NTNU

Norwegian University of
Science and Technology

Tracking and positioning of objects using a network of Bluetooth Low Energy devices

Jan Tore Guggedal

Master of Science in Cybernetics and Robotics

Submission date:	June 2018
Supervisor:	Geir Mathisen, ITK
Co-supervisor:	David Edwin, Nordic Semiconductor

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

This master thesis presents a fully functional indoor positioning platform that's been developed and tested and a a proposed positioning algorithm for use on data collected by the platform. The platform consists of Power over Ethernet (PoE) enabled *anchor nodes* that have Bluetooth Low Energy (BLE) radio capability and Ethernet connectivity, a PoE network switch and a computer for data processing. The nodes continuously scan for BLE advertisements from objects that are being tracked, called *tags*. When such BLE advertisements are received, the nodes create scan reports containing RSSI value and metadata for the received packet. The report is sent to the computer via Ethernet for further processing. In the computer, the scan report is parsed and stored to a database. Optionally, the computer can also apply a positioning algorithm to the received data in real-time to position and track objects. Alternatively, it can fetch the data from the database and apply the algorithm at a later stage.

The positioning algorithm in this thesis uses multilateration using a user-defined number of nodes' data to estimate a tag's position. Distances from each node to the tags are estimated using RSSI values and log-distance path loss model with parameters that are found by experimentation and regression analysis. Kalman filter is applied to the inherently noisy RSSI values. After multilateration has estimated a position for the tag, a Kalman filter is applied to the position estimate.

The results show that the position estimates have an average error of 1.44m and a median error of 1.21m. 40% of the position estimates are within 1m of the true position and 85% are within 2.5 meters. Finding a position estimate for a tag is relatively fast, and it's shown that 5 algorithm iterations after the platform first discovered a tag, the position estimate can be only marginally improved by more iterations. One iteration is in this context equal to one BLE advertising interval. After the initial few iterations, the position is stable also for moving objects due to the Kalman filtering.

The thesis concludes that the developed indoor positioning platform is fully functional for tracking and positioning of BLE enabled devices, and suggests areas for improvement. It also finds the positioning algorithm adequate for indoor positioning, although the position estimate accuracy is not on par with the best comparable alternatives.

Sammendrag

Denne masteroppgaven presenterer utviklingen og testingen av en fullt funksjonell plattform for innendørs posisjonering og en posisjoneringsalgoritme for anvendelse på data samlet inn av plattformen. Plattformen består av *ankernoder* med Power over Ethernet (PoE) som har Bluetooth Low Energy-radiofunksjonalitet og Ethernet tilkobling, en nettverkssvitsj med PoE og en datamasking for prosessering. Nodene søker kontinuerlig etter BLE-annonsepakker fra objekter som spores, kalt *tags*. Når en BLE-annonsepakke er mottatt, lager nodene en skannerapport som inneholder den mottatte signalstyrken (RSSI) sammen med annen metadata om pakken. Rapporten sendes via Ethernet til datamaskinen, der den prosesseres og innholdet lagres i en database. Det er også mulig å anvende posisjoneringsalgoritmen på innkommende data i sanntid for å posisjonere og spore objekter. Alternativt kan data hentes fra databasen og algoritmen anvendes i ettertid.

Posisjoneringsalgoritmen i denne oppgaven bruker *multilaterasjon* på et brukerdefinert antall noders data for å estimere en tags posisjon. Distansen mellom hver node og tagen estimeres ved å bruke RSSI-verdier og *log-distance path loss*-modellen med parametere funnet ved eksperimentering og regresjonsanalyse. Kalman-filter brukes på de ustabile RSSI-verdiene. Etter at multilaterasjon har estimert en posisjon for en tag, brukes også et Kalman filter på posisjonsestimatet.

Resultatene viser at posisjonsestimatene har en gjennomsnittlig feil på 1.44m og en median feil på 1.21m. 40% av posisjonsestimatene er innenfor 1m fra den sanne posisjonen og 85% innenfor 2.5m. Å finne et posisjonsestimat er relativt raskt, og det er vist at 5 algoritmeiterasjoner etter at plattformen for første gang oppdagde en tag, er det lite å hente på flere iterasjoner med tanke på å øke presisjonen. En iterasjon er i denne sammenheng lik ett BLE-annonseringsinterval. Etter de første få iterasjonene, er også posisjonsestimatet for et bevegelig objekt stabilt.

Oppgaven konkluderer med at det er utviklet en fullt funksjonell plattform for innendørs posisjonering og sporing, og foreslår områder med potensial for forbedring. Den finner også at posisjoneringsalgoritmen er adekvat for innendørs posisjonering, selv om presisjonen i posisjonsestimatene ikke er på nivå med de beste av eksisterende løsninger.

Problem statement

Indoor positioning and tracking is generally more challenging than outdoor tracking due to the lack of GPS signals and the many obstacles in an indoor environment. This project will use a network of devices with Bluetooth Low Energy radio for tracking and positioning of objects. These Bluetooth capable nodes also have Ethernet capability, time synchronization capability and are powered over Ethernet (PoE).

The nodes in the network can pick up signals from objects with Bluetooth in the area and extract information about the link quality between itself and an object. The time synchronization capability can be used to precisely time the radio signals. This data may be collected from the Bluetooth devices using Ethernet and processed centrally to estimate where a certain object is located and moving using data from multiple nodes. Different algorithms and analysis methods can be used to determine position and also provide tradeoffs between the quantity of data required and the precision of the position/movement information. Problems like point a phone at a light to select a light or automatically determining relative positions of the different nodes in the network can be examined by the algorithms.

Preface

This master thesis, “Tracking and positioning of objects using a network of Bluetooth Low Energy devices”, has been carried out in cooperation with Nordic Semiconductor in Trondheim. The focus of the thesis has been to develop and test an indoor positioning and tracking platform using hardware created by Nordic Semiconductor.

Nordic Semiconductor have provided all necessary equipment and facilities, such as office space, electronics laboratories, PC, software with relevant licenses, all hardware, test site and lots of guidance and help when needed. Help has been received from employees at the electronics lab for soldering and troubleshooting circuit boards, from hardware designer to better understand the underlying fundamentals and how to best utilize the hardware and radio, from software developers to find solutions to difficult problems and bugs, and from people at the coffee machine challenging my thinking around various aspects of the positioning problem. In particular, supervisor at Nordic Semiconductor, David Edwin, has provided continuous help and feedback during the development and testing of the platform. Valuable assistance has also been offered by Wiznet with the Wiznet Ethernet chip that’s been used and for writing a custom bootloader suited to the nRF52840 microcontroller.

All hardware was provided at the start of the project, none has been made during this project. However, as the PCB for the nodes in the project were the first prototype iteration, minor modifications have been made where found necessary to make it function as intended.

My main contributions with this thesis are

- Firmware for all microcontrollers
- Hardware testing, verification and minor modification
- Positioning algorithm development and testing
- Software for data collection from the platform’s nodes
- Deployment of the positioning platform in an indoor environment
- Testing of the platform
- Software for processing of data and analysis of the test results

I would like to thank Nordic Semiconductor for giving me the opportunity to work on this thesis and for providing all needed assistance, and I want to thank my supervisors for constructive and useful feedback throughout the project. Finally, I want to thank my wife, Ingunn, for supporting with me throughout the work on the thesis, even through the dark valleys of endless and frustrating debugging.

Contents

List of Figures

List of Tables

1 | Introduction

1.1 Motivation

Global Positioning System, GPS, is a widely used and popular technology for positioning and tracking of assets. It works well, with good accuracy and is implemented in many every day consumer products such as watches and phones. However, GPS is not a reliable system for indoor use, simply because inside a building, the radio signals from the satellites that GPS depends on, are weak or not detectable at all. A building's exterior is an effective shield for GPS signals, and an indoor environment has many walls, floors and obstacles that makes indoor positioning with GPS unstable and often impossible.

There have been many attempts to overcome the challenges in indoor environments. Many researchers have presented their solutions with promising results, but a review by Microsoft in 2017 concluded that the indoor positioning problem is still not solved [18]. Microsoft organizes an annual competition where teams from all over the world come to compete on having the best indoor positioning system, the results have been improving, and some contestants have shown centimeter-level precision for their indoor positioning systems. However, the systems are so far not considered compact end sufficiently cost effective to set up to call the indoor positioning problem solved. Therefore, the research on indoor positioning systems continues.

In this thesis, development of an indoor positioning platform and a proposed positioning algorithm will be presented. The platform is intended to serve as a versatile test platform for positioning algorithms and signal processing to enhance the precision of arbitrary positioning algorithms. When it's developed, it can serve as a basis for further research and testing into the field of indoor positioning and tracking systems. This thesis will focus on development of the platform, the positioning algorithm and functional testing of the two.

1.2 Project outline and limitations

The thesis aims to answer to the challenges raised in the problem statement given by Nordic Semiconductor, as seen on page v.

The report will present an indoor positioning platform that enables tracking and positioning of assets in an indoor environment. The thesis presents a complete solution using Bluetooth Low Energy radio mode. The concepts will however be applicable for all the other supported radio modes of the platform's utilized System on Chip, nRF52840. The thesis presents how the platform hardware is designed and connected and how the microcontrollers of the hardware is programmed. A positioning algorithm based on well-known principles of indoor positioning and positioning in general, such as multilateration and Kalman-filtering is proposed. It's described how the platform collects data and sends them to a computer for processing and positioning algorithm application. The platform is then deployed to a testing environment and several tests are performed. The proposed positioning algorithm is applied to data collected by the platform both in real-time and by post-processing. The results are analyzed and discussed, and the platform as a whole is evaluated.

The thesis aims to cover the central aspects of the development process, testing, results and evaluation. Unfortunately, there's not room for everything within the scope of the thesis. Therefore, priorities have been made. This report will for instance not go into detail about the bootloader and option of updating the firmware over the network. The reason for this is that the solution is by and large developed by Wiznet, the Ethernet chip manufacturer, and it's only briefly been tested and confirmed to work. However much I would have liked to expand on the solution provided by Wiznet and be able to utilize it fully in the platform, other parts of the platform had to be prioritized.

It should also be noted that priorities had to be made when both testing and analyzing the results from the tests. There were a lot of interesting aspects, algorithm variants and statistical methods that were tempting to try and experiment with, but only a few could fit into the scope of the thesis. Hopefully, the priorities have been reasonable, and it's believed that the tests and results cover the most central characteristics of the platform and algorithm.

1.3 Previous work

In a previous project, TTK4551 at NTNU, concluded fall 2017, parts of the hardware later referred to as *nodes* were used [10]. Most of the design and specification of the hardware was done by Nordic Semiconductor in the fall 2017, with minor contributions by me. The previous project implemented functioning firmware for the nodes, such as network drivers for Ethernet, IP, TCP and UDP. A proof-of-concept multiprotocol gateway was made and functionally tested by creating an application that used Message Queue Telemetry Transport for Sensor Networks (MQTT-SN) for two-way communication between peripheral Bluetooth Low Energy and Thread devices and an Internet-based MQTT client. Parts of the knowledge and code base from [10] concerning the network functionality is implemented and further expanded in this thesis.

1.4 Related work

Indoor positioning is a research field with increasing interest [19]. As outdoor tracking and positioning largely can be viewed as a solved problem with today's GPS and other satellite technology, the same can't be said of indoor positioning systems. There have been many attempts to make indoor positioning systems, and some have deployed their systems into the open market [25, 20]. However, the most used systems have limitations in accuracy, while the most accurate systems are expensive and have labour-intensive installation and setup processes [18].

The earlier mentioned Microsoft competition for indoor positioning systems started in 2014 to see how far research and implementation has come on the area and how it develops. Of the solutions examined since then, the majority do not rely on new infrastructure, but are instead so-called *software-based* solutions that seek to utilize existing infrastructure, such as WiFi signals, TV signals or FM radio signals. On the other hand we have *hardware-based solutions* that require dedicated hardware to either be installed or to be carried by the object being tracked [18].

In September 2017, Dimitrios Lymberopoulos and Jie Liu published a thorough review of the lessons learned over the past four years of Microsoft's indoor localization competition and the current state of the research field [18]. Even though researchers all over the world have worked on the problem for over a decade, proposing hundreds of different solutions, they still consider the indoor positioning problem unsolved in terms of convergence towards a widely accepted technology of choice. Many solutions are promising, but either accuracy, ease of implementation or cost prohibits the solutions from becoming widely used and accepted as the leading indoor positioning technology. For example, LIDAR technology has proven to be the consistently most accurate over the years, but it's not a realistic contender to be widely adopted because of its cost and equipment size. Lymberopoulos and Liu in fact emphasize that to become a widely adopted localization technology, the concept would need to be compatible with existing products such as mobile phones to avoid overhead with regards to external equipment.

This thesis will focus mainly on hardware-based solutions using Bluetooth Low Energy and similar technologies, along with algorithms applicable to those technologies. Hardware-based systems require installation of dedicated hardware to function as intended. The rest of this section will present a selection of the existing solutions for both hardware-based and software-based solutions, starting with the most popular within both categories, fingerprinting.

1.4.1 Fingerprinting

The most widely used technique for indoor positioning and localization systems using Wi-Fi, BLE, other wireless technologies or even magnetic fields, is fingerprinting [18]. Fingerprinting uses either existing infrastructure or installs dedicated hardware. Either way, the procedure is the same.

Fingerprinting is split into two stages: an offline analytic stage and an online stage where actual tracking and positioning happens. The offline stage involves careful examination of the wireless characteristics of the environment. Wireless signal strength is recorded systematically at many positions to form what's called the wireless fingerprint of the area [5, 6]. From the measurements, one can make maps on how the radio signals from the different beacons vary at different locations. An example of such a map for one beacon is shown in fig. 1.1. The maps can be compiled by recording signal strength from multiple sources,

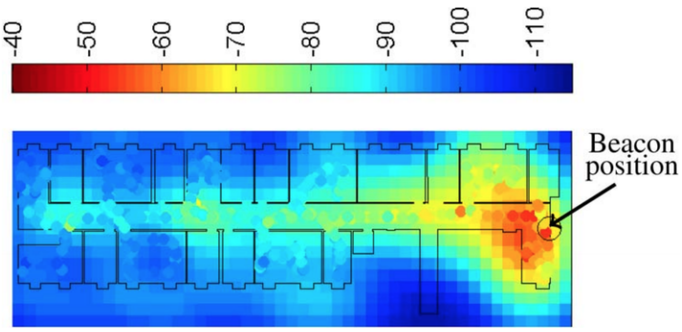


Figure 1.1: A signal strength map for one BLE beacon used for fingerprinting, from [6]. The colors indicate the recorded RSSI for advertisements from the labeled beacon in that particular point. Figure source: [6], figure 5, page 2423.

When the offline training period and examination of the area is done, online tracking and positioning can start. An asset can be placed at a location and record the signal strength values from the beacons at that spot. From the recorded data and previously defined fingerprints it can infer the likelihood of being at one specific location within the environment.

The typical way of doing fingerprinting using Bluetooth Low Energy is to have beacons evenly placed in an area and record the received signal strength indicator (RSSI) from them. All the logic of the positioning system then resides in the asset itself, or alternatively and depending on the use-case the asset device can forward the data to some central device that performs the positioning algorithm.

An interesting point raised by Lymberopoulos and Liu in their evaluation in [18] is that even though many implementations rely on the exact same signals in the same environment at the same time, the accuracy can vary widely, from a mean error of 1.1 meters to over 5.23m. This shows that the specific implementation and processing of signals is very important for the resulting position estimate's accuracy. [18] concludes that today's, or at least September 2017's, best fingerprinting in a real-life environment will be able to achieve a mean error in the range of 3-4 meters over time.

Fingerprinting's major drawback is that it assumes a constant environment. When moving furniture, people are moving around in the area or moving a beacon or access point, the fingerprints are not valid anymore, and the accuracy will deteriorate. The fingerprinting process needs to be repeated frequently to maintain high accuracy leading to a high usage overhead.

1.4.2 Trilateration and multilateration

Trilateration is similar to the technique used by GNSS (Global Navigation Satellite System, using for example GPS, GLONASS and Galileo) devices to calculate their positions. GPS uses carefully synchronized clocks to time signal travelling time and thereby calculate distances to satellites. For indoor systems, one wants to achieve "indoor satellites" using beacons that send wireless signals. The receiving device can then estimate the distance to the transmitting device using any number of techniques: ToF (time of flight), ToA (time of arrival), TDoA (time difference of arrival) or received signal strength indicator, RSSI. Some of the most accurate indoor systems use UWB signals to accurately estimate distances and positions, with mean errors in the low tens of centimeters [37]. BLE based systems typically have clocks with lower resolution leading to insufficient timing accuracy and therefore need to rely on the less accurate received signal strength indicator, RSSI, to estimate distances, yielding a higher mean error [7, 31, 40].

The concept of trilateration is to use three measured distances from three known positions to the object that's being tracked. When the distances and anchor positions are known, one can solve a set of equations to find the unknown position of the object. The theory behind trilateration is presented in greater detail in section 2.2.

Trilateration can be expanded to consider more than three nodes, which is in fact what GPS devices do. This is called *multilateration* as it uses multiple nodes, but this extension of trilateration is often referred to as trilateration as well in the literature. It should be noted that there is some discussion as to whether multilateration refers to a different technological solution than multilateration, but the consensus seems to be moving in the direction of trilateration being a sub-group of multilateration [42] In this report, the term multilateration is used. Multilateration is also explained in more detail in section 2.2.

