Implementing NB-IoT: Communication with a Load Cell

Johannes Almroth

27th January 2020

Acknowledgements

Many thanks to Vetek, who funded the purchase of all the hardware equipment used in the project. I am grateful to the Communication Research Lab at Uppsala university for providing me with space and tools to conduct my work.

I would like to thank, especially, Laura Feeney PhD, for your outstanding patience, words of encouragement as well as your time spent in guiding me. In addition, I would like to thank professor Per Gunningberg.

Heartfelt thanks to my partner, Sabina Smedsrud, for always cheering me up and supporting me throughout this journey.

Abstract

IoT technology has been proclaimed as a new technological prowess that will change our economy as well as our cities and way of living. Despite these bold statements, IoT is far from being easily implemented by companies not directly working with any of the enabling technologies, such as telecom. Narrowband-IoT (NB-IoT), a new radio protocol focusing on wide area coverage and low power consumption, is being heralded by the 3GPP as one of the key technologies necessary to push society into the age of IoT. NB-IoT networks are still extremely new in a lot of countries, and while the SIM-cards necessary to use these networks can be readily purchased from telecom companies, the lack of implemented projects might scare the everyday layman looking to implementing IoT within his/her business. The purpose of this paper is to provide an example for how an IoT device can be implemented in practicality, specifically with a scale. A micro-controller is hooked up to a load cell, from which the data produced is sent to the net via a cloud platform.

Contents

1 Introduction **1**

2 Background **3**
2.1 General Background 3
2.2 NB-IoT vs. LTE-M 3
2.3 NB-IoT Market in Sweden 4

3 Requirements **5**
3.1 Data Polling Rate 5
3.2 Sensor Failure 6
3.3 Sensor Disconnect 7

4 Design & Implementation **9**
4.1 Hardware Implementation 9
4.1.1 Scale 9
4.1.2 ADC 10
4.1.3 Microcontroller 10
4.1.4 Power Source 10
4.1.5 Wiring 11
4.1.6 Failures 11
4.2 Software Implementation 11
4.2.1 reader.py 12
4.2.2 error_tracker.py 12
4.2.3 Data Transmissions 13

5 Results **15**
5.1 Discussion 15
5.1.1 Limitations 15

6 Conclusions **17**
6.1 Future Work 17

List of Figures

a Decreasing values 7
b Graph with spikes 7
a Sensor recovers after some invalid reads 7
b Sensor doesn't recover after invalid reads 7
c Sensor produces too much invalid data 7
a The sensor disconnects 7

4.1 The wiring schematic for the load cell 9
4.2 The front and back of the HX711 10
4.3 The load cell wired to the ADC 11
a Data transmitted via Wifi 13
b 13

Chapter 1

Introduction

IoT (Internet of Things) is a broad, diverse and growing field within the IT sector. Many organizations predict that it will come to impact large areas of our daily life, and many telecom companies are experimenting with different kinds of real world applications that can benefit from this change. The basic idea is the same for everything, which is to take a device and connect it to the internet. Examples range from simple toasters to complex self-driving cars.[11] The focus is to enable communication between devices without the need for a human middleman, thus optimizing whatever application is being implemented. A simple example is a building equipped with multiple IoT-enabled thermostats, which are controlled by a central heating system. Given effective software, heating can be regulated in an energy efficient manner while still keeping visitors adequately warm throughout the day. Another example might be a parking meter, which can forward the availability of its parking spot to some central system which in turn forwards the closest available spot to an end-user. The potential applications are numerous, but factors such as energy consumption and security have proven to be considerable roadblocks that pose huge challenges to most IoT projects.

Vetek is a Swedish scale supplier located in Väddö, situated approx. 100 kilometers north of Stockholm. Vetek constructs their own scales and weighing systems, as well as reselling products from other manufacturers.[19]

Vetek aims to improve their services, and as such are interested in the possible use cases of IoT technology, and ultimately see how that can be applied to their own products. With something as simple as an IoT-enabled scale, they can offer customers products that can be placed in remote areas without the need for constant checkups, enabling long term monitoring and easier analyze of data. An example might be monitoring road salt depots, to enable smarter refill routes during winter time, or a fodder station to map the behaviors of local wildlife.

