting to the server
103   delay(1000);
104   while(!MQTTSub()){delay(500);}                                          //
        Subscribing to all needed topics
105
106 }
107
108 void loop() {
109   MQTTReceived();                                  //Checking if we received
        anything from Matlab or from the MQTT server.
110   SerialReceived();
111 }
112
113
114
115 bool netStatus() {
116   int n = fona.getNetworkStatus();
117
118   Serial.print(F("Network status ")); Serial.print(n); Serial.print(F(": "));
            //Getting the strength of the network.
119   if (n == 0) Serial.println(F("Not registered"));
120   if (n == 1) Serial.println(F("Registered (home)"));
121   if (n == 2) Serial.println(F("Not registered (searching)"));
122   if (n == 3) Serial.println(F("Denied"));
123   if (n == 4) Serial.println(F("Unknown"));
124   if (n == 5) Serial.println(F("Registered roaming"));
125
126   if (!(n == 1 || n == 5)) return false;
127   else return true;
128 }
129
130 bool MQTTStart(char* Name, char* server, char* username, char* key){          //
      Starting the MQTT connection with the server.
131   char ConnectionCommand[200];
132   char IdentificationCommand[50];
133   sprintf(ConnectionCommand, "AT+CMQTTCONNECT=0,\"%s\",90,1,\"%s\", \"%s\"", server,
        username, key);
134   sprintf(IdentificationCommand, "AT+CMQTTACCQ=0, \"%s\"", Name);
135   if(!fona.sendCheckReply("AT+CMQTTSTART","+CMQTTSTART: 0",2000)){return false;}
          //Enabling the MQTT connection
136   if(!fona.sendCheckReply(IdentificationCommand,"OK",2000)){return false;}
          //Setting the name the server will see for the shield
137   if(!fona.sendCheckReply(ConnectionCommand,"OK",2000)){return false;}
          //Connecting to the server using the username and the password
138   return true;
139 }
140
141 void MQTTDisconnect(){                                                  //Disconnecting
       from the MQTT server.
142   if(!fona.sendCheckReply("AT+CMQTTDISC","OK",2000)){return false;}
143   if(!fona.sendCheckReply("AT+CMQTTSTOP","OK",2000)){return false;}   //Disabling the
        MQTT connection
144   return;
145 }
146
147 void MQTTSend(char* topic, char* message){              //Send something to a topic
      on the MQTT server.
148   char TopicCommand[20];
149   char MessageLengthCommand[20];
150   sprintf(TopicCommand,"AT+CMQTTTOPIC=0,%i",strlen(topic));                //Building
      the topic and message command lines
151   sprintf(MessageLengthCommand,"AT+CMQTTPAYLOAD=0,%i",strlen(message));
152   fona.sendCheckReply(TopicCommand,"OK",10);
153   fonaSS.write(topic);                                                     //Giving
      the topic
154   delay(4);
```

```
155   fona.sendCheckReply(MessageLengthCommand,"OK",10);
156   fonaSS.write(message);                                                    //Giving
        the message
157   delay(4);
158   fona.sendCheckReply("AT+CMQTTPUB=0,1,60","OK",10);                        //
        Publishing
159   return;
160 }
161
162 bool MQTTSub(void){            //Subscribe to a specific topic.
163   char testMessage[12] = "Hello World!";
164   int allWellSubs = 0;
165   for(int i = 0; i < ARRAY_SIZE(Subtopics); i++){
166     char* subtopic = Subtopics[i];
167     char SubTopicCommand[25];
168     char TopicCommand[25];
169     bool isItWellSub = false;
170     sprintf(SubTopicCommand, "AT+CMQTTSUB=0,%i,1,1",strlen(subtopic));
171     fona.sendCheckReply(SubTopicCommand,"OK",20);                           //Sending
      the topic to subscribe to
172     fonaSS.write(subtopic);
173     fona.sendCheckReply("AT+CMQTTSUB=0","OK",10);                           //
      Subscribing to the topic
174     delay(25);
175     MQTTSend(subtopic, "Hello World!");                                     //Sending a
       test message
176     delay(10);
177     for(int i = 0; i < 10; i++){
      //Verifying that we received what we sent to the server
178       MQTTReceived();
179       Serial.println(replybuffer);
      //So checking if we are well subscribed.
180       if(replybuffer[0] == testMessage[0] && replybuffer[8] == testMessage[8]){
181         isItWellSub = true;
182         break;
183       }
184     }
185     if(isItWellSub){
186       allWellSubs++;                    //Checking if all subscriptions were
      successful
187     }
188   }
189   if(allWellSubs == ARRAY_SIZE(Subtopics)){
190     return true;
191   }
192   return false;
193 }
194
195
196 void MQTTReceived(){                              //Checking if we received anything
      and if so processing the info depending on the topic and the message.
197   String s = fonaSS.readStringUntil('\n');
198   int newMessage = 0;
199   const char startReceivedMessage[13] = "+CMQTTRXSTART";      //Starting transmission
       message
200   char topic[5];
201   for (int k = 0;k<13;k++){
202     if((char) s[k] == startReceivedMessage[k]){         //Checking if we received the
       starting message
203       newMessage++;
204     }
205   }
206   if (newMessage > 12){
207     String line;
208     for (int i = 0; i < 4; i++){                //Extracting the topic and the
      message from the full msg received.
```