A general trend found when investigating many similar solutions in [18], is that RSSI is mostly used for fingerprinting, while timing of signals is increasingly popular for trilateration and multilateration. However, the results between the different techniques is not directly comparable, as the complexity for timing based signals and fingerprinting is superior. The best systems in the category of using BLE or RFID and RSSI values for trilateration and multilateration, have a mean error of their estimates in the range of 0.5m to 1.0m [31]. It should be noted that the characteristics of the BLE devices used in [31] seem different to what one would expect, probably due to the environment in which tests were conducted. The RSSI values decline more than usual as distance increases. It makes distance estimation less error prone, but for some reason it's not in line with the widely accepted log-distance path loss model for RSSI and distance relationship (see also section 2.1.2.1). Another project ([2]) had with more recognizable BLE characteristics, Kalman filtering and trilateration. The authors achieved a system with 90% of the position estimates within 1.8 meter in one scenario, and 90% within 4.6 meters in another scenario. For an unspecific indoor positioning system, the meta study in [18] shows that the mean error of all positioning systems is typically in the range of 1.5 meters to 5 meters.

1.4.3 Sensor combinations

Many solutions use multiple sensor systems to increase accuracy. Commonly used sensors are gyroscopes, accelerometers and pedometers, but also other sensors and technologies have been tested in new solutions. The best performing system in Microsoft's indoor localization competition 2018 in the 3D category where deployment of infrastructure is allowed, used Ultra Wide Band (UWB) radio signals fused with augmented reality techniques to achieve a mean 3D positioning error of 0.27 meters of a moving asset [23]. Their solution uses augmented reality features of mobile phone APIs together with an external device providing UWB capability. A future goal for the researchers behind the solution is to instead use Bluetooth radio or other "mobile phone native" technology.

Another sensor combination is multilateration and dead-reckoning fused with Kalman filter in [33]. Dead-reckoning is the process of estimating an object's position based on a previous known position by taking the speed, heading, acceleration and other factors into account. Commercially available solutions, such as Coursa from InvenSense use one BLE beacon as an anchor point for dead-reckoning [25]. When the beacon is registered by a phone, an app running in the background knows that the phone is at one specific point, usually the entrance of a shop. From there, a fusion of gyros, accelerometers and magnetometers are used to estimate the path of the person in the shop. This way, the customer can get relevant offers at certain locations, and the shop owner can track customer behavior.

The combination of dead-reckoning and fingerprinting has also been used, allowing for less fingerprinting measurements to be taken in the offline period [21]. In [38], the combination of fuzzy models and particle swarm optimization was used to obtain promising results, with 95% of the estimated positions being within 1 meter of the real position.

A promising solution for relative indoor positioning, was presented in [45]. The solution, intended for first-responders, uses sensors attached to a person's legs to measure relative movements to estimate a 3D movement path. In Microsoft's indoor localization competition held in April 2018, the solution achieved the best mean positioning error of 2.4 meters on a moving object in the category for 2D positioning without pre-installed infrastructure [1]. The results have however been revised and found to actually be 1.3 meters mean error after the results were first published, which is a very impressive result for a solution without external infrastructure.

1.5 Contribution

My contribution in this thesis is the following:

- Writing firmware for all microcontrollers
- Hardware testing, verification and minor modification
- Configuring Power over Ethernet network switch
- Deployment of the positioning platform in an indoor environment
- Testing of the platform and data collection
- Positioning algorithm development and testing
- Application of positioning algorithm to collected data

- Analysis of the test results
- Evaluation of the platform and positioning algorithm

1.6 Structure of the thesis

- **Chapter 2**: Background theory for the platform implementation. Includes radio fundamentals, trilateration and multilateration, communication protocols and Kalman filter.
- **Chapter 3**: Indoor positioning platform design. Presents the platform conceptual design, hardware, firmware for microcontrollers and processing of collected data.
- **Chapter 4**: Positioning algorithm. The algorithm in the thesis uses RSSI to estimate distances and use these estimates for multilateration. This chapter presents the positioning algorithm.
- **Chapter 5**: Test descriptions. The platform has been deployed in an indoor environment for testing of both its functionality and the positioning algorithm performance. The tests are described in this chapter.
- **Chapter 6**: Results from the conducted tests of the positioning algorithm.
- **Chapter 7**: Discussion of the platform implementation and test results.
- **Chapter 8**: Concluding remarks.
- **Chapter 9**: Future work. A presentation of alternative paths for further development and testing of the platform.

2 | Background

This chapter lays out the theoretical background for central concepts in the indoor positioning platform. First, some aspects of radio technology are presented, followed by the communication protocols used by the platform. Finally, the positioning methods trilateration and multilateration are presented.

2.1 Radio fundamentals

Radio signals are electromagnetic waves that propagate through a medium. The waves have a set of interesting properties with regards to indoor positioning, such as a specific amplitude, frequency and signal strength. Radio waves of a given frequency and amplitude will in ideal conditions propagate unchanged with the speed of light. In real situations, this will not be true, although the change in propagation speed from vacuum is very small for many cases. In an indoor environment, the signal will most likely not travel unchanged from a transmitter to a receiver. Its power density will gradually be reduced when travelling through air or any other material. This is called *path loss* and is caused by factors such as [24]:

- **Multipath effects**: When a radio wave travels from a transmitter to a receiver, the signal may take more than one path. This happens when the wave hits objects and is reflected or diffracted. These "wave copies" will arrive at the receiver at slightly different times, interfering and combining to a received wave that may differ from the one that was transmitted. Depending on the bandwidth and frequency, small changes in positioning of either antenna may result in very different radio waves at the receiver. In indoor environments these effects are prominent, as there are more obstacles to bounce off of, and often there is no direct line-of-sight between the antennas.

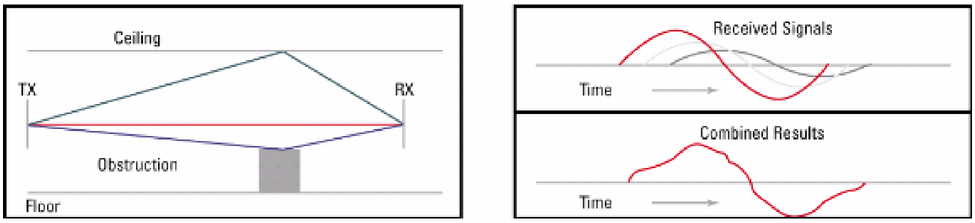


Figure 2.1:]
Multipath radio propagation. The received signal is distorted due to multipath effects compared to the sine wave that was sent from the source. Figure source: [3].

- **Propagation losses**: A radio wave is sent from an antenna, and will gradually expand as it travels in a manner dependant on the antenna. Often the radio front takes a spherical form, and the power density will hence decrease as the front expands when travelling.

- **Absorption losses**: when radio waves encounter an object, the waves may be absorbed by the material in the object, for example when passing through a wall or a human body.

- **Interference**: Different radio waves can interfere with each other, distorting the signals. This is a prominent factor for example in environments with many devices communicating on the same frequency, such as the open ISM 2.4 GHz band that is used by Wi-Fi, Bluetooth, Zigbee and many other wireless protocols. This type of interference can also by of malicious nature, with an entity specifically jamming a frequency band, leaving it practically useless for communication.

2.1.1 Received Signal Strength Indication, RSSI

To indicate how strong a radio signal is, a commonly used metric is Received Strength Signal Indicator, RSSI. RSSI has the unit of dB or dB per milliwatt, dBm. The calculation of RSSI d is not standardized across all radio standards and may not even be specific for a given radio protocol. Different manufacturers can also choose to implement and present RSSI differently. This may result in varying and inconsistent RSSI measurements across different radio communication protocols and equipment. However, most protocols follow the same pattern, where RSSI values are in the range of about +20 dBm to -100 dBm, which is also what Bluetooth does [9].

The RSSI values are usually negative, and the less negative it is, the stronger is the signal. So an RSSI of -40 dBm indicates a stronger signal than an RSSI of -60 dBm. The range of RSSI values one can expect to see from a device depends on the factors mentioned in section 2.1 and also the initial power the signal is sent with from the transmitter.

2.1.2 Using RSSI to estimate distance

Using RSSI measurements, it's possible to estimate the distance between the transceiver and the receiver device. There are several approaches to estimate the distance from RSSI values, such as the Log-distance path loss model, the International Telecommunication Union's (ITU) model for indoor environments and an empirically based distance models [24, 39]. Only the Log-distance path loss model will be presented in more detail and further used in this report.

In an environment with no obstacles between the transceiver and receiver, the RSSI will be quite stable. If obstacles are introduced along with reflective material in the environment, the RSSI value will fluctuate more and small changes in orientation and positioning of either of the devices may result in larger deviations from the line-of-sight values.

2.1.2.1 Log-distance path loss model

The log-distance path loss model assumes a logarithmic relationship between the RSSI value and the distance [24]. It takes the form of

$$RSSI = RSSI_{d_0} - 10 \cdot n \cdot \log_{10}\left(\frac{d}{d_0}\right) + X_g \tag{2.1}$$

Where

$RSSI$ is the RSSI value at distance d

$RSSI_{d_0}$ is the RSSI value at distance d_0 (usually 1m)

n is the path loss exponent for the given environment

X_g is a Gaussian random component with zero mean value and standard deviation σ reflecting the path loss in dB caused by shadowing effects on the signal.

Rearranging, we can estimate the distance d from a known RSSI value:

$$d = d_0 \cdot 10^{-\frac{RSSI - RSSI_{d_0} - X_g}{10n}} \tag{2.2}$$

This model does require a certain knowledge of the environment to set reasonable values for n and the random variable X_g. There are tables with suitable values for n and σ which may be used, shown in section 2.1.2.1. However, more accurate approximation of these can be found by experiments on the actual system in question.

Environment	$\boldsymbol{n}$
In building, line-of-sight	1.6 - 1.8
Free space	2
Obstructed in factory	2 - 3
Urban area cellular radio	2.7 - 3.5
Shadowed, urban cellular radio	3 -5
Obstructed in building	4 - 6

Table 2.1: Path loss exponents, as given in [24]

The Log-distance path loss model also requires to do some measurements of $RSSI_{d_0}$ to correctly determine the offset for the equipment in use. Some publications have chosen to calculate $RSSI_{d_0}$ from free space path loss model given by

$$PL = 32.45 + 20 \cdot \log_{10}(f) + 20 \cdot \log_{10}(d) \tag{2.3}$$

Where

PL is the path loss in dB

f is the radio frequency in MHz

d is the distance in kilometers

However, different equipment can have very different RSSI values at one meter distance, so to achieve the best approximation with log-distance path loss model, $RSSI_{d_0}$ should preferably be found by actual measurements at distance d_0.

2.2 Trilateration and multilateration

This section will present two of the most commonly used techniques indoor positioning systems: trilateration and multilateration. Many will argue that the terms may be used interchangeably [42], but I've chosen to separate them according to the meaning of their respective prefixes, *tri* and *multi*, as a convenience when presenting them.

Trilateration uses three reference points, here called anchor nodes or simply nodes, each placed at a known position. Distances from the nodes to the tracked object are measured or estimated, and these distances are used to calculate where the object is positioned.

Trilateration is often presented as a geometrical problem where the only solution lies at the perfect intersection between three circles with radii equal to the distances to the respective anchor nodes, as shown in fig. 2.2. For a 2-dimensional positioning problem, three nodes are needed to find exactly one solution, and for 3-dimensional positioning, 4 nodes are needed. In the theoretical world this provides perfect positioning. Finding the exact solution is done by solving a set of three equations with two unknowns, the $x-$ and $y-$coordinates:

$$(x-x_1)^2+(y-y_1)^2=d_1^2 \tag{2.4}$$

$$(x-x_2)^2+(y-y_2)^2=d_2^2 \tag{2.5}$$

$$(x-x_3)^2+(y-y_3)^2=d_3^2 \tag{2.6}$$

Where x_i and y_i are coordinates for the nodes and d_i is the estimated distance, or radius, between the nodes and the object being localized, for $i=\{1,2,3\}$. Expanding and rearranging the equations gives a new set of two equations:

$$(-2x_1+2x_2)x+(-2y_1+2y_2)y=d_1^2-d_2^2-x_1^2+x_2^2-y_1^2+y_2^2 \tag{2.7}$$

$$(-2x_2+2x_3)x+(-2y_2+2y_3)y=d_2^2-d_3^2-x_2^2+x_3^2-y_2^2+y_3^2 \tag{2.8}$$

These two equations can be combined to solve for the two unknown coordinates, x and y by setting them as a system of equations:

$$Ax+By=C \tag{2.9}$$

$$Dx+Ey=F \tag{2.10}$$

Which gives the solutions

$$x = \frac{CE - BF}{AF - CD} \tag{2.11}$$

$$y = \frac{AE - BD}{AF - CD} \tag{2.12}$$

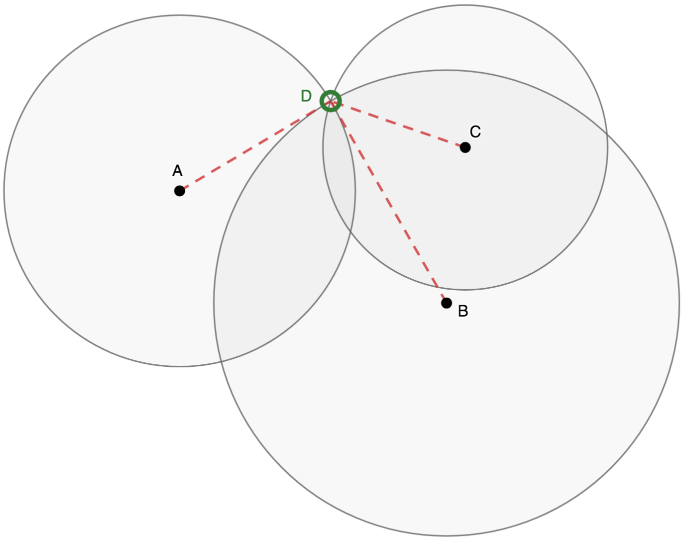


Figure 2.2: Trilateration with three anchor nodes with distances d_{AD},d_{BD} and d_{CD} to the asset location in point D. The figure shows the ideal case, where the circles having radii equal to the distances intersect in one point, the true asset location, D.

The same reasoning is applied when expanding the system to include z-coordinates and find coordinates in 3D.

However, solving the equations requires perfect measurements of the true distances to the anchor nodes, which is not feasible in a real deployment. There will be noisy and non-perfect measurements, and the circles (2D cases) or spheres (3D cases) will not intersect in one single point. An example of such a 2D case, where the circles intersect in an area rather than a point is seen in fig. 2.3. There can also be other variants, such as two circles overlapping, none overlapping and one circle enclosed by another. One solution proposed to overcome the issue in [2] is to continue to view the problem geometrically and find a solution to the problem by first calculating the center point between the circles and then adjusting the estimated asset location by weighing most the shortest estimated distances. Other approaches use a least squares algorithm to find an estimate for the position [46].

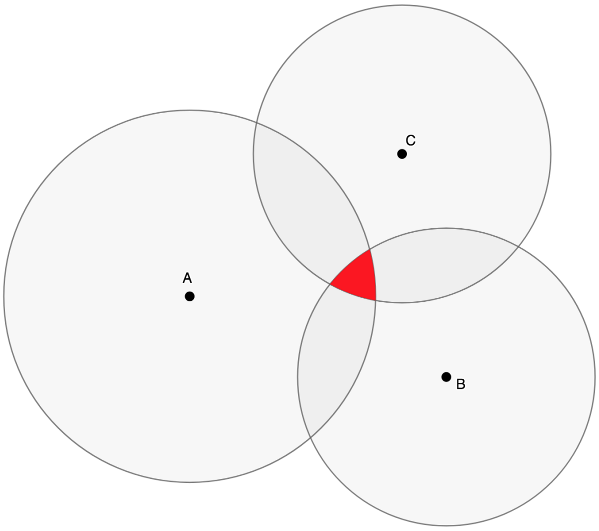


Figure 2.3: Trilateration with imperfect distance estimates between nodes and asset, resulting in no single solution for the positioning problem, but rather an are in which the asset is likely to be located.

Trilateration can be expanded to consider more than three nodes, which is what GPS devices do. This is called multilateration as it uses multiple nodes, but this extension of trilateration is often referred to as trilateration as well in the literature. It should be noted that there is some discussion as to whether multilateration refers to a different technological solution than multilateration, but the consensus seems to be moving in the direction of trilateration being a sub-group of multilateration [42].

As previously mentioned, all real-world measurements have an error component. Using distance estimates from more than three nodes may increase the accuracy of the positioning system. Positioning using multiple nodes is generally viewed more of an optimization problem of an overdetermined system rather than a geometrical problem. A common approach is to use a non-linear least squares algorithm to find the best estimate for the position of the tracked object [34]. The different nodes' estimated distances to the object may be equally weighted in the algorithms, or they may be assigned different weights defined by some relevant node property. Usually the distances are weighted by an inverse relationship to the distance, so that shorter distances are considered the more reliable by the positioning algorithm, end therefore tend to "pull" stronger in the estimated position [2]. An example of this is shown in fig. 2.4.