The largest challenges for this type of device lies in energy consumption and broadcast range. Low energy consumption is needed so that any maintainer doesn't need to make constant check-ups to switch batteries all the time. This poses limitation on the type of scale that can be used, which in extension affects parameters such as scale accuracy and capacity. A wide broadcast range is needed so that the device isn't limited to being close to a base station. This puts restraints on what type of communication protocols can be used, as traditional ones such as Wi-Fi and Bluetooth won't work in the aforementioned examples.

With the advent of IoT, the 3GPP (3:rd Generation Partnership Program, a standardization organization for telecom) has developed new wireless communication pro-

tocols intended to be used by these devices. One of these, the NB-IoT (Narrowband-IoT) protocol is particularly suitable for the challenges mentioned above, as its focus lies (among else) in wide area coverage and long battery life. A microcontroller (a small computer) is needed to handle the data polled from the scale, as well as sending it via some wireless communication protocol. The microcontroller chosen for this project is a FiPy, as it has the capability to handle multiple wireless technologies, one of them being NB-IoT. Some other essential pieces of hardware needed is some form of power supply, as well as an ADC (Analog-to-Digital Converter) that connects the microcontroller and the scale.

In this paper, an attempt is made to implement and NB-IoT enabled load cell (the term load cell is interchangeable with scale for all intents and purposes). Due to hardware difficulties, a functioning connection between the load cell and the microcontroller could not be established while transmitting data, so some dummy data was transmitted later on. Some behavioral restrictions are placed on the code running the microcontroller, to closer resemble a real-world application. Examples of these behaviors are handling of erroneous data and disconnection of the scale.

In the following chapter, some general background information regarding the IoT technology and industry as a whole will be presented, as well as the predictions surrounding it. The purpose of NB-IoT and its market situation in Sweden will also be discussed. In order to emulate some behavior needed in the context of a real-world application, a few requirements have been placed on the device in the way that it handles the polling and transmission of data. The desired behavior is of an all-purpose IoT scale, with the requirements being set by Vetek. These requirements will be presented in the Requirements chapter. The hardware and software implementation of the device will be presented in the Methodology chapter. The outcome of the implementation will be presented in the Results & Discussion chapter, and conclusions as well as a discussion of possible future work are brought up in the final chapter.

Chapter 2

Background

2.1 General Background

Internet of Things (IoT) has been lauded as a world-changing technology that will significantly affect our economy as well as our way of living. In a report by the GSMA, the total number of IoT devices is estimated to triple by 2025, bringing the total number of devices to $25.2 *billion*. Meanwhile, the global IoT revenue will fourfold from 2018, increasing it to $4.4 *billion*.[6] While there undeniably is a lot of excitement and potential economic impact associated with IoT, currently, many consumers just associate the term with connecting a common toaster or coffee machine to a Wi-Fi network. While this technically fits the definition for an IoT device[11], the significant use cases will probably be implemented with different sensors, such as scales, thermometers, etc. that will further improve automatization & optimization processes. As an example, the key categories within the predicted growth are smart homes (*e.g.,* security devices) and smart buildings (*e.g.,* energy consumption sensors).[6] For the predicted growth to happen, businesses need to take a chance and work on projects that implements different IoT technologies, and to enable this, the GSMA has developed some LPWAN (Low-Power Wide-Area Network) protocols that focus on different key aspects that make IoT possible. Some of these aspect include long battery life, high connection density, indoor coverage and geo-tracking capabilities. Aside from security, one of the biggest challenges regarding IoT devices relate to limitations arising from energy infrastructure. As mentioned earlier, one of the core issues NB-IoT aims to achieve is to be a low-power technology, thus decreasing the maintenance needed for battery-powered devices. A claim often paraded with NB-IoT is that it enables a battery-time of up to 10-years[5], though it's worth mentioning that over such a period of time the underlying IoT technology (in the form of microcontrollers/sensors) will probably require more frequent maintenance than the batteries themselves.