```
209       line = fonaSS.readStringUntil('\n');
210       if(i == 1){
211         line.toCharArray(topic, sizeof(line));
212       }
213       if(i == 3){
214         s = line;
215       }
216     }
217     if((s.length()-1) == 1){
218       Switch = (char) s[0];
219     }
220     else{
221       for(int k = 0; k<sizeof(replybuffer); k++){       //Emptying the replybuffer
    before storing the new message.
222         replybuffer[k] = NULL;
223       }
224       s.toCharArray(replybuffer, s.length());           //Casting the new message in
    the array
225     }
226     switch (topic[0]){
227       case 's':
228         if(Switch == 48 || Switch == 49){               //If the topic is starting
    with 's' (switch), turning on or off the built-in led
229           digitalWrite(13,(Switch-48));
230           Switch = '\n';
231         }
232       break;
233
234       case 'L':
235         if(Switch >47){
236           for(int i = 0; i < Switch-47; i++){         //Turning on the specified
    number of leds
237             digitalWrite(Leds[8-i], HIGH);
238           }
239           Switch = '\n';
240         }
241         else if(replybuffer[0] == 'R'){               //Turning on the red led
242           digitalWrite(LED1, HIGH);
243         }
244         else if(replybuffer[0] == 'B'){               //Turning on the blue led
245           digitalWrite(LED2, HIGH);
246         }
247         else if(replybuffer[0] == 'G'){               //Turning on the green led
248           digitalWrite(LED3, HIGH);
249           }
250         else{
251             for(int i = 0; i < ARRAY_SIZE(Leds); i++){      //Turning off all leds
252               digitalWrite(Leds[i], LOW);
253             }
254           }
255       break;
256       default:
257         Serial.println(topic);
258         Serial.println("Invalid topic");       //If the topic is not right
259       break;
260     }
261   }
262 }
263
264 void SerialReceived(){                 //Checking if we received the specific header
    from Matlab and if so processing the information.
265   while(Serial.available()){
266     char header[3] = "";
267     if ((header[0] = Serial.read()) == 'n'){
268       for(int i = 1; i < sizeof(header); i++){
269         header[i] = Serial.read();
```

```
      }
    }
    if(header[0] == 'n' && header[1] == 'e' && header[2] == 'w'){
        char msgReceived[10] = "          ";
        int index = 0;
        char c;
        while(Serial.available() && index<10 && (c = Serial.read()) != '\n'){
      //Putting the new message in a char array
          if(c != 0){
            msgReceived[index] = c;
            index++;
          }
        }
        if(index != 0){
          for(int i = index; i < strlen(msgReceived); i++){                //Emptying
    the characters after the message
            msgReceived[i] = NULL;
          }
          int msgLength = strlen(msgReceived);
          int prevMsgLength = strlen(prevMessage);
          for(int i = 0; i < max(prevMsgLength,msgLength); i++){           //Putting
    the new message in the previous one
            if(prevMsgLength<=msgLength){
              prevMessage[i] = msgReceived[i];
            }
            else{
              if(i<msgLength){
                prevMessage[i] = msgReceived[i];
              }
              else{
                prevMessage[i] = ' ';                                      //Adding
    space at the end so that Matlab can receive it (10 char message)
              }
            }
          }
          MQTTSend(MQTT_TOPICSTATE,msgReceived);                       //Sending the
    received message to the server
          }
        Serial.write("S");
        Serial.write(prevMessage);                              //Sending the received
    message to Matlab
        Serial.write('\n');
      }
  }
}


bool FTPConnect(char* server,int port, char* username, char* password){           //
    Connect to the FTP server.
  char FTPConnectionCommand[150];
  sprintf(FTPConnectionCommand, "AT+CFTPSLOGIN=\"%s\",%i,\"%s\", \"%s\",0", server,
    port, username, password);
  if(!fona.sendCheckReply("AT+CFTPSSTART","OK",2000)){return false;}
  delay(200);
  if(!fona.sendCheckReply(FTPConnectionCommand,"OK",2000)){return false;}
  delay(2500);
  return true;
}

bool FTPDisconnect(){                                                            //
    Disconnecting from the FTP server.
  if(!fona.sendCheckReply("AT+CFTPSLOGOUT","OK",2000)){return false;}
  if(!fona.sendCheckReply("AT+CFTPSSTOP","OK",2000)){return false;}
  return true;
}

```

```
327 bool FTPGet(char* dir, char* fileName){                          //Get a file on the
        FTP server.
328   char FTPGetCommand[50];
329   sprintf(FTPGetCommand,"AT+CFTPSGET=\"%s%s\"", dir, fileName);
330   fona.sendCheckReply(FTPGetCommand,"OK",1000);
331   return true;
332 }
```