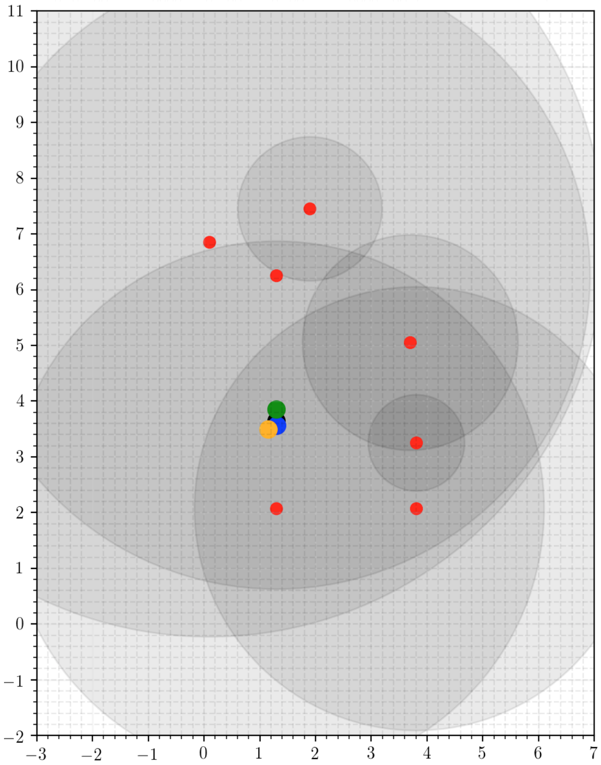


Figure 2.4: Multilateration using 7 nodes, represented by red dots and grey circles with radii equal to the estimated distance to the tracked object. The green dot indicates the true position of the object that's being tracked. The yellow, blue and barely visible black dots are position estimates using slightly different approaches. Some of the node overestimate the distances by a large margin, while others underestimate it equally much. No nodes has the distance exactly correct, but by combining all the nodes' distances, the position estimate is still within 20-50 cm of the truth in this particular scenario.

An example of function that's used to calculate the sum of errors for a set of n nodes in the non-linear least squares algorithm is the following:

$$\sum_{i=1}^{n} \sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} \cdot \frac{1}{r_i^2} \tag{2.13}$$

Where x_i, y_i and z_i are the preliminary position estimated, and r_i is the estimated distance between node i and the tracked object.

Using non-linear least squares, the (x, y, z) coordinates with the least sum of error are found. The example in eq. (2.13) uses the inverse squared distances as weight, giving less weight to nodes with larger distances. It's also possible to use other weightings, such as the cube of distances or simply distances.

2.3 Communication protocols

This section of the report will present the fundamentals of the communication protocols Ethernet, IP (Internet Protocol) and UDP (User Datagram Protocol) used for the wired network communication. Finally, it will also present the short-range wireless protocol Bluetooth Low Energy (BLE).

The wired protocols can be placed in both the Open Systems Interconnection, OSI, model and the TCP/IP model for Internet protocols. There is some discussion what model is best suited for this particular protocol stack, but in this thesis, the TCP/IP model will be used as it was specifically designed for these protocols.

The TCP/IP model describes how the communication protocols are built in a layered and hierarchical fashion to achieve communication between devices. Figure 2.5 shows how the layers are placed on top of each other and how the model compares to the OSI model, along with the protocols of choice in this thesis. Ethernet belongs both on the physical and link layer, IP on the network layer and protocols such as UDP and TCP belong on the transport layer [15]. On the application, layer it's stated that the thesis uses a proprietary protocol, more closely explained in section 3.3.4.2, as it's not a protocol standard per se.

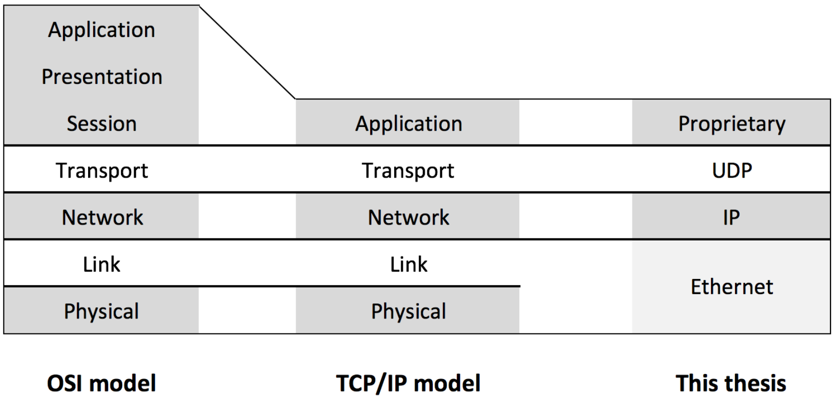


Figure 2.5: The OSI and TCP/IP models along with the protocol stack of choice in the thesis.

2.3.1 Ethernet

Ethernet is a very commonly used communication protocol for high speed data transfer over wires and is the de facto standard for wired communication in Local Area Networks, LANs. Ethernet can be used with multiple cable types, such as coax cables, optical cables and various configurations of twisted pair cables. Most commonly used to connect a computer to a network, is twisted pair cables, usually of the type Category 5 or the newer Category 6 cables that allows for higher transfer speeds [15].

Ethernet can provide data transfer rates from the Mbit/s range up to several hundred Gbit/s. Typical home and office networks use 100 Mbit/s or 1 Gbit/s, with the twisted pair standard cables named 100BASE-TX and 1000BASE-T, respectively [14].

In the TCP/IP model [15], Ethernet is found on the physical and link layers. The physical layer handles the transport of individual bits over a physical medium, in this case a wire. On the link layer data is put into standardized Ethernet *frames* with a specified structure shown in fig. 2.6.

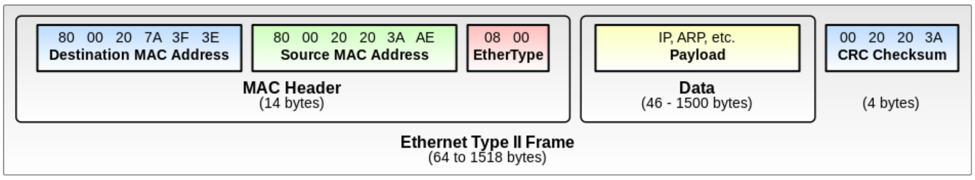


Figure 2.6: Ethernet frame structure. Source: `https://en.wikipedia.org/wiki/Ethernet_frame`

As seen in fig. 2.6, the Ethernet frame header holds information about the frame, such as source and destination addresses. The addresses are called MAC addresses, where MAC is an acronym for Media Access Control. They are 48 bits long, and usually written with hexadecimal representation. An Ethernet frame must contain these addresses in order to send data to the correct device, and for that device to be able to know it's the intended recipient and from which entity the data came.

Ethernet is a one-hop protocol, where data is sent from device to another on the same *collision domain*, which means that the devices are wired together, either directly or through a hub that forwards every bit received from one device to all other connected to that hub without any addressing. If a device receives an Ethernet frame addressed to itself or one of the defined broadcast addresses that every device listens to, it will proceed to read the payload of that frame. If the frame is addressed to anyone else, it shall discard that frame. Usually this is handled by only having to devices per collision domain and having an Ethernet switch transport data between the correct collision domains, avoiding the risk of receiving data intended for some other entity entirely.

Ethernet frames can have up to 1500 bytes of payload. Although one can set the payload in an Ethernet frame directly, the payload is usually received by the the Ethernet protocol from higher level protocols, and most often in the TCP/IP model that higher level protocol is the Internet Protocol, IP.

2.3.1.1 Ethernet network

LANs in private residences and businesses usually use either Ethernet or Wi-Fi to communicate. When using Ethernet, the devices are usually connected to a *switch* that handles the data traffic between the devices in the network. The switch is connected to each device through an Ethernet cable, and thus removes the possibility of data crashes on the wires, as only one peripheral device and the switch are on the same collision domain. This increases the total data transfer rate.

2.3.1.2 Power over Ethernet, PoE

A network switch is usually only operating as a communication central, but it can also provide electrical power to the connected devices. This feature is called PoE, Power over Ethernet. There are several ways to achieve power over Ethernet, and the IEEE 802.3 working group has standardized two in the Ethernet standard [14] and in amendments to the standard [13]. PoE removes the need for a dedicated power connection for the connected device, simplifying installation and deployment of Ethernet connected equipment. PoE is often used for equipment such as IP telephones, Wi-Fi access points and security equipment, but can power any device adhering to the PoE specification. The PoE enabled switch will negotiate possible power delivery with the connected devices on connection, and the standard opens for a wide array of power levels that is supported. The voltage level of the supply is between 37 V and 57 V, and the current maximum supported power on one line is 99 Watt.

2.3.2 Internet Protocol, IP

In the TCP/IP model, the Internet Protocol resides at the Internet layer [15]. IP provides end-to-end addressing of data packets, which means that one device can send data to another device using its IP address and not be restricted to a single hop such as is the case with Ethernet. The IP address is either 32 or 128 bits long for IP version 4 or 6, respectively. For the implementation in this project, IPv4 has been used.

IP datagrams consist of two parts: header and payload. The IP header holds information about the source and destination for the datagram. It also has information about protocol version, length of the datagram, checksum, time-to-live counter and other options fields [15].

IP datagrams contain data from higher protocol layers in its payload. It adds its header and passes the datagram on to the Ethernet protocol for encapsulation into an Ethernet frame.

2.3.3 User Datagram Protocol, UDP

On the transport layer in the TCP/IP stack, we find protocols such as TCP and UDP. Although the protocol model is named after TCP, it is optional to use TCP. Where TCP is a connection oriented protocol that transfers data reliably between entities, UDP is an inherently unreliable, best-effort protocol. In UDP, there are no built-in acknowledgements to ensure that data reaches its destination. Due to this, a UDP message, also called datagram, has smaller overhead than a TCP message, both in terms of the message size itself and because UDP requires less data to be sent since it has no connection to establish and maintain, and there are no acknowledgements on messages. This makes UDP a simpler protocol to implement than TCP, but with the trade-off that data may be unknowingly lost.

A UDP datagram consists of a UDP header and the payload. The UDP header is only 8 bytes long, compared to TCP's minimum header size of 20 bytes. The UDP header has information about the source and destination port numbers (16-bits numbers), the length of the datagram and calculated checksum. The datagram can in total be up to 65 535 bytes if one doesn't take into account the limitations of the underlying protocols [15].

Source and destination ports are software structures that can be used by applications to target each other. When sending a UDP datagram, one will first establish a UDP *socket*, which is a communications end-point in a network. After the socket is established, one can provide destination IP address and port number and send messages to the desired destination end-point. The information tuple provided to the socket when sending, is used to populate the corresponding fields in the UDP and IP headers.

The scenario is similar when receiving UDP datagrams: A socket is first established, then the application can bind to that socket using its own local IP address (typically 127.0.0.1) and an arbitrary available port number. It will then receive incoming data with its own (LAN) IP address and defined port number set as destination address and port, respectively.

2.3.4 Bluetooth Low Energy, BLE

Bluetooth Low Energy (BLE) was adopted into the Bluetooth specification version 4.0 by the Bluetooth Special Interest Group (Bluetooth SIG) in 2010 [8]. BLE uses the open ISM 2.4 GHz frequency band. The specification divides the band into 40 channels, where each is given 2 MHz band. Three of the channels are reserved for advertising while the other 37 are dedicated to communication between connected devices. Within a connection, the two devices negotiate a specific order to hop between the channels, a procedure called *frequency hopping*. By doing this, BLE reduces the risk of interference and errors in the data transfer if parts of the 2.4 GHz band is used by other nearby devices. The three advertising channels are placed strategically in the frequency band to avoid the most used Wi-Fi channels, channel 1, 6 and 11, shown in fig. 2.7. Frequencies for the advertising channels are shown in section 2.3.4.

Channel	**Frequency [MHz]**
37	2402
38	2426
39	2480

Table 2.2: BLE advertising channels

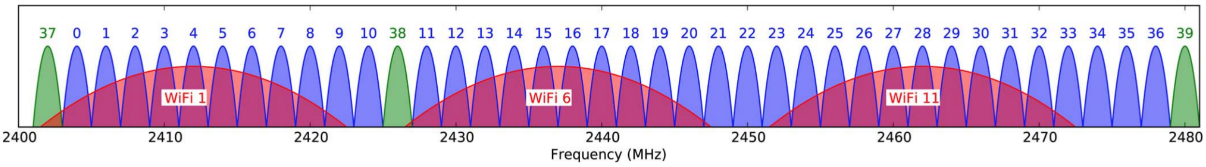


Figure 2.7: BLE channels overview with the most used Wi-Fi channels, 1, 6 and 11. The BLE advertising channels 37, 38 and 39 are placed in between these most busy Wi-Fi channels. Figure source: [6], figure 1, page 2419.

The current version of the Bluetooth specification, is Bluetooth 5. The specification added, among other things, 2.0 Mbit/s radio mode, doubled range for BLE and advertising packet extension.

BLE is a one-to-one communications protocol when a connection is established. However, a BLE device can also act as a *beacon* and send advertisements without entering a connection, and thus communicate one-to-many. Advertisements can be picked up by any BLE device that scans for such advertisements. The specification defines a dedicated *access address* for advertisements. The access address is used to address a BLE device as it will listen for packets for that specific address. Having the advertisement access address standardized allows for all devices to broadcast to anyone device that listens. Advertisements happen periodically every *advertising interval*, which can be in the range from 20 ms to 10.24 seconds. Each advertising interval, an advertisement packet is sent on all the three advertising channels, 37, 38 and 39. See fig. 2.8 for an illustration of how advertising and advertisement scanning may look like.

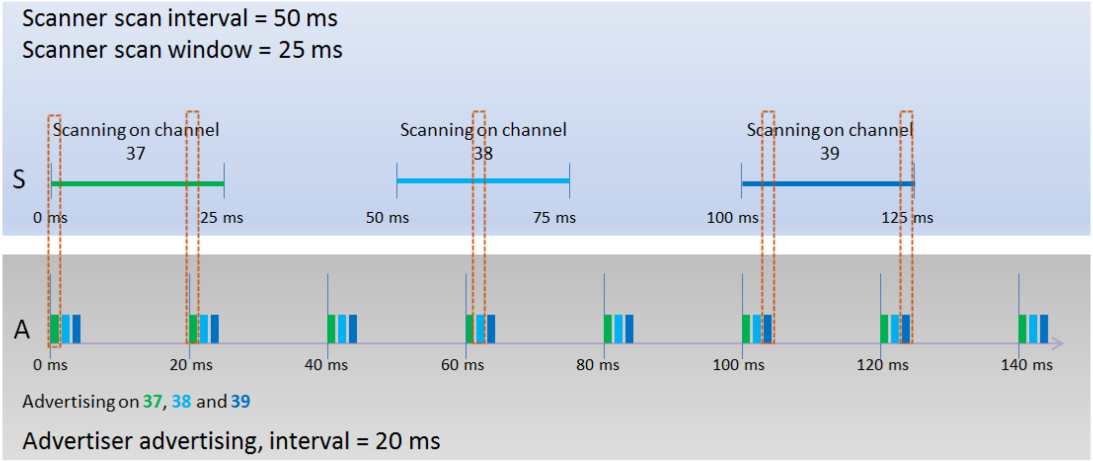
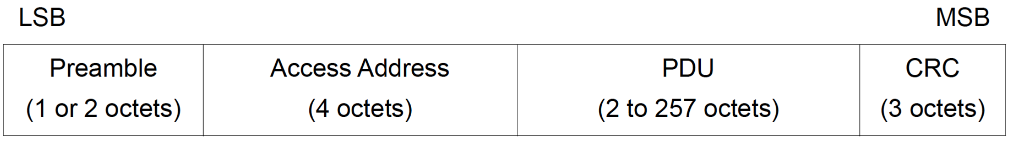


Figure 2.8: BLE advertising and scanning intervals. If the scanner is not continuously scanning, an added random period makes sure that the scanner will discover the advertiser if it continues to scan for some intervals. Figure source: [12].



LSB			MSB
Preamble (1 or 2 octets)	Access Address (4 octets)	PDU (2 to 257 octets)	CRC (3 octets)

Figure 2.9: BLE packet format for BLE's uncoded PHYs, 1 Mbit/s and 2 Mbit/s. Figure source: [9], Vol 6, Part B, section 2.1, figure 2.1.

A BLE packet has the format as shown in fig. 2.9. An advertising has the same basic structure. All BLE advertisements must have the access address `0x8E89BED6` according to the specification. A BLE advertisement packet has in addition to the BLE packet format a standardized format for the payload, shown in fig. 2.10. The advertising data (AD) structures contain first a *length* byte with the number of octets in the structure, followed by n bytes describing its type according to the specified list in [36], and finally $length - n$ bytes with the data content of the structure. Device name, TX power level and battery level are examples of AD types. An advertisement may contain multiple AD structure totalling up to 31 bytes in a standard BLE advertisement PDU.