2.2 NB-IoT vs. LTE-M

The two most prominent of the protocols developed by the 3GPP are NB-IoT and LTE-M. LTE-M has the most functionality, including voice capabilities and device positioning. Thanks to its wider bandwidth frequency it also has lower latency and boasts a data rate up to 1 Mbps.[18] In return, device complexity and costs are higher compared to NB-IoT. The focus of NB-IoT was to enable indoor coverage, low cost development, long battery life and high connection density, which makes the

technology ideally suited for low data rate applications in extremely challenging radio conditions.

2.3 NB-IoT Market in Sweden

According to Swedish telecom company Telia, they were the first to introduce the NB-IoT technology in Sweden, as well as the Nordic countries overall.[16] They further claim that their network will be in range for over 99.9% of Sweden's population, as well as provide a speed of 200 kb/s in more than 95% of the country.[15] The grand opening of the network was on the 24th of May 2019, and pilot projects were conducted as early as a year before this in multiple locations across the country. Telia currently offers a starter kit for any actor interested in the technology, with a trial period of 6 months that includes access to Telia's IoT portal and APIs as well as 5 SIM cards, each with a 30MB data cap per month. Telia doesn't seem have many strong competitors when it comes to the Swedish IoT market, though Tele2 have partnered up with Nokia to offer similar services, and according to a press release from 2018, they have rolled out both LTE-M and NB-IoT across their networks.[13] Telenor has launched a IoT network in Norway with NB-IoT functionality in 2018[14], and according to an exchange with their customer support, followed suite in Sweden in the beginning of October. The fact that Telia already has partnered up with a multitude of cities and companies give the indication that they have a head start in the market.

Chapter 3

Requirements

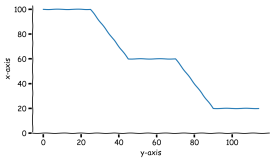
In this section we will discuss some the software requirements imposed on the IoT-enabled scale device that will be implemented, and what behavior we wish to see from the device in the case of different scenarios. These areas were chosen because of their relative simple nature, as well as their relevance to other similar devices. The purpose of these requirements is to emulate the constraints placed on devices in the real world, specifically an all-purpose IoT-scale that Vetek could use in a pilot project for evaluation and future iterations.

There are no specific requirements set for the hardware implementation given the initial Any additional requirements in terms of energy consumption, sizing, etc. would be too vast for the scope of this paper.

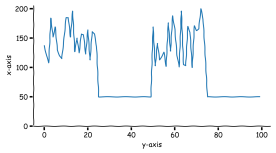
3.1 Data Polling Rate

In most IoT devices the relevant data is provided by some form of sensor, whether it be a scale, thermometer or something entirely different.[11] The type of sensor being used has a huge impact on the IoT device, especially when considering that they have to be powered by the same energy source. The simplest way of deciding when to poll data from a sensor is to let it do so at a fixed and constant rate, often enough to be relevant, and seldom enough as to not waste precious energy. However, if within the context of the application we can conclude that no data needs to be polled (for a while), then subsequently no data will need to be sent, and thus we save energy on both ends of the system. For some applications there might even be longer periods of downtime where it's not relevant to conduct monitoring on the given sensor, *e.g.*,during nighttime, closing hours, etc. Another interesting angle is modifying the polling rate depending on the data itself. A simple example of a behavior would be to have a slower polling rate at stable values, and increase it when experiencing large enough changes. Given the conditions of an IoT device powered by batteries, it's not unreasonable to assume that readings might not always be accurate at times. Depending on the sensor, spikes and drops of false values might occur, and not taking these scenarios into account would be prudent.