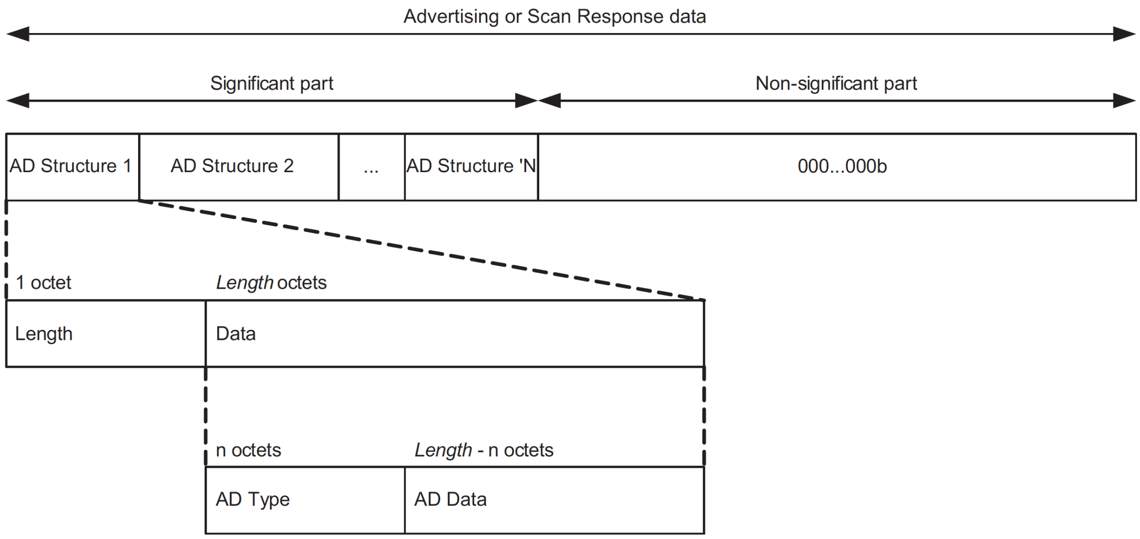


Figure 2.10: BLE advertisement packet payload format. There can be as many advertisement data structures as one wants, as long as the total byte length fits into the PDU of a BLE advertisement packet. Figure source: [9], Vol 3, Part C, section 11, figure 11.1

If an advertising device is configured to be connectable, two BLE devices can negotiate a connection setup. The devices will then configure which frequency hopping scheme to use, and set other connection parameters. Once the connection is established, the two devices can send data to each other every *connection interval*. The timing of these intervals is strict, and the radios will not be active between the connection intervals, at least not for that specific connection. Communication between BLE devices will not be presented in more detail, as the BLE devices in this project do not enter connections.

2.4 Kalman filter

In this thesis, Kalman filtering has been tested and used for both RSSI value filtering and 3D position filtering. This section presents a basic intuition of the Kalman filter and the equations used behind the scenes to calculate. It will not go into detail of the equations and its derivations as the Kalman filter per se is not a main focus of this thesis.

Measurements and sensor readings can be noisy, and using them unfiltered in a process can give unexpected and unreliable results. RSSI readings, for example, can change vastly from one measurement to the next, even though the involved devices and environment are exactly the same. Using raw RSSI readings and in extension distance estimates in a positioning algorithm can give very unstable position estimates, and often values that can't possibly be true. A device attached to a walking person can only accelerate and move so fast, but the inferred acceleration and speed from RSSI values are detached from these constraints. This is a typical case where Kalman filtering can be useful. The Kalman filter can use knowledge of the sensor's noise and previous state to determine what is statistically likely to be true in the next iteration. A sensor reading way out of what was expected can, depending on what is considered normal for the sensor, be mostly ignored and the previous state can be trusted more when computing the next state estimate. The opposite can also be true. It all depends on the system's characteristics and statistical properties [17].

The underlying idea of Kalman filtering, and several other filtering algorithms, is shown in fig. 2.11. Based on the previous estimated state, called the *posterior*, x_{t-1}, a prediction of the next state, called the prior *prior*, $\bar{x}_t$, is calculated. Then, the residual between the prior and the measurement is calculated, before the Kalman gain, K, is calculated based on which of the measurement and the prior is expected to be the most accurate. The Kalman gain scales the residual and places the final state estimate, x_t, somewhere between the prior and the measurement, depending on which of them is trusted the most.

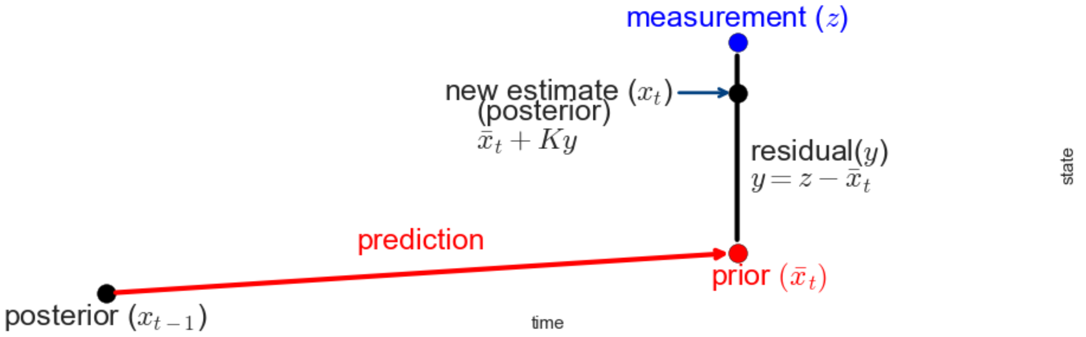


Figure 2.11: Simplified Kalman algorithm showing the steps of the algorithm from one state to the next. Figure source: [17], chapter 4.

This is the intuitive approach to Kalman filtering. The mathematics are complex and will not be presented in detail in this thesis, but is superficially introduced below.

When creating a discrete Kalman filter, it's initialized with the matrices $\mathbf{x}_k, \mathbf{F}_k, \mathbf{H}_k, \mathbf{P}_k, \mathbf{R}_k, \mathbf{Q}_k$, where

- $\mathbf{x}_k$ is the initial guess for the state
- $\mathbf{F}_k$ is the system matrix describing the relationships between the states
- $\mathbf{H}_k$ maps the true states to the observed ones
- $\mathbf{P}_k$ is the initial error covariance matrix
- $\mathbf{Q}_k$ is the covariance of the process noise
- $\mathbf{R}_k$ is the covariance of the measurement noise

The Kalman algorithm can be split into two stages: the prediction stage and the state update stage. Below, the Kalman filter is expressed mathematically for these two stages. Note that the variable names for the various matrices may differ between sources, and the version used here is the same as in the Python module `filterpy` ([16]) and [17, 41] to hopefully limit the confusion when reading this report.

Prediction stage:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{2.14}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k \tag{2.15}$$

Where

$\mathbf{B}_k$ is the control input matrix

$\hat{\mathbf{x}}_{k|k-1}$ is the predicted state estimate given the previous state, $\mathbf{x}_{k-1}$

$\mathbf{P}_{k|k-1}$ is the covariance matrix for the a priori estimate given the previous covariance matrix, $\mathbf{P}_{k-1}$

For the update stage we have:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.16}$$

$$\mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} \tag{2.17}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1} \tag{2.18}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{2.19}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{2.20}$$

$$\tilde{\mathbf{y}}_{k|k} = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.21}$$

Where

$\mathbf{z}_k$ is the measurement

$\tilde{\mathbf{y}}_k$ is the residual

$\mathbf{S}_k$ is the residual's covariance

$\mathbf{K}_k$ is the Kalman gain

$\hat{\mathbf{x}}_{k|k}$ is the updated state estimate

$\mathbf{P}_{k|k}$ is the updated estimate covariance matrix, to be used in the next prediction stage

The resulting *posterior* state estimate, $\hat{\mathbf{x}}_{k|k}$, is determined using optimal Kalman gain through these equations. This means that the residual error is minimized, and it's the optimal estimate for state given the information provided the algorithm. In practical use, one will have to initialize the matrices to reasonable values. It's important that the matrices describing the system model, $\mathbf{F}_k$ and $\mathbf{H}_k$ are correct, and in simple systems such as in this thesis that's usually not a problem. More difficult is to determine good values the $\mathbf{Q}_k$ and $\mathbf{R}_k$ matrices. This can be done through trial and error, but preferably through empirical trials or statistical knowledge of the system. This thesis uses a combination: initial guesses based on empirical testing and further adjustment by trial and error. More on the practical implementation of the filters in this thesis can be found in

3 | Indoor positioning platform

This chapter presents the implementation of the indoor positioning platform. First, an overview of the platform is given, then the hardware is presented, followed by the firmware implementation. Finally, processing of the collected data is presented.

3.1 Overview

The indoor positioning platform presented in this project consists of the following main parts:

- **Anchor nodes**, or simply nodes, with Bluetooth Low Energy and IEEE 802.15.4 radio capability and Ethernet connectivity
- **Tags** to be tracked by the nodes
- **Network switch** with PoE, Power over Ethernet
- **Back-end computer** to receive, store and process data from the nodes

The relationships between the different parts are shown in fig. 3.1, where an optional cloud service is included as well. The working principle of the platform is as follows: Several nodes are placed in an indoor environment, preferably attached to the ceiling. All nodes are connected to the same network switch through Ethernet compatible cables. The nodes continuously use their radios to scan for tags that broadcast packets with a predefined format. When the nodes receive a packet, they create a *scan report* with information about the packet. The scan report is then sent to the back-end computer using Ethernet. This happens every time a tag broadcast packet is received. The scan reports are parsed by the computer as they are received. The information in the reports is stored to a database and optionally processed in real-time to find the position of the tag. The same data can also be forwarded to a cloud service where the same can happen; both database storage and processing.

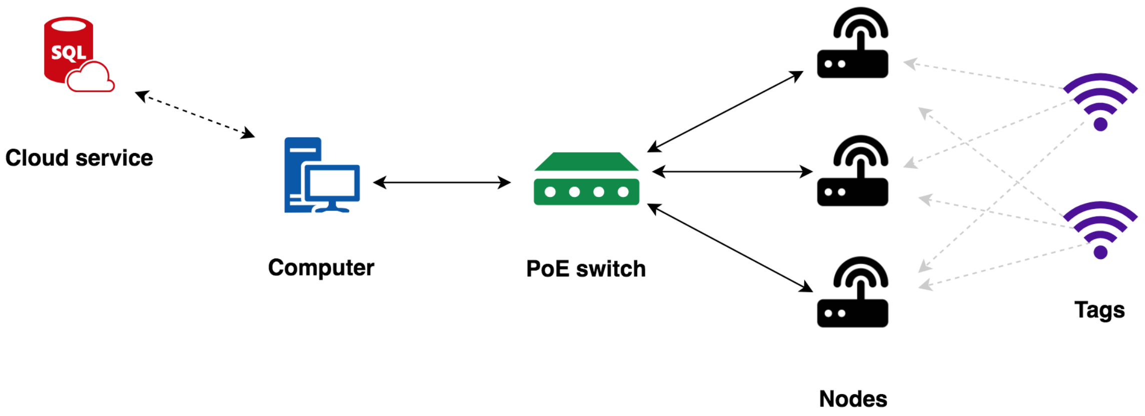


Figure 3.1: Positioning platform architecture overview.

The tags are beacons that broadcast a small amount of data using one of the radio modes that the nodes can pick up. There is only one-way communication from the tags, so they can be considered a passive part of the platform. A tag can be a specifically designed device for the task, or it can be a generic device such as a mobile phone sending out BLE advertisements at regular intervals. In this project, dedicated devices have been used for full control during development and testing. For a node to find a tag, the data packet the tag broadcasts needs to be of a specific format. It may for example be a Bluetooth Low Energy advertisement that must start with a certain sequence of characters or any other rule. This way, only relevant information is collected, which is particularly important if scanning for BLE advertisements or similar, as there are a lot of devices of likely no interest advertising within an average indoor environment, such as watches, phones, headsets, mice and keyboards.

The nodes send all collected data and metadata about the tags over Ethernet. This is done to leave the nodes' radios in a scanning state as much as possible and thus capture more information than if they needed to handle other data traffic wirelessly as well.

Note that the nodes are programmed in a way that allows them to also broadcast and act as tags themselves. This opens up for using the the tags to pick up signals from the nodes and run algorithms on the tag itself to estimate positions. This is a common approach used in the literature. This thesis will focus on the nodes collecting data, as given in the problem statement. However, the option to do it the other way is built-in to the platform and makes it more versatile as a test and development platform.

Every node that sends data to a computer, can also receive commands and be controlled by that computer, described in closer detail on page 47. In this project's implementation the same computer receives all data from the nodes and also controls them, but it's designed to allow splitting the tasks between two or more devices in a larger deployment.

The nodes are programmed with a bootloader that enables firmware upgrade over the network, using Trivial File Transfer Protocol, TFTP. However, the bootloader has been provided by Wiznet, and only briefly tested to confirm it's functional, and it will not be presented in more detail in this thesis.

3.2 Hardware

In this section, the following four main hardware parts of the indoor positioning platform will be presented:

- **Nodes**
- **Tags**
- **Network switch**
- **Back-end computer**

3.2.1 Nodes

The nodes are circuit boards made by Nordic Semiconductor designed for this purpose. They have nRF52840 SoC (System on Chip) that provides microcontroller and radio functionality and controls the Ethernet chip and other peripherals on the board. The boards have the following central features, which are also shown in fig. 3.2:

- **nRF52840 SoC**
- **Wiznet W5500** Ethernet chip and RJ45 connector
- Transformer for PoE voltage to 12V
- 1 white high-power LED and 2 green 3V LEDs controlled by the SoC that can be used for status indication
- Footprint for Wiznet WizFi310, Wi-Fi module
- Connector for time synchronization line
- USB hub and USB to dual UART bridge that enables UART from both nRF52840 and W5500
- Micro-USB connector

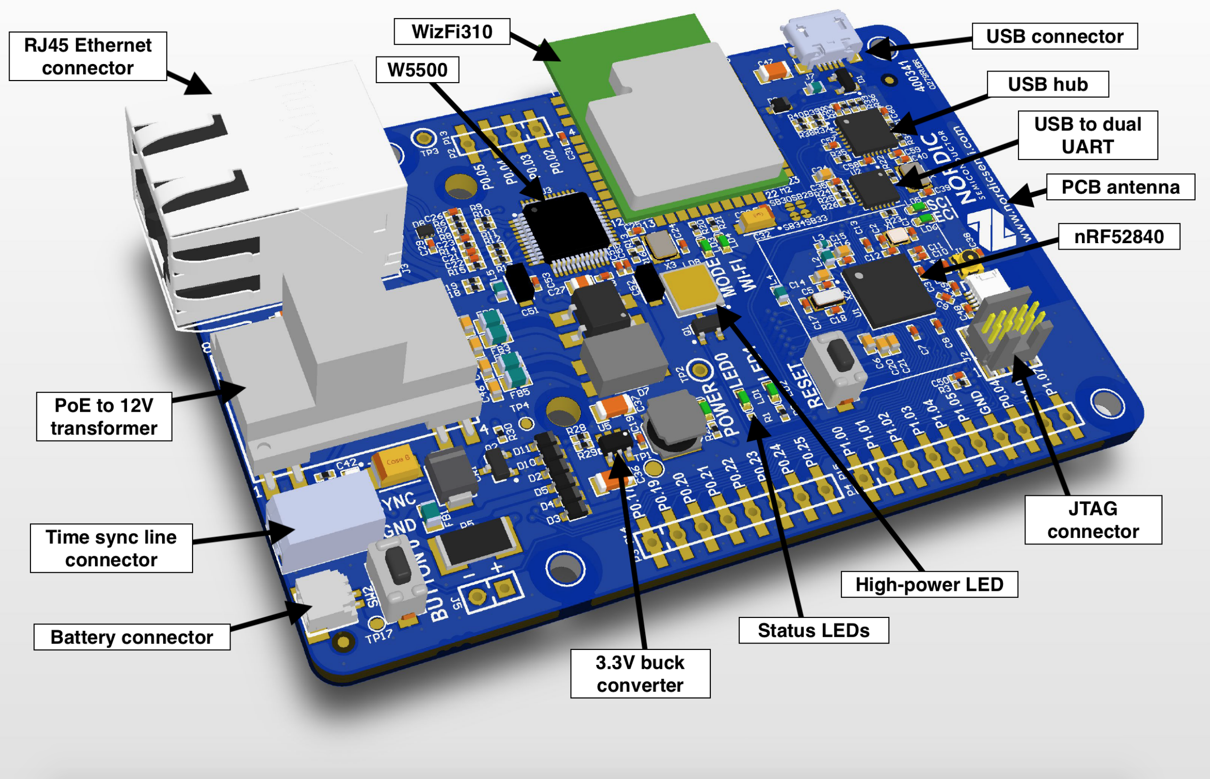


Figure 3.2: 3D model of the node board as used in this project, with the exception of the Wi-Fi module WizFi310 that's included for completeness in the model, but it has not been mounted on the boards that have been used.

The schematics and PCB layout for the node boards are included in Appendix A. In the following sections, the most important features of the node hardware will be presented in more detail.

3.2.1.1 nRF52840 System on Chip

To provide microcontroller and radio functionality, Nordic Semiconductor's nRF52840 has been used. nRF52840 is a System on Chip solution with CPU, hardware modules such as radio and timers, memory and general purpose input/output functionality in one single integrated chip [29]. This chip has in part been chosen because of its radio module with multiple radio modes. It has support for Bluetooth Low Energy, IEEE 802.15.4-2006 and various proprietary 2.4 GHz protocols. This means that popular communication protocols such as Bluetooth Low Energy, Zigbee and Thread are supported by nRF52840, and the application can switch dynamically between these protocols at runtime.

Of all the nRF52840 features, the following list are the most relevant for this project, with hardware module names in uppercase letters as found in the SoC's product specification [29]:

- ARM Cortex M4-F 32-bit processor with floating point unit
- 1 MB flash and 256 kB RAM
- Hardware modules (upper case words refer to module names as found in [29])
  - Communication protocol modules such as SPI, QSPI, I^{2}C, USB and NFC
  - GPI: General Purpose Input Output
  - GPIOTE: General Purpose Input/Output Tasks and Events. Module that can generate events when a GPIO pin changes state and can change the state of a GPIO pin based on an event, often generated by a different hardware module
  - TIMER: provides 5 timers with 16 MHz clock source
  - 2.4 GHz RADIO with the following bit rates [29]
    - Bluetooth Low Energy: with data rates 125 kbps, 250 kbps, 500 kbps, 1 Mbps and 2 Mbps
    - IEEE 802.15.4-2006: 250 kbps
    - Proprietary radio protocols with 1 Mbps and 2 Mbps

The board has a PCB trace antenna connected to nRF52840 (see fig. 3.2) along with a connector for an external antenna. Only the PCB antenna has been used in this project.

All the boards that have been used in this project use nRF52840 engineering revision A, with part name nRF52840-QIAA. This revision was an early version of the chip, and had the following relevant errors as found in the official errata document [28]:

- Anomaly 153 in [28] addresses issues with RADIO module related to RSSI values varying with temperature outside of the range +10 to +30 degrees Celsius. Relevant for this project is that the RSSI shows one increment too high if the temperature is above +30 degrees Celsius.
- Anomaly 110: subsequent BLE packets may not be received correctly. Solved in firmware by executing `TASK_DISABLE` after RADIO usage.
- Anomaly 142: Sensitivity is 1 dB lower than specified in [29], meaning its correct sensitivity is -94 dB. No workaround possible, according to [28].
- Anomaly 143: 100% CRC failure rate on some specific addresses.

3.2.1.2 Wiznet W5500

Wiznet's W5500 chip provides Ethernet connectivity for the board. The W5500 chip supports SPI up to 80 MHz frequency, which is well beyond the 8 MHz nRF52840 supports. W5500 has built-in, hardwired support for IP, UDP and TCP, and also natively supports other protocols such as ICMP, ARP and PPPoE. It supports up to 8 concurrent sockets [43].

W5500 is connected to nRF52840 as a slave using a standard SPI configuration with MISO, MOSI, SCLK and CS/SS lines. In addition, W5500 has an interrupt line, called INTn, The different lines have functions as described below:

- **MISO**: Master Input Slave Output. The slave can output data on this line.
- **MOSI**: Master Output Slave Input. The master outputs data on this line.
- **SCLK**: Serial Clock. Clock line controlled by the master.
- **SS**: Slave Select / Chip Select. Select signal for the slave device, controlled by the master, or optionally hardwired if the chip will always be selected.
- **INTn**: Interrupt pin that W5500 can be configured to change digital state on certain events.

In this context, nRF52840 is the SPI master, and W5500 is the SPI slave, which means that nRF52840 controls the SCLK and SS lines, and outputs data on the MOSI line. W5500 outputs data on MISO line and controls the INTn interrupt line.

From W5500, the TX and RX pair lines go to an RJ45 connector where the boards are connected to the network switch. The nodes are connected to the switch through minimum Cat 5e twisted pair cable, which is sufficient to provide 100 Mbit/s transfer speed and Power over Ethernet.

3.2.1.3 Time synchronization line

The nodes are also connected to a synchronization line and ground using the connector shown on the bottom left in fig. 3.2. These lines are in this project connected in a daisy-chain configuration. Other configurations can also be used as long as they serve the same purpose and to not introduce significant delays or impedances that affect the signal detection by the boards.

Using the synchronization line, a signal can be sent at regular intervals. Whenever the synchronization signal is detected by a node, it will reset a timer that runs continuously. Thus, the node can always know how long time has passed since the last synchronization signal. The point of this is to make the platform resilient to clock drift on individual boards so that the user can know exactly when a radio signal was received at any given node, and therefore it's also important that the way the lines are connected doesn't introduce delays.

3.2.2 Tags

A tag can be just about any piece of hardware capable of broadcasting the specified data using one of the radio modes that the nodes can scan for. In this project, PCA10056 nRF52840 Development Kits have been used for testing to have full control over what's being broadcasted and how it's done [26]. However, also devices such as mobile phones and just about any other BLE or IEEE 802.15.4 device can be tracked by the platform. The hardware is not important, but the radio mode and configuration of the radio packets have to match what the nodes expect to see.

3.2.3 Network switch

The network switch used in the platform is a Cisco Catalyst 3560-CG. It's a Gigabit Ethernet switch with PoE capability and can thus power the nodes that are connected to it. All nodes are connected to the same switch, along with a computer that receives all data from the nodes for processing. In total, there are 8 ports with optional PoE, and 2 without on the switch's interface. Even though an 8-port version is used in this platform, the principles are applicable to other switches with more ports.

The switch is configured to use Dynamic Host Configuration Protocol (DHCP), which is a protocol that handles the connected nodes and assigns IP addresses to them within a defined address range. All nodes and computers that connect to the switch will be automatically assigned an IP address as long as the device has DHCP enabled. The switch has been configured to provide IPv4 addresses in the 10.0.0.0/8 address range for private networks.

3.2.4 Back-end computer

All nodes collect and send information about nearby tags to a computer where the information is stored and can be processed in real-time. This computer is connected to the same LAN and switch as the nodes, and receives an IP address in the same range. The computer must be able to run Python and a MySQL database to function as intended. However, the minimum requirements are low as the network traffic is relatively modest and in minimum configuration, all the back-end computer needs to do is store the received information into a database.

If more advanced processing is to be done in real-time, such as tracking of an asset or multiple assets along with graphical presentation, the demands increase, depending on how many tags and what processing is to be done. There's not been done thorough measurements and investigation into the area in this project, but empirically it's been shown that a MacBook Pro 2014 generation 2.2 GHz Intel Core i7 with 16 GB RAM has handled the workload without any problem or hiccup. A more limiting factor than the actual hardware seems to be that Python natively utilizes only one CPU core with the project's current software solution.

3.3 Firmware

This section will present the firmware that's been created for the platform to work as intended. Firmware is in this context defined as the software that runs on the microcontrollers. In this project, firmware has been written for the nodes' nRF52840 and for the nRF52840s on the tag development boards. Firmware has also been written at an early stage in development for dedicated time synchronization boards. Later the time synchronization firmware was incorporated into the node firmware. The node firmware has been split in two: the bootloader and the application. The bootloader enables the nodes to receive firmware updates over Ethernet using Trivial File Transfer Protocol, TFTP. The bootloader was delivered nearly complete by Wiznet, with only minor modifications needed, and is therefore not presented in detail here.

First, the tools used to write the firmware is presented. Next, the tag firmware is presented, followed by time synchronization and finally node application firmware.

3.3.1 Tools end prerequisites

The bootloader firmware for the nodes has been mostly delivered by Wiznet to suite the requirements of their TFTP implementation. Wiznet uses Keil µVision v5 IDE for their bootloader, and therefore the modifications that's been necessary to do, such as changing IP address for over-the-network firmware updates, has been done in the same program.

All application firmware for both nodes, tags and dedicated time synchronizing board is written in C programming language using SEGGER Embedded Studio v3.31, a cross-platform IDE (Integrated Development Environment) for embedded programming [35]. SEGGER Embedded Studio has built-in support for nRF52840, and it's thus a well-suited platform for development in this project. Programming of the chips can be done directly from the IDE.

Nordic Semiconductor's Software Development Kit 14.2 has to some extent been used when writing the application firmwares, with the most important part being the SPI driver and register definition files [27]. Wiznet has used the same SDK in the bootloader implementation, along with Nordic Semiconductor's SoftDevice s140 v5 [30]. Drivers for Wiznet's W5500 Ethernet chip were provided to my earlier project in [10], and the same are used here and found in [44]. Other than that, an existing radio driver solution made by a Nordic Semiconductor employee has also been used and modified for this implementation [11].

3.3.2 Firmware for tags

As presented in 3.1, the tags are beacons that broadcast a very limited amount of data using one of the radio modes supported by the nRF52840, and the tags have no tasks other than this.

On the tags, a timer runs continuously and triggers an interrupt every 200 ms, called the advertising interval, in which the tag sends out a BLE advertisement packet on each of the BLE advertising channels, 37, 38 and 39. The advertisement packet follows the standard BLE packet format shown in fig. 2.9. During testing, the advertisement packet payload format shown in fig. 2.10 has not been used for convenience to allow for an incremental counter to be sent as payload in each advertisement interval. This does not affect how the nodes pick up the advertisement packet, it just slightly simplifies the practical implementation of the positioning algorithm.

In the BLE standard, an access address, `0x8E89BED6`, is defined for advertisement packets. This access address is common to all advertisement packets, so that all devices will recognize it and receive it. To avoid the that the nodes receive a lot of irrelevant data that takes up radio time and sends a lot of useless data to the back-end computer, a custom access address, `0xBE89BED6` has been defined in the tag's firmware. As presented in more detail in section 3.3.4, the nodes can choose to listen for this access address or the standard BLE advertisement access address.

The tag firmware has been written without use of Nordic Semiconductor's SDK and SoftDevice to let the application maintain control over which BLE channel is used at any time.

3.3.3 Firmware for time synchronization

The time synchronization feature is a wire that's daisy chained to all the nodes within the network. The voltage of the line alternates between 0 and 3.3V with a given interval. Default interval has been set to 1.0 second. A separate firmware was initially written for a nRF52840 Preview Development Kit to control the synchronization line. Time synchronization functionality was later merged into the node firmware, so that any node can take the role as time synchronization controller within a network of nodes where all are connected to the same time synchronization daisy-chain.

The following description and the actual implementations of the time synchronization firmware are exactly the same on both the dedicated time synchronization development kit and on a node that takes the role as time synchronizer.

Toggling of the synchronization line voltage between 0 and 3.3V is achieved by using the GPIOTE, TIMER and PPI modules in the nRF52840. The GPIOTE module gives control of inputs and outputs, the TIMER module enables precise timing, and the PPI module makes it possible to have TIMER events trigger GPIOTE tasks. A timer is in the firmware by default configured to trigger a compare event every second. As this event happens, a designated PPI channel connects the event to a GPIOTE toggle task that toggles the configured output pin between high and low digital output, which is 3.3 V and 0, respectively. Using PPI allows this to happen without CPU usage, which is an important feature to maintain accurate intervals if concurrently running other time-sensitive tasks such as scanning for tags or sending scan reports.

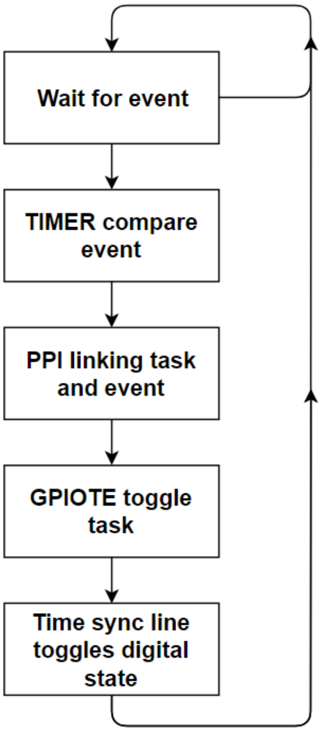

Wait for event
TIMER compare event
PPI linking task and event
GPIOTE toggle task
Time sync line toggles digital state

Figure 3.3: Flow chart for time synchronization firmware.

3.3.4 Firmware for nodes

The application firmware on the nodes has three main tasks to perform to function as intended

- **Scan** for tags and create scan reports
- **Send scan reports** to back-end computer over Ethernet
- **Listen for commands** from back-end computer and respond adequately

These tasks are performed in the main loop of the firmware. Before they are able to run, a startup sequence has to be executed for the node to be configured properly. In the following sections, the startup sequence and main loop are presented.

3.3.4.1 Startup sequence

When a node is powered up, it goes through a startup sequence with the following steps:

1. Initialization of hardware modules
    - High-frequency clock
    - **GPIOTE**: General Purpose Input/Output Tasks and Events module that can sense state or set states of input and output pins
    - **PPI**: Programmable Peripheral Interconnect module that can link tasks and events of different hardware modules
    - **SPI**: Serial Peripheral Interface bus module for communication with W5500 Ethernet chip
2. LEDs are set to initial state
3. Ethernet is initialized
4. IP address is obtained using DHCP
5. The node waits for broadcast message from back-end computer containing the computer's IP address and assigned UDP port number
6. Node receives IP and port from computer and establishes a UDP (or optionally TCP) socket with the given IP and port tuple
7. Scanning for tags is initiated
8. Node enters the main loop

Waiting for IP and port number from the computer is a blocking function, so the node cannot enter the main loop without having established a socket for communication with the computer. This means that all nodes may be powered up and go through the startup sequence, but will not scan for tags until the computer sends them its network information.

3.3.4.2 Main loop

The main loop includes the following repeating sequence, as also shown in fig. 3.4:

1. If scanning is enabled
   - Scan for tags on BLE channels 37, 38 and 39, and create corresponding scan reports if tag is found.
   - Send scan reports for each of the channels to the computer using the UDP (or TCP) socket opened in the startup sequence
2. If advertising is enabled
   - Set incremental counter value as payload for BLE advertisement
   - Send advertisement on channel 37, 38 and 39 consecutively
3. Optionally send who-am-I messages identifying the node with MAC address and IP address
4. Check for new commands from the back-end computer

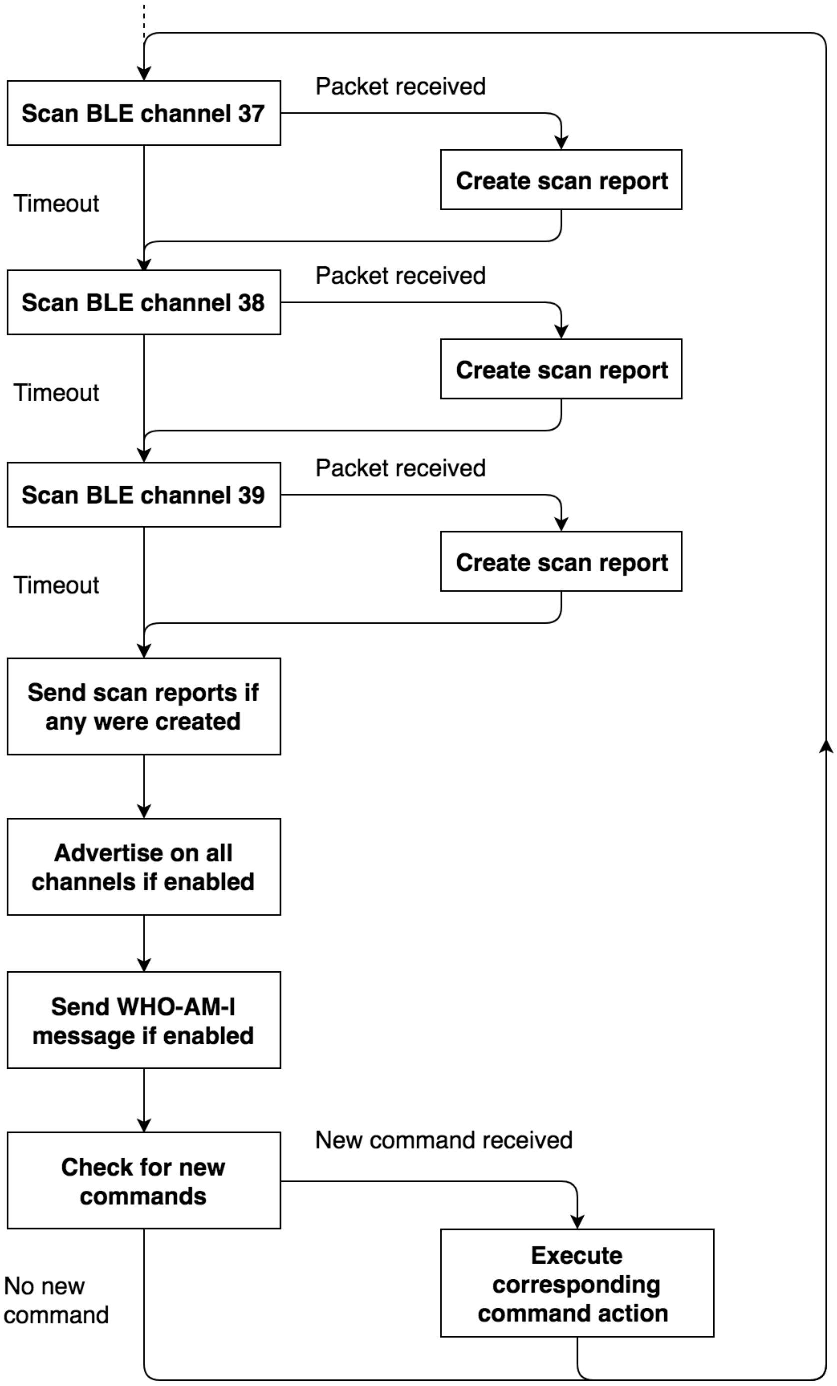


Figure 3.4: Flow chart for the main loop of node firmware.

Scanning for tags

When scanning for tags, the firmware is programmed to do so in a sequential manner, where the three BLE advertisement channels are scanned in the sequence 37, 38, 39. If implementing for IEEE 802.15.4 radio mode, a similar setup could be used for those channels.

Scanning uses the RADIO and TIMER modules in nRF52840 [29]. The radio module can be configured to use BLE modulation for bit rates ranging from 125 kbps to 2 Mbps. In the firmware it's statically set to 1 Mbps. Every time the radio is initialized, the radio frequency is set to the frequency corresponding to the BLE channel number on which the current advertisement packet will be transmitted, see section 2.3.4 for specified frequencies.