In this paper, we want the device to change its polling rate depending on the data values, such that the rate increases at periods of activity and change, and lower the rate when the data stabilizes around a given value. The device should also try to take false data values into account.



(a) Decreasing values



(b) Graph with spikes

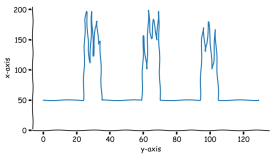
In figure a we see the sensor outputting stable data around a fixed value, to then experiencing a decrease, two times. We want the readings to increase during the data changes, and decrease during the stable plateaus.

In figure b we see the sensor outputting some data values with sudden spikes in them. For the sake of simplicity, we will assume that our scale has the maximum capacity of 100 kg, and thus can easily discern that any value above this is a false reading. In other contexts, false readings might be harder to detect and handle.

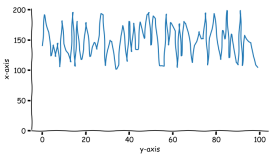
3.2 Sensor Failure

In this paper, we define sensor failure as a sensor giving too many unreliable or false data values to be considered functional. The goal of identifying such a state in an IoT device is to prevent unstable data from being interpreted as valid, which in turn can save the end user from unwanted consequences. Depending on the longevity and purpose of the device, the threshold of when to declare a sensor as failing may differ, especially as this state can be quite fluid. A functional sensor means different things for different devices and applications. A simple way might be to conclude that if x% of data is considered invalid during the last 24hrs, an alarm should be raised to the device administrator. Complications arise when failures need to be reported quickly, or estimated more thoroughly. It's also possible that the sensor can be temporarily unreliable due to external circumstances, and given enough time, these circumstances might pass. On one extreme you can have a device that reports failures too frequently and bogs down whatever dashboard is handling its status report. On the other, you can have a device taking too long to determine a sensor failure so that false data is believed to be valid in the meantime.

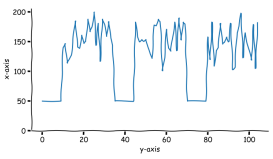
The requirements set on our device state that it has to have some form of self-regulation of when to send these error signals, allowing the sensor some time to recover. If the sensor doesn't recover within a given timeframe, or outputs too much erroneous data, it will raise an error. The reason for still raising an error even though the device recovered is that the sensor might be in need of maintenance if the % of invalid data is too high. This all depends on the filtering of invalid data, as well as at what data threshold the device loses effectivity.



(a) Sensor recovers after some invalid reads



(b) Sensor doesn't recover after invalid reads



(c) Sensor produces too much invalid data

In figure a the sensor outputs some invalid data, yet it recovers. No errors should

be raised.

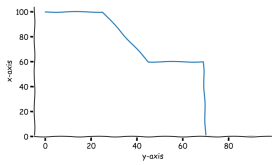
In figure b the sensor doesn't recover from reading invalid data, and an error should be raised.

In figure c the sensor does recover, but still outputs enough invalid data that an error should be raised.

3.3 Sensor Disconnect

We define a sensor disconnect as when no credible data is being produced at all. If the sensor doesn't recover, immediate maintenance is needed for any continued functionality. In this device we know that a sensor disconnect results in the value 0.0 being returned at every poll by the microcontroller to the sensor. While this might be a valid read in some scenarios, in a real world application we would probably never get such a stable value at exactly 0.0. Rather, it would fuzz around at maybe between 1.5 and 0.0 (for example). Disregarding this, we can also know that sensor has disconnect if the data values drop to 0.0 at a too quick pace to be a real-world measurement.

The requirements on this device states that it should raise a disconnect error if the sensor doesn't recover within a given timeframe.



(a) The sensor disconnects

In figure a the data values drop from 60 to 0 in the span of one data point, which would indicate a sensor disconnect.

Chapter 4

Design & Implementation

The work in this paper is divided up into two major areas; the hardware and the software implementations. The aim of this section is to give an overview of the implementation and the design decisions made for each area.