Scanning for tags starts immediately after the startup sequence is finished. First, the node scans for advertisements on channel 37 matching the defined access address, `0xB9BED600` (see section 3.3.2 for reasoning behind this choice). If no tag is found within 500 ms, it proceeds to scan channel 38. The same timeout limit applies when scanning channels 38 and 39 as well. This ensures that the main loop will not hang if there are no tags nearby. The timeout of 500 ms is set arbitrarily with a tag advertising interval of 200 ms in mind, and can be changed in the firmware's configurations header file, `config.h`.

When an advertisement is received on one of the channels during scanning, a **scan report** is created. The scan report is a C structure data type that's used to populate the following key/value pairs in a JSON (JavaScript Object Notation) string:

Field name	**Description**
`NodeID`	The node's network interface MAC address
`Timestamp`	Time since last synchronization signal
`Address`	The discovered tag's BLE address
`RSSI`	Measured RSSI value
`CRC`	every broadcasted message picked up by the node has a CRC (Cyclic Redundancy Check) associated with it. The message is considered erroneous and invalid if the CRC is not equal to 1
`LPE`	LPE is a self-made acronym for Long Packet Error indicating that the received packet was longer than theoretically possible if it was valid, indicating it was for example corrupted by interference
`Counter`	The tags broadcast an incremental counter for every message they send on a given channel. Can be used to avoid applying positioning algorithm to outdated data
`Sync_controller`	`1` if the node is controlling the time synchronization, `0` otherwise

Table 3.1: Scan report content

After the scan report is encoded into a JSON string, it's sent in a UDP datagram to the back-end computer.

Advertising

All nodes have the ability to start advertising standard BLE advertisements using the same access address as for scanning. This is enabled via the command system, and uses the same code as presented in the section about tag advertisements, section 3.3.2. A node can have both advertising and scanning enabled at the same time.

Command system

To be able to control the nodes from a central computer, it was decided to implement a command system. The command system is very basic and has been created with no regard to data security and robustness as a convenience and proof of concept. It does not follow a specific protocol standard, but can rather be seen as an ad hoc proprietary application layer protocol.

Commands are broadcasted as ASCII strings in a UDP datagram from a computer and received by the nodes on the same socket as is used in the startup sequence in section 3.3.4.1. The commands are then parsed in the nodes and corresponding action is taken if the received command matches any of the defined valid commands. To be considered a valid command, the string has to start with the substring “`CONTROL_COMMAND:`”, followed by a byte with a valid command code. The commands can initiate the following actions:

- Enable and disable scanning and advertising
- Control the high-power LED: turn it on or off and set its light intensity by regulating the PWM duty cycle
- Start and stop sending who-am-I messages
- Receive a new IP address for the back-end
- Receive message that new firmware is available for download over TFTP
- Change access address for the BLE radio
- Become time synchronization signal controller and set the interval for the signal

A complete list of commands and corresponding command codes is found in appendix B.

3.4 Data processing

All scan reports created by the nodes are sent to a central computer. This computer can also be viewed as the server or master in the network, as it serves the nodes with necessary control commands and receives and stores information about detected tags in return.

All messages from a node to the server contains a an ASCII string with scan reports formatted as JSON (JavaScript Object Notation) key/value pairs, as described in section 3.3.4.2. The reason for using JSON becomes evident once the datagram is received at the computerM Parsing of JSON strings to native data types is a straight-forward task in most programming languages using either built-in functionality or easily available modules.

3.4.1 Overview

The back-end computer's primary role is to collect the data from the nodes and store it to a database. In addition, and depending on the processing power of the computer, the functionality can be expanded to also cover real-time tracking of the tags of various complexity.

On the back-end computer, Python 3 has been used as the programming language. Python was, despite limited familiarity and knowledge, chosen because of its relative ease of use and the many packages that provide functionality such as sockets for UDP and TCP, advanced mathematical operations and tools for graphical presentation. Python is also platform agnostic, and the same code can run on many operating systems with minor, if any, modifications. Its drawback is that it's not the fastest language [22]. It was considered to use a language with a simple way to achieve multithreading, such as Go, but in the end Python was chosen due to its ease of use and easily available resources.

In fig. 3.5, a flow chart for the back-end processing program is shown. The program receives scan reports from all nodes, parses them and stores the data to a database. It can then optionally apply the positioning algorithm presented in algorithm 1, page 60, to the data before repeating the loop. The different parts of the program will be discussed in this section.

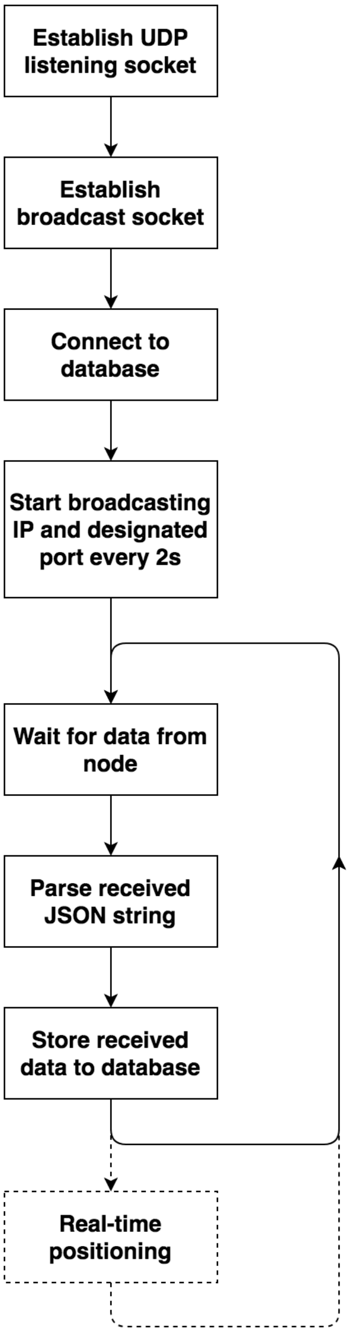

Establish UDP listening socket
Establish broadcast socket
Connect to database
Start broadcasting IP and designated port every 2s
Wait for data from node
Parse received JSON string
Store received data to database
Real-time positioning

Figure 3.5: Flow chart for back-end computer processing of scan reports from the nodes. The minimal option is to run the program without real-time positioning and tracking, and only store data to database. The content of the last, optional box “Real-time positioning”, is shown in fig. 3.6.

3.4.2 Receiving data

To receive data, the program creates a UDP socket using Python’s `socket` module, and binds it to its local IP address and an unused port. After the socket is successfully initialized and established, the program can receive all data that is sent to that particular port number using the socket's `recvfrom` method.

If the received data format matches the JSON format that is expected from the nodes, Python's `json` module is used to parse the data into a Python dictionary which can then be used to access the data in the rest of the program.

3.4.3 Data storage in database

After reception and parsing of the JSON string, the script stores all data into a database. It can be either be a local database solution, a cloud-based database service, or a combination of both, where data is synchronized automatically to a cloud-based database. The database in the project implementation is a MySQL database running locally on a MySQL v5.7.21 server. Cloud-based solution using Amazon Web Services with similar setup has only been briefly tested to confirm that it's working.

The database has the general structure described in table 3.2, and can be expanded with more fields to accommodate implementation specific needs.

Table 3.2: Database structure

Name	Description
ID	Unique database record ID
Date_time	The time at which the record was generated
Node_ID	Unique node ID, equal to the Ethernet network MAC address
Timestamp	Time when node received radio packet. In units of 1/16 μs since time sync
IP	IP address of the node
Address	The tag's radio address
Channel	The channel number that the node received the tag data on
Counter	Optional, incremental number sent by the tag to identify the beacon signal
TX_power	The advertised TX power of the tag
RSSI	The RSSI value as recorded by the node
CRC	The CRC register status upon reception of the tag data.
LPE	Long Packet Error. 1 if an error makes the radio packet longer than possible if valid.
Sync_controller	1 if the node is the one controlling the synchronization signal, 0 if not.

Connection to the database is achieved by API calls using the Python module `PyMySQL`. Data is stored to the database using the database connection handle returned upon connection establishment and standard SQL `INSERT` queries.

3.4.4 Positioning and tracking

Processing of the data from the nodes can be done either in real-time as the data comes from the nodes, or at any other time by fetching data from the database. The positioning procedure will be the same regardless of how the program accesses the data.

When processing data from the nodes, a Python implementation of the algorithm presented in chapter 4 is used. A flow chart of the data processing implemented Python program is shown fig. 3.6. This chart represents the content of the “Real-time positioning” box shown at the end of fig. 3.5.

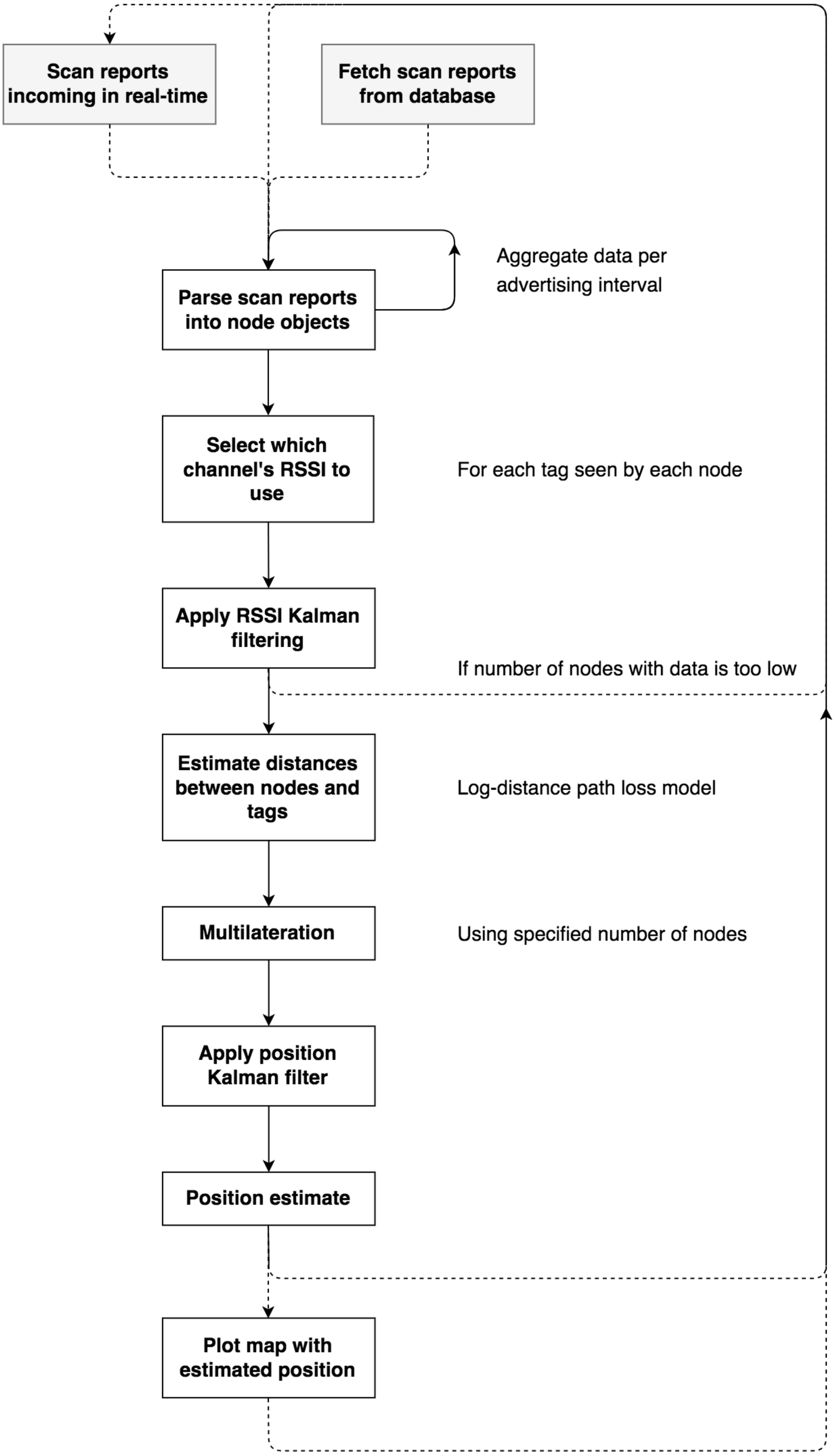

Scan reports incoming in real-time
Fetch scan reports from database
Parse scan reports into node objects
Aggregate data per advertising interval
Select which channel's RSSI to use
For each tag seen by each node
Apply RSSI Kalman filtering
If number of nodes with data is too low
Estimate distances between nodes and tags
Log-distance path loss model
Multilateration
Using specified number of nodes
Apply position Kalman filter
Position estimate
Plot map with estimated position

Figure 3.6: High-level flow chart for the Python implementation of the positioning algorithm. The two shaded boxes at the top are the two different entry points: receiving node scan reports from the network or from database.

The data is stored in the python script in node and tag objects that have been created for the purpose. The node class contains properties about the node, such as its ID, its position in the current coordinate system and its IP address. It also has a list of tags that it has recorded data from. When a new tag is discovered by the node, the tag is added to this list. Every tag in the list is a tag object. The tag class holds the following information about the tag: BLE address, last counter value, a list of RSSIs, a Kalman filter instance and a Kalman-filtered RSSI.

Data from the nodes is stored into these data structures as they are arrive at the socket or as the database result is iterated over. Further, the Python program follows the exact steps as shown in algorithm 1 to estimate the position at a tag, which will not be repeated here other than visualized in fig. 3.6. It's also possible to run the positioning algorithm with variations in what parts are enabled. For example to compare how the Kalman filters affect the estimates.

When a position estimate is obtained, the Python module `matplotlib` can be used to plot the position in a coordinate system or map. An example of this is shown in fig. 3.7, where three different combinations of the positioning algorithm were implemented for the same tag.

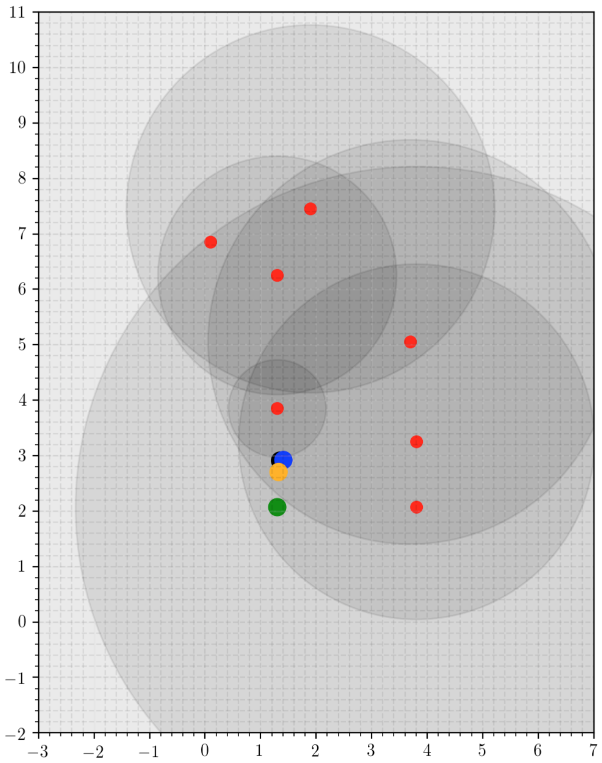


Figure 3.7: Plot of position estimates using multiple approaches. The red dots are nodes, surrounded by grey circles with radii equal to the estimated distance to the tag. Green dot is true tag position, yellow dot is estimated position using Kalman filtered RSSI and position, blue dot is using only filtered RSSI, and black dot is using no filters. The axes are in meters.

3.4.5 Kalman filtering

The RSSI and position Kalman filters are implemented in Python using the `filterpy` module by Roger Labbe [16]. For a more thorough explanation of the Kalman filter, its functionality and mathematics, see section 2.4.

The RSSI filter is a one-dimensional Kalman filter with one state, the estimated RSSI, and one input, the RSSI as measured by the node. One Kalman filter is used for the RSSI values attributed to each tag a node has found. The RSSI filter matrices are as follows, with values based on the findings in [2] and experimentation for the Q and R values:

$$
\begin{aligned}
F &= 1 \\
H &= 1 \\
P &= 5 \\
Q &= 0.07 \\
R &= 3.0
\end{aligned}
$$

The position filter is more complex. It has six states; position, p and velocity, v, in each of the x-, y- and z-directions, and three "measurements"; the position estimates $\hat{p}_x, \hat{p}_y$ and $\hat{p}_z$. Hence, the state matrix $\mathbf{x}$ is a column vector of length 6:

$$\mathbf{x} = \begin{bmatrix} P_x & v_x & P_y & v_y & P_z & v_z \end{bmatrix}^T \tag{3.1}$$

which is initialized to

$$\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.2}$$

Further, the other system matrices are initialized as follows:

$$\mathbf{F_k} = \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Where dt is the advertising interval of the tags given in seconds, defaulting to 200 ms.

The observation model matrix, $\mathbf{H_k}$ is given by

$$\mathbf{H_k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} z \tag{3.4}$$

The **P**, **Q** and **R** matrices are as mentioned determined partially based on the work presented in [2] and partially on experiments. They are not fine-tuned, and when the filters performed well enough, it was decided to not spend much time tuning them to perfection.

$$\mathbf{P_k} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{3.5}$$

$$\mathbf{Q_k} = \begin{bmatrix} 0.1 & 0.1 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0.1 \\ 0 & 0 & 0 & 0 & 0.1 & 0.1 \end{bmatrix} \tag{3.6}$$

$$\mathbf{R_k} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \tag{3.7}$$

These matrices are passed as 2-dimensional `numpy` arrays to the `filterpy` module's Kalman filter class. Every time a new position is estimated using the positioning algorithm (algorithm 1), the two stages of the Kalman are executed using the Kalman class' `predict` and `update` methods: first the prediction, and then the update stage based on the estimated position.