4.1 Hardware Implementation

A similar paper conducted at KTH earlier this year served as the main baseline for how the hardware was to be setup.[8] The main components are the microcontroller, the scale as well as the ADC (Analog-to-Digital Converter). Apart from this, an adequate power source is needed to provide energy to all components.

4.1.1 Scale

The scale used for this paper is a Tedea Huntleigh - Model 1022. It's a small and simple model, and the specific device used in this paper had a maximum capacity of 50 kg.[4] In figure 4.1 we can see the labels of the four wires needed to hook up the load cell. The *Input+* and *Input-* signify the voltage input and ground. [17] *Output+* and *Output-* will output a positive respectively a negative charge of about 1.5 voltage . During weighing, the internal resistance in the load cell will change ever so slightly, and the two outputs will have a small difference in the millivoltage range. This difference represents the weight measurement, and can be translated to a corresponding kg/lb value. In general, the more voltage the scale is supplied with, the greater this millivoltage range can be, which (in theory) means larger accuracy when weighing. Bear in mind that no two load cells are the same, and need to be calibrated to output the correct kg/lb value. Even though no calibrations were made to the load cell during this paper, an increase in the voltage provided to the load cell did correlate to an increased stability in the values produced.

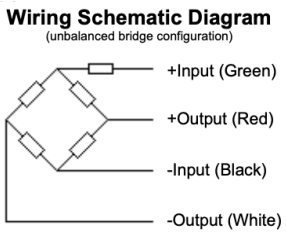


Figure 4.1: The wiring schematic for the load cell

4.1.2 ADC

To convert the millivoltage output from the load cell into a digital signal, an ADC is needed. The device used in this paper is an HX711, and apart from being a converter, it also serves as an amplifier for the load cell signal. We can see the front of the piece in figure 4.2, and on the left side are the pinout where all the wires from the load cell should be connected. It's worth bearing in mind that the color coding of the wiring is not the same for all load cells, and the backside should be checked so that the connections follow the correct wiring schematic.

It outputs data via two of its pins, the DAT and the CLK. The CLK pin will output 0 if it's ready to send data, and 1 if it is not ready. When it is ready, the DAT pin will send a series of 0s and 1s that can be converted from binary to a decimal value, which will then represent the output of the load scale.[12] Multiple code libraries have been written to handle this for the user, and which only require a specification of which pins are being used by CLK and DAT respectively. For this paper, a library written for micropython was used.[3]

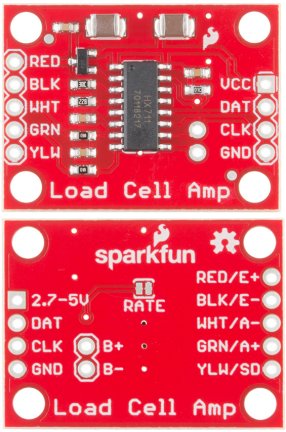


Figure 4.2: The front and back of the HX711

4.1.3 Microcontroller

The microcontroller used for this paper was the FiPy development board from PyCom. It boasts a wide range of capabilities when it comes to communication protocols, NB-IoT being one of the five available.[9] With the supplied expansion board, connections via pinout is possible. It runs on micropython, which is an implementation of Python 3 optimized to run on microcontrollers.[2]

4.1.4 Power Source

In the early stages of the project, the hardware was powered via USB cable from a computer. Since the USB was of type micro, the voltage output was 5V. The benefit of this setup was a simple wiring schema where each component powered the next in line. The drawback was that the FiPy could only supply the ADC with 3V, which in turn affected the ADC's ability to read data from the load cell. During testing, the output rate of the raw data would be infrequent and erratic, sometimes taking several seconds to produce a single value. The values themselves did not correspond to increases and decreases in force being applied to the load cell, and would seemingly

spike and crash at random. These problems were largely in part due to insufficient voltage being supplied to the ADC and load cell as later setups would reveal.

To remedy this, an approach using two different power sources was tried, where the ADC was powered by a wall outlet at a higher voltage, while the FiPy kept the USB. This resulted in electrical interference throughout the system, because of two different grounds being present in the circuit. The output rate of the data values had improved to a bit more stable rhythm than before, and the raw data values were a bit more responsive to the force applied to the load cell.

The best setup that was tested involved an Otii battery toolbox, which is an advanced piece of hardware used to profile and emulate batteries.[1] With this piece of equipment, a common ground was provided to all components, as well as adequate voltage. It was capable of supplying 5V to both the FiPy and the ADC at the same time, which in turn enabled stable output of data values. When force was applied to the load cell, the data values responded accordingly with an increase or decrease in value. Due to time constraints and hardware availability, the setup could not be used for real network transmissions of the load cell data.

All data values produced by the load cell and ADC during these tests were raw values, as a calibration for kg/lb values for our intents and purposes would not bring any significant enhancements to the project.

4.1.5 Wiring

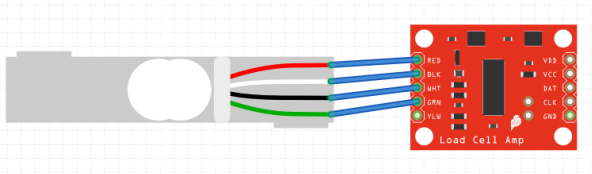


Figure 4.3: The load cell wired to the ADC

4.1.6 Failures

During the first half of the project a faulty load cell was being used, which resulted in major delays of the hardware implementation. During the later half of the project no adequate solution to the problems with the power source could be devised in time for the practical testing of the hardware. Despite this, a short-lived setup with the Otii battery toolbox enabled a fairly functional load cell that could output tangible data values.

4.2 Software Implementation

The FiPy code runs on MicroPython, an implementation of Python 3 optimized for microcontrollers. MicroPython only executes two files on its system's root folder, the boot.py and the main.py files. Any remaining code must be placed in the /lib/ folder. The boot.py runs first, and is intended to contain low-level code that is meant to configure the hardware. The main.py file contains the main program loop, and imports auxiliary files from the /lib/ folder. The two files used in the /lib/ folder are reader.py and error_tracker.py. A test file has been written for each module,

to ensure that the software did not regress during development. All python classes and their respective tests can be found in the appendix.

4.2.1 reader.py

The reader.py file contains the Reader class, which is responsible for processing the data polled from the load cell and passing it on to being transmitted. To instantiate a functional reader object, it needs to be passed a poller function as well as a transmitter function. The purpose of having these functions passed to the instance of the class instead of being hardcoded into the class is to follow the separation of concern design principle.[7] The poller function should accept no argument and is expected to return the current data value produced by the load cell when called. The transmitter function in turn should accept the data value to be transmitted.

The main method of the reader instance is the run() method. When called, the run() method performs a cycle consisting of data polling, a check for false values, adjustment of the polling rate as well as a possible transmission.

Error Check

The polled data value is subsequently checked for validity in the form of out-of-bounds values or extreme delta changes. A separate class called Error_tracker monitors the interval and frequency of these occurrences, and the purpose of the class is to raise an exception when the error rate is deemed too high, and some form of remedial action needs to be taken. The internal workings of the Error_tracker will be explained in a separate section.

Adjustment of polling rate

If the value is deemed valid, it is added to a FIFO (First-in, First-out) buffer of the most recent values. The contents of the buffer are then summed into a total delta value, which is used to adjust the polling rate. If the total delta surpasses a pre-defined threshold, the polling rate is increased, whereas if it is lower it might be maintained or decreased. When we say total delta value, we mean the delta values between two points, added to the delta of the next two points, as such:

$$\sum_{n=0}^{10} x_n - x_{n+1}$$

4.2.2 error_tracker.py

The purpose of the Error_tracker class is to keep track of the error occurrence and frequency. The intended usage is to instantiate an instance of the class, and call the error_occurred() method when an invalid read has occurred. This method increases an internal counter, which will then raise an exception if it passes a pre-defined threshold.