3.4.6 Multilateration

Multilateration is implemented with non-linear least squares algorithm to find the position estimate based on a list of tuples of with the format `(nodePosX, nodePosY, nodePosZ, distance)`. The Python module `scipy` and its method `least_squares` has been used to do this. The function for which the sum of errors is minimized is shown in eq. (2.13). `least_squares` also allows to set bounds for what the tag position can be estimated to [4]. The bounds can be left out, or rather set to the actual outer bounds of the coordinate system the tag moves within. It can also be sued to lock the z-coordinate to in practice perform two-dimensional positioning. However, this can also lead to a larger Euclidean distance error between the estimated and true positions than what would have been the case without the bounds. The multilateration function returns a tuple of the form `(posEstX, posEstY, posEstZ)`.

3.4.7 Controlling the nodes

All the nodes are connected to the same network and listen for UDP broadcast messages. As this project does not consider data security, the platform at the current stage allows the nodes to be controlled by any device on the network as long as the device sends messages adhering to the predefined control command scheme presented in section 3.3.4.2.

To send commands, two different approaches have been used. Sending commands manually has been done using the program Packet Sender for macOS (`https://packetsender.com/`). Packets for each command has been created and stored in the program, and is sent by clicking a button next to the desired command. Automated commands as part of a larger program has been done using Python. Also here the command packets have been created according to the defined format seen in Appendix B and are sent using a broadcast socket.

4 | Proposed positioning algorithm

This chapter presents the positioning algorithm that's been used on the data collected by the positioning platform in this thesis. The design choices and the algorithm itself are explained.

There are many approaches to indoor positioning system, as presented in section 1.4. Some algorithms, such as fingerprinting, rely on thorough groundwork and testing in the environment where the objects will be tracked [6]. This is generally the most precise way of doing indoor positioning with BLE and similar technologies, but it's work demanding and heavily dependant on the environment being static for the fingerprint to be valid. Other approaches are more dynamical, and attempt to work well in all environments without need of pre-work such as fingerprinting before positioning and tracking of assets can begin, usually at the cost of lower accuracy.

This report presents a positioning algorithm that combines known approaches and apply them to the data from the positioning platform also presented in this report. The use of the positioning platform opens up for more processing power in the back-end computer than what is often available at the devices performing positioning, and it's attempted to take advantage of this by applying Kalman filters to the RSSI values and estimated positions. Because the anchor nodes used in the platform have a custom firmware written specifically for this purpose, more information than usual in such setups can be extracted, with control over radio frequency, CRC status and potentially information on corrupt packets that passed the CRC test. There's also the option to use specific frequencies if the standard ones are heavily trafficked, as may well be the case in environments with many Bluetooth and Wi-Fi devices.

The following approaches have been combined for the positioning algorithm:

- Log-distance path loss model for the RSSI-distance relationship
- Dynamic channel selection for each new unique advertisement from the tags, called *channel diversity* in [2]
- Multilateration with theoretically unlimited number of nodes contributing in the

position estimation, using non-linear least squares

- Kalman filters for both RSSI values and 3-dimensional position estimate

The positioning algorithm pseudo-code is shown in Algorithm 1 works on a set of data that logically belongs to the same logical unit or time frame. This can for example be all data that's been received during one of a tag's advertising interval if it uses BLE, where data is sent on three different channels with different frequencies each advertising interval. This also enables an interesting option of choosing which channel's data to use for that particular iteration of the algorithm.

Algorithm 1 Positioning algorithm

```
procedure Positioning($numberOfNodesToUse, rssiSelectionMode$)
    while incoming data belongs to same interval do
        $newData \leftarrow$ ReceiveScanReport
        $aggregateNodedData[newData.nodeID].append(newData)$
    for all $node$ in $aggregateNodedData$ do
        for all unique $tag$ in node do
            $(rssi, channel) \leftarrow$ SelectRssiChannel($tag, rssiSelectionMode$)
            if RSSI Kalman filter is enabled then
                $rssi \leftarrow$ KalmanFilterRssi($nodeData.ID, tag.ID, rssi$)
            $node[tag.ID].distanceEstimate \leftarrow$ EstimateDistance($rssi, channel$)
            $nodes.append(node)$
    $nodes \leftarrow$ Sort($nodes, distances$)
    $estimatedPosition \leftarrow$ Multilateration($nodes[: numberOfNodesToUse]$)
    $position \leftarrow$ KalmanFilterPosition($estimatedPosition$)
    return $position$
```

When data is aggregated over an advertising interval or other logical frame, it's stored using the data source node's ID as key into a dictionary, hash table, associative array or other programming language native data structure with compatible functionality. When aggregation is finished, which means that all possible data from one interval is received, the algorithm proceeds to process the data. The aggregated data is iterated over per node, and per tag for each node in an inner loop. There, RSSI is selected according to the channel diversity scheme that's chosen: maximum RSSI, average RSSI or any other heuristic. Kalman filter is then optionally applied to the RSSI value, before the distance is estimated using either the raw or filtered RSSI value. Depending on implementation, the log-distance path loss model parameters may be dependent on the channel at which the RSSI was recorded. Lastly, the `multilateration` function is called with the node data is called. Only the node data for the $numberOfNodesToUse$ shortest estimated distances is used. Below is a closer explanation of all the function calls that are used in the positioning algorithm.

The function call ReceiveScanReport is an implementation specific function that receives one scan report at the time. In a real-time use-case this would be a function listening to a network socket and continuously receiving data from the nodes, parsing it and returning an implementation specific object with all the received data.

SelectRssiChannel is a function that dynamically selects which of the aggregated RSSI values that's going to be used and is described further in algorithm 2, 61.

KalmanFilterRssi and KalmanFilterPosition are function calls that apply Kalman filters to RSSI and position, respectively. The RSSI Kalman filter may be enabled or disabled by the implementation as it may introduce a delay in the system. The Kalman filter implementations are not described further, as this thesis has used an “off-the-shelf” Kalman filter solution for the filters and has not gone into details of this.

To estimate distances, the function EstimateDistance is called. This is an implementation of the log-distance path loss model presented in section 6.1.

Algorithm 2 RSSI selection algorithm

```
procedure SelectRssiChannel(tag, selectionMode)
    if $selectionMode = "max"$ then
        $index \leftarrow$ MaxIndex($tag.rssis$)
        $rssi \leftarrow tag.rssis[index]$
        $channel \leftarrow tag.channels[index]$
        return $(rssi, channel)$
    else if $selectionMode = "mean"$ then
        $rssi \leftarrow$ Mean($tag.rssis$)
        return $(rssi, Null)$
```

Lastly, to estimate a position, the multilateration function is called with the node data that will be used in the position estimation. This function uses non-linear least squares or equivalent to estimate the position.

Algorithm 3 Multilateration

```
procedure Multilateration(nodes, numberOfNodesToUse)
    $positionEstimate \leftarrow NonLinearLeastSquares(nodes.positions, nodes.distances)$
    return $positionEstimate$
```

5 | Test descriptions

Several tests have been conducted to examine various aspects of the indoor positioning platform. In addition to the tests described in this chapter, the qualitative practical functionality and usability of the positioning platform has been continuously evaluated during the tests. It's not a formal test per se, so it will not be described in this chapter, but the experiences are discussed in chapter 7.

The tests described in this chapter fall into three categories:

- RSSI - distance relationship
- Positioning of static tag
- Positioning and tracking of moving tag

Their setups and configurations are explained in this chapter. Common for all the tests are the following points:

- The radio mode is set to Bluetooth Low Energy, 1 Mbps PHY.
- The tags broadcast advertisements with access address `0xB9BED600` and an incrementing counter as the only payload.
- All tests have been conducted in the same room, without changes to furniture or other configuration, see fig. 5.1.
- If nothing else is mentioned, the nodes have been placed as shown in fig. 5.2. All wall distances and node placements were repeatedly measured with laser measurer with an accuracy of ±1.5 mm. Due to existing infrastructure and equipment in the ceiling, the nodes had to be placed as they are in the figure. A more even spread of the nodes could be more optimal, but in a real deployment, it's also unlikely to be able to place the nodes in a perfect grid.
- The tags in all tests have been PCA10056 nRF52840 Development Kit if otherwise not specified.

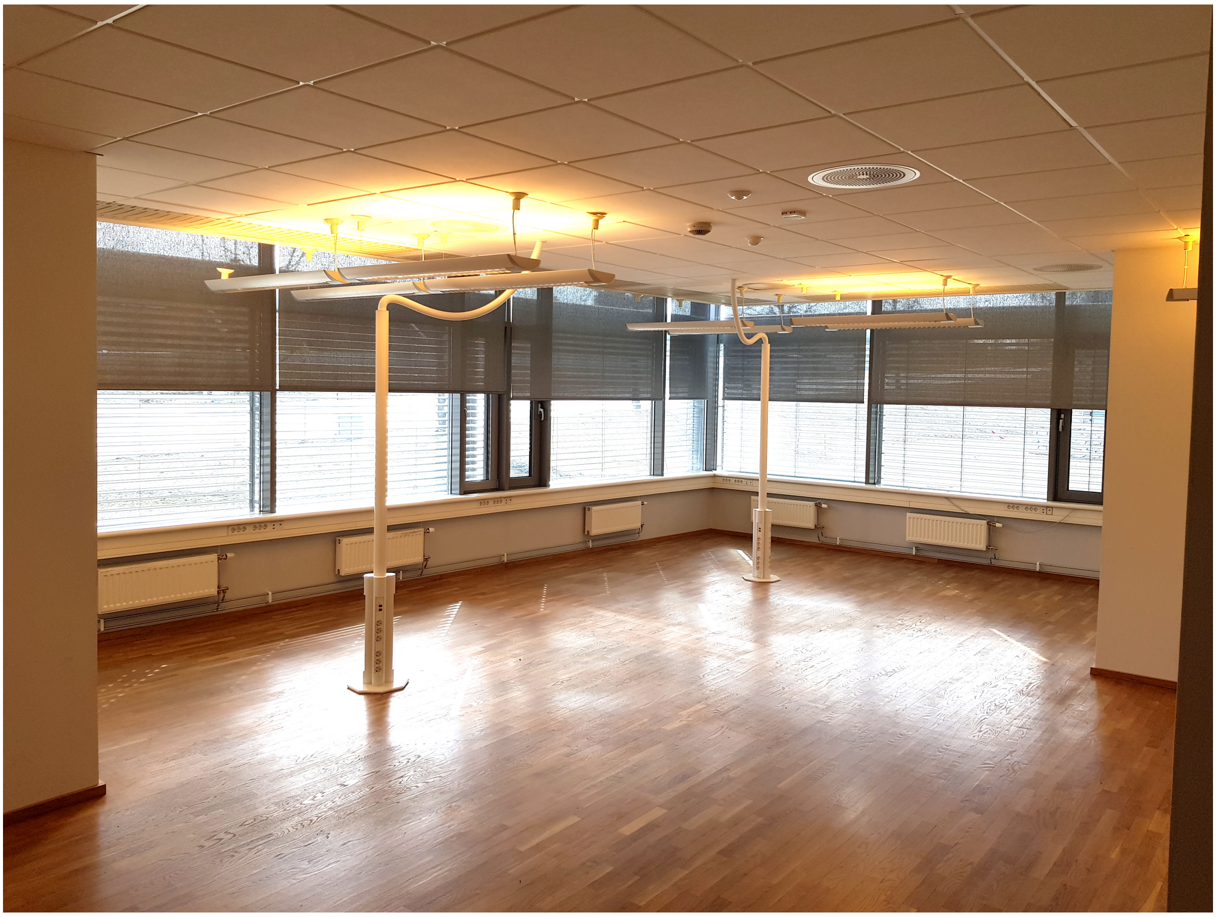


Figure 5.1: The room where the platform was tested.

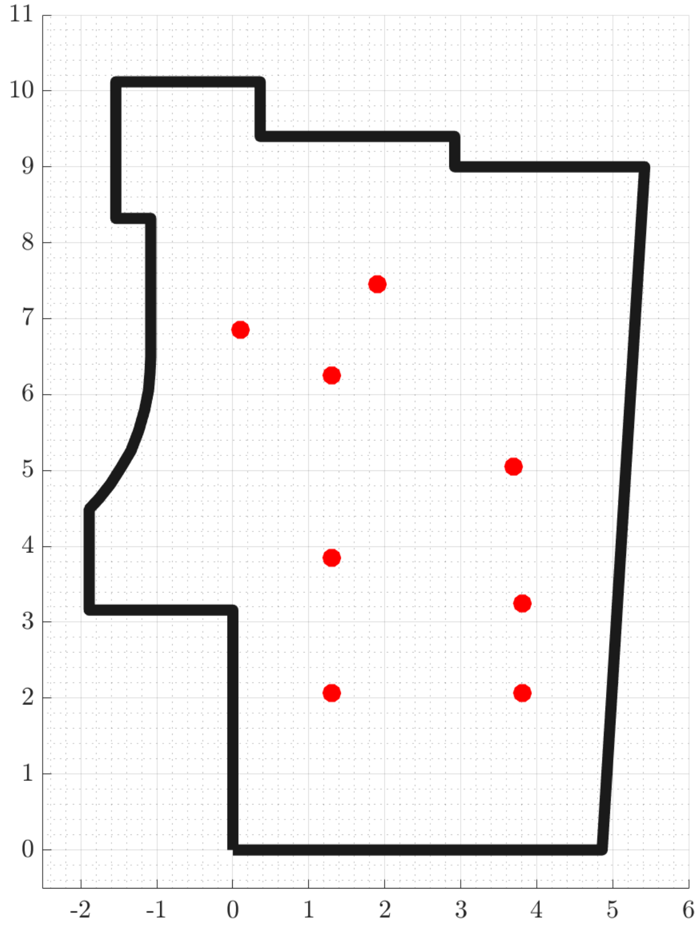


Figure 5.2: Node placement in the testing area. Red dots are nodes, black lines are walls. The x- and y-axis are in units of meters. All nodes are placed at the same height, 2.70 m.

5.1 Testing RSSI and distance relationship

Testing of the relationship between RSSI values and distances has been done to establish the parameters to be used in log-distance path loss model (eq. (2.1)) in the positioning algorithm. RSSI is the only measurement information the algorithms has to estimate distances and hence positions, so it's important to find a good estimate for the model parameters. The following test has been done to examine the RSSI - distance relationship:

- Measure RSSI having **free line-of-sight** between sender and receiver at incrementally increasing distance. Performed with multiple both senders and receivers to possibly discover differences between devices. Tested with sender and receiver on the same vertical plane, elevated 75cm from the floor.

The testing procedure was to record RSSI values for at least 60 seconds for each distance. Advertising interval for the tags was set to 200ms and the tag advertised on all three BLE advertising channels, which results in a total of at least 900 data points for each distance. The test was started at 0.5m and increased up to 10.5m in 0.5m increments. All distances were measured with laser distance measure with an error of ±1.5mm. Figure 5.3 shows how the node in the test was set up.

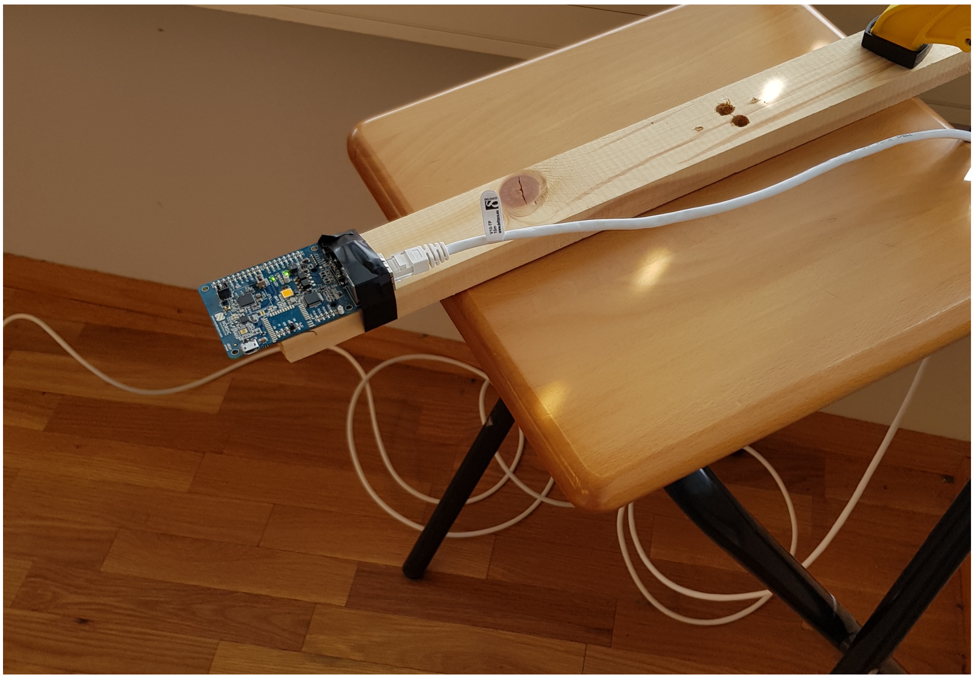


Figure 5.3: Test setup RSSI - distance relationship. The node is elevated 75cm. The tag is set up in the same way, both with antennas in free air, and free line of sight.