The Grace Period

When invalid reads occur due to some temporary circumstance, it might not be beneficial to count all errors within a given timeframe towards raising an exception. We are not interested in sending an alert when short periods of errors occur, instead, it is at the long-lasting periods of polling errors an exception should be raised. To

account for these short bursts, we use a GRACE_PERIOD constant. This constant is used for the time window (in seconds) after the error_occurred () method has been called, during which sequential calls will not count toward the exception threshold.

The Cooldown Period

We don't want errors to rack up towards an exception over a longer period of time, so it would make sense to reset or decrease the internal error counter after a while. To enable this, a cooldown period is started after the grace period, during which any calls to to the error_occurred () method will interrupt the cooldown period, and a new grace period will ensue. However, if the cooldown period ends without any calls being made to the error_occurred method, the internal error counter will decrease by one. This way of self-regulation ensures that only a long-lasting and consistent frequency of errors raise an exception.

4.2.3 Data Transmissions

Data transmission tests were done separately from the load cell using dummy data. Pycom, the parent company behind the microcontroller used in this project also operate a cloud-based device management platform. [10] Via the pybytes platform, configuration of the network settings of the device can be managed via a firmware updater. In figure 4.4a and figure 4.4b we can see data being transmitted via Wifi on a home network. The data is then displayed via graphs on the pybytes platform.

(a) Data transmitted via Wifi

(b)

Due to time limitations in the project, no actual tests of the transmission of the data via NB-IoT was made

Chapter 5

Results

With the help of the Otii battery toolbox, a functioning load cell-to-microcontroller setup was devised, though no end-to-end data transfer via network was made with the real data. Separate tests with the FiPy showed capabilities to send data via WiFi in conjunction with the PyCom platform, PyBytes.The device also fulfilled the software behavior requirements imposed to emulate real world situations and usage, though it's worth repeating that these requirements were based not based on tests and research.

5.1 Discussion

Despite hardships and complications when implementing the hardware, the results suggest that building and programming an NB-IoT enabled device connected to a load cell is possible with fairly simple consumer available hardware devices. It is a reasonable assumption that the hardware hindrances encountered in this paper can be bypassed with adequate planning and experience in assembling electric hardware.

These results show a small starting post of implementing a IoT device using a load cell as its sensor. With the rise of 5G and IoT platform services there exists a real commercial interest to enable more and more actors to innovate and create products for consumers and companies alike. This project can serve as a starting point for what basic needs and behaviors that need to be considered when planning and designing such a device.

5.1.1 Limitations

This project was done on the smallest scale possible, and extensive research in multiple different areas need to be conducted to even get an IoT enabled scale close to being produced for functional use, commercial or private. Furthermore, these results can't account for wether the software implementation was close to emulating the desired behavior of a device from a practical standpoint, since no potential end-users were involved in the requirement specification. Since the testing was only done indoors in a clean and controlled environment, unknown variables present in the real world could very well produce a multitude of challenges that change the way that the device works. Another interesting angle is how different locations would interact with the connection to the cellular network it relies on for communication. The NB-IoT technology makes huge promises, but at the end of the day it is up to the local telecom company to fulfill the underlying conditions that make those claims possible, which would be Telia in our case.

Expanding on this, since the NB-IoT technology is fairly new in a lot of countries, there is bound to be extremely different implementation experiences from region to region depending on the network provider. In fact, NB-IoT devices could be rendered obsolete in entire regions depending on the network provider.

Chapter 6

Conclusions

The res

6.1 Future Work

A somewhat personal opinion regarding prioritizing any future work to be done on this IoT scale is to involve an end-user in some shape or form early on. Ideally, a UX-designer would lead some form of market research study to more accurately specify what needs and requirements this kind of device would need to satisfy. This would be used to guide any further modifications and limitations put on the device. A more proper analyze of the needed hardware components needs to be done. Optimization in the form of hardware space, energy consumption and economical costs are essential if any more than a handful of devices are to be produced. A more UX and design related question would pertain how standardized the software of the device needs to be constructed. Depending on how the use cases and market needs, can one solution fit all, or is there a way to tailor to needs in an effective way?

While the predictions and promises for the IoT industry remain hopeful and grandeur, no new devices and products will appear from thin air and propel IoT into being an integral pillar to our modern society just like that. The technology available today enables smarter products that have the potential to vastly improve our lives so long as we keep testing, trying, failing and improving. This paper was one, albeit small, test of these new technologies and many exciting projects are sure to come.

Bibliography

[1] Qoitech AB. Otii battery toolbox. URL https://www.qoitech.com/products/battery-toolbox.

[2] Damien George. Micropython. URL https://micropython.org/.

[3] David Gerber. hx711-lopy. https://github.com/geda/hx711-lopy, 2019.

[4] Vishay Precision Group. Model 1022, 2017. URL http://www.scalesnet.com/files/users/PDF/TEDEA/1022.pdf.

[5] GSMA. Narrowband – internet of things (nb-iot), 2019. URL https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/.

[6] GSMA Intelligence. The mobile economy. GSMA's Mobile Economy report series https://www.gsma.com/r/mobileeconomy/, 2019.

[7] Philip A. Laplante. *What Every Engineer Should Know About Software Engineering*. CRC Press, Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742, 2007. URL https://books.google.se/books?id=pFHYk0KWAEgC&pg=PA85&dq=%22separation+of+concerns%22&hl=en&sa=X&ei=WQ_aUNn5DYjNiwLS54GADQ&redir_esc=y#v=onepage&q=%22separation%20of%20concerns%22&f=false.

[8] Lisa Mach and Maneejun Kadepisarn. Internetuppkopplade vågar till södersjukhuset. Bachelor's Thesis, June 2019.

[9] PyCom. Fipy, 2019. URL https://docs.pycom.io/datasheets/development/fipy/.

[10] Pycom. Pybytes, 2020. URL https://pycom.io/solutions/software/pybytes/.

[11] Steve Ranger. What is the iot? everything you need to know about the internet of things right now, 2018. URL https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/.

[12] AVIA Semiconductor. 24-bit analog-to-digital converter (adc) for weigh scales. URL https://cdn.sparkfun.com/assets/b/f/5/a/e/hx711F_EN.pdf.

[13] Tele2. Tele2 iot och nokia ger företag större makt med ny modell för globala iot-implementeringar, 2018. URL https://www.tele2.com/sv/Media/pressmeddelanden/2018/tele2-iot-och-nokia-ger-foretag-storre-makt-med-ny-modell-for-globala-iot-implementeringar

[14] Telenor. Telenor launches nationwide iot network over 4g, 2018. URL https://www.telenor.com/media/announcement/telenor-launches-nationwide-iot-network-over-4g.

[15] Telia. Telia först i sverige med rikstäckande nät för sakernas internet - narrowband iot, 2018. URL http://press.telia.se/pressreleases/telia-foerst-i-sverige-med-rikstaeckande-naet-foer-sakernas-internet-na

[16] Telia. Nb-iot, 2018. URL https://www.teliacompany.com/en/about-the-company/internet-of-things/nb-iot/.

[17] Dara Trent. Understanding load cell specifications, 2019. URL https://www.800loadcel.com/white-papers/377.html.

[18] Shanquing Ullerstig, Ali Zaidi, and Christian Kuhlins. Know the difference between nb-iot vs. cat-m1 for your massive iot deployment, 2019. URL https://www.ericsson.com/en/blog/2019/2/difference-between-NB-IoT-CaT-M1.

[19] Vetek. Om vetek, 2019. URL https://www.vetek.se/om-vetek/content.