5.2 Positioning of static object

Testing is done by attaching 8 nodes to the ceiling at a height of 2.70m. The nodes are attached to existing rails using cable ties, with a 2cm spacer from the ceiling, as can be seen in fig. 5.4. All nodes were positioned the same way and fastened to the ceiling in the same way. After installation, all nodes' position were measured with laser distance measure and plotted into the coordinate system seen in fig. 5.2 using a Python script written for this purpose. It worked by turning on the high-power LED on only one of the nodes, wait for someone to plot in the x, y and z coordinates for the node before saving this into a configuration file on the computer. The same node configuration file could then be used for all tests as they were never moved. The file name for the configuration file was stored into database along with the scan reports and other information as described in section 3.4.3.

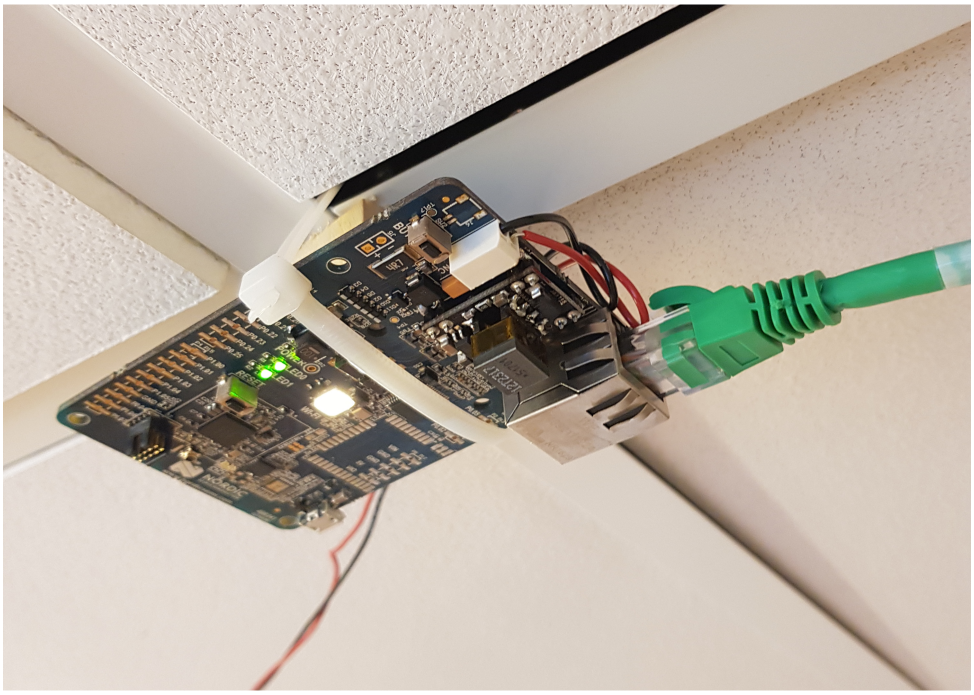


Figure 5.4: Nodes were attached to the ceiling with cable tie, with a 2cm wooden spacer between the PCB and the ceiling. Time synchronization line and common ground (red and black wire, respectively) were daisy-chained to all nodes.

All nodes were connected to the same PoE network switch, Cisco Catalyst 3560-CG, with Category 6 cables. The time synchronization lines were daisy-chained to each other and controlled by one of the nodes.
Performed tests with static tags are as follows:

- Positioning of stationary tag at multiple known positions. Tested in a grid 1.0m by 0.5m (x by y) sized grid covering the entire testing area.
- Self-positioning: testing of a node advertising from its location attached to the ceiling

The tests are all performed in the same environment with no changes to the surroundings. They have been conducted within a few hours, so it's likely that there should be similar interference from other devices on all tests.

5.3 Tracking of moving object

Tests of moving objects have been done using the same node setup as for tests on static objects described in section 5.2 and shown in fig. 5.2. Testing was done with a tag in four different configurations:

- Attached to a stand, with the antenna vertical, pointing up in 85cm height and being pulled by a rope.
- The aforementioned stand held by a person with the tag at 180 cm height and walking speed of about 1.2 m/s.
- Attached to a person's pocket (shown in fig. 5.5), emulating a key card or other device that can potentially be used as a natural tag.
- Hanging around a person's neck on a person's chest, emulating a device that may be carried by a person in need of special care and used for tracking.

The paths used for pulled stand with vertical tag were as follows, given in coordinate tuples (x, y):

- From $(1.8, 1.0)$ to $(1.8, 8.4)$
- From $(4.0, 1.0)$ to $(0.0, 8.4)$

When attached to the body, the following paths were also tested:

- From $(0.0, 8.4)$ to $(4.0, 1.0)$
- From $(0.0, 8.4)$ via $(4.0, 1.0)$ and $(1.8, 1.0)$ to $(1.8, 8.4)$
- From $(4.0, 7.5)$ via $(4.0, 1.0)$ and $(1.8, 1.0)$ to $(0.0, 8.4)$

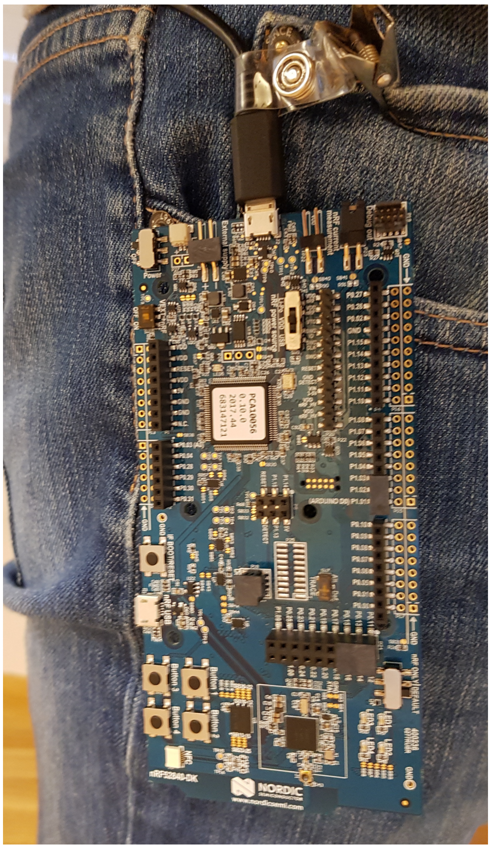


Figure 5.5: Tag attached to person during test on tracking moving tag.

6 | Results

This chapter contains the results from the tests described in chapter 5. First, results from the RSSI - distance relationship tests are presented. These also involve determination of parameters for the log-distance path loss model that is used to estimate distances from each node in the positioning algorithm. Next follow results from static tag positioning tests, and finally results from tracking of moving tag are presented.

6.1 RSSI - distance test results

RSSI values have been recorded for distances from 0.5m to 10.5m in 0.5m increments along a straight line. The RSSI values from all these tests are shown in fig. 6.1, where each channel's RSSI values are shown separately. The RSSI values for each channel tend to be grouped together within 8 to 10 dB difference, but there are also more extreme values. Looking at the readings for 2m distance, the RSSI values for all channels span from a minimum of -62 dBm to a maximum of -37 dBm, a difference of 25 dBm. We'll later see that this corresponds to a difference in distance estimate of 21.5m. It's also visible from the plot that it wouldn't be surprising to see an RSSI of -50 dBm at any distance between 2 and 10 meters.

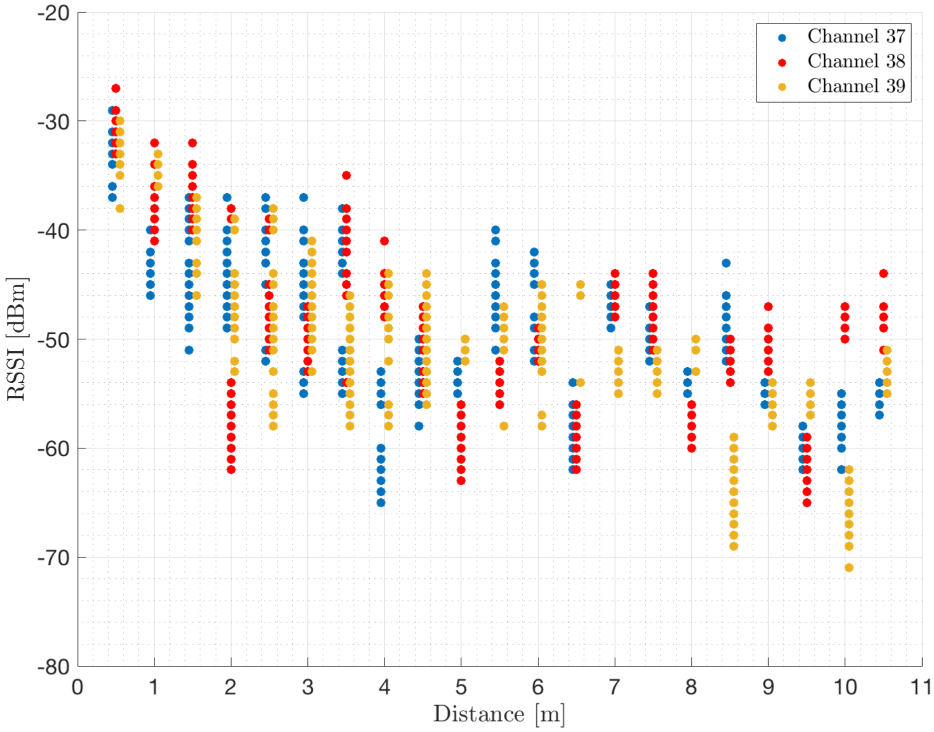


Figure 6.1: RSSI values plotted for distances from 0.5m to 10.5m. The different channels are slightly spaced away from the true distance to better visualize the differences.

Looking at the same data with a logarithmic x-axis in fig. 6.2, it's more clear also visually that the distance increases with decreasing RSSI values. The best log-distance path loss model is also plotted in the same figure. The parameters of $RSSI_{d_0} = -38.0$ and $n = 1.78$ for the model were found using Matlab's `nlinfit`. $RSSI_{d_0}$ is defined as the RSSI value at 1.0 meter distance, and the same value of -38.0 was found both using `nlinfit` and by calculating the average value of all measurements at 1.0m. A value of $n = 1.78$ for the path loss exponent is in line with the expected values according to section 2.1.2.1.

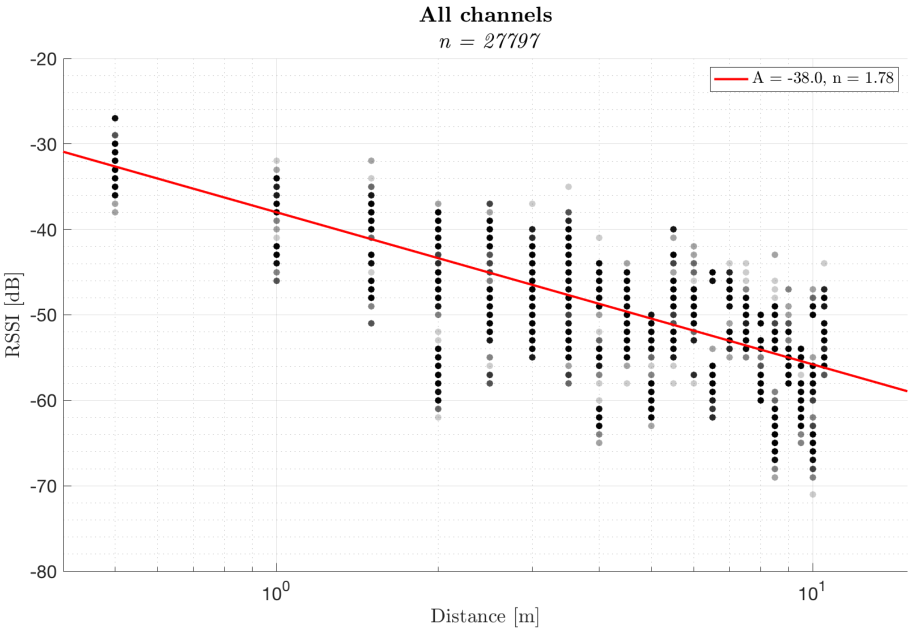


Figure 6.2: RSSI values plotted for distances from 0.5m to 10.5m with logarithmic x-axis together with best fit log-distance path loss model line with parameters $RSSI_{d_0} = -38.0, n = 1.78$.

Figure 6.3 shows RSSI value plots sorted by advertising channels, maximum RSSI and average RSSI per advertising interval. All plots do also have a line showing how the log-distance path loss model looks like for the estimated parameters for each of the RSSI value sets. The parameters are estimated by using both Matlab's `nlinfit` and `lsqcurvefit` functions, which give the same results. The figures show the Root Mean Square Errors for the fitted curves as well. As visible in the plots, both max and average values have lower RMSE than each channel individually. This is possibly an indication that using one of the two will result in more accurate distance estimates, and hence more accurate positioning estimates when used in the positioning algorithm.

Figure 6.4 shows a similar approach, but here a fixed value has been used for $RSSI_{d_0}$, so only n is fitted using `nlinfit`. Instead, $RSSI_{d_0}$ is as mentioned in a previous paragraph found to be -38.0 dBm. As seen in the figure, the least RMSE is for the average RSSI value per advertising channel using this approach, which is also the most common approach in the literature.

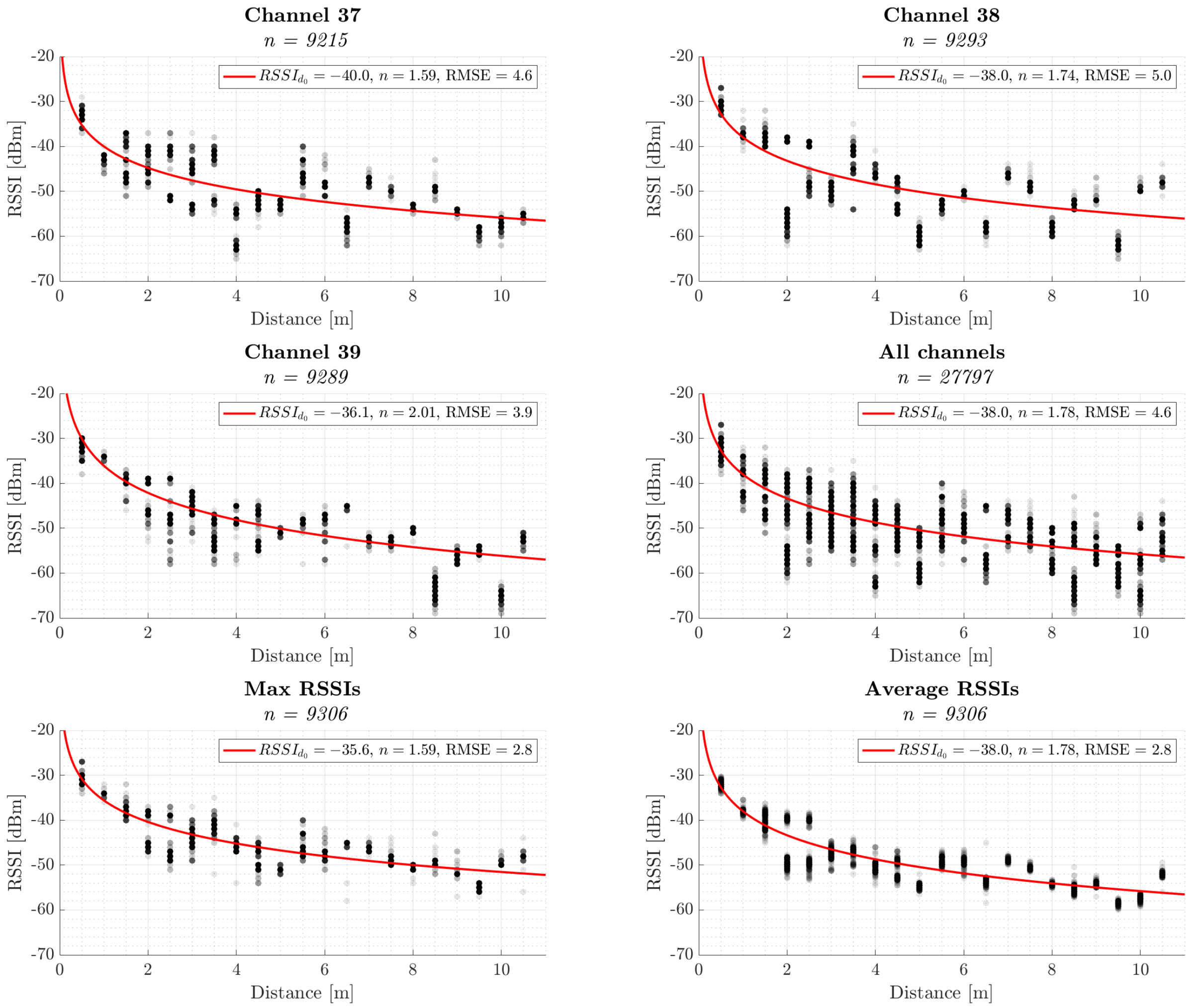


Figure 6.3: RSSI values and estimated parameters for log-distance path loss model plotted for distances from $0.5m$ to $10.5m$ for each individual advertising channel, for the maximum RSSI among the three channels per advertising interval and average RSSI for each advertising interval. Both $RSSI_{d_0}$ and n have been fitted here, using Matlab's `nlinfit`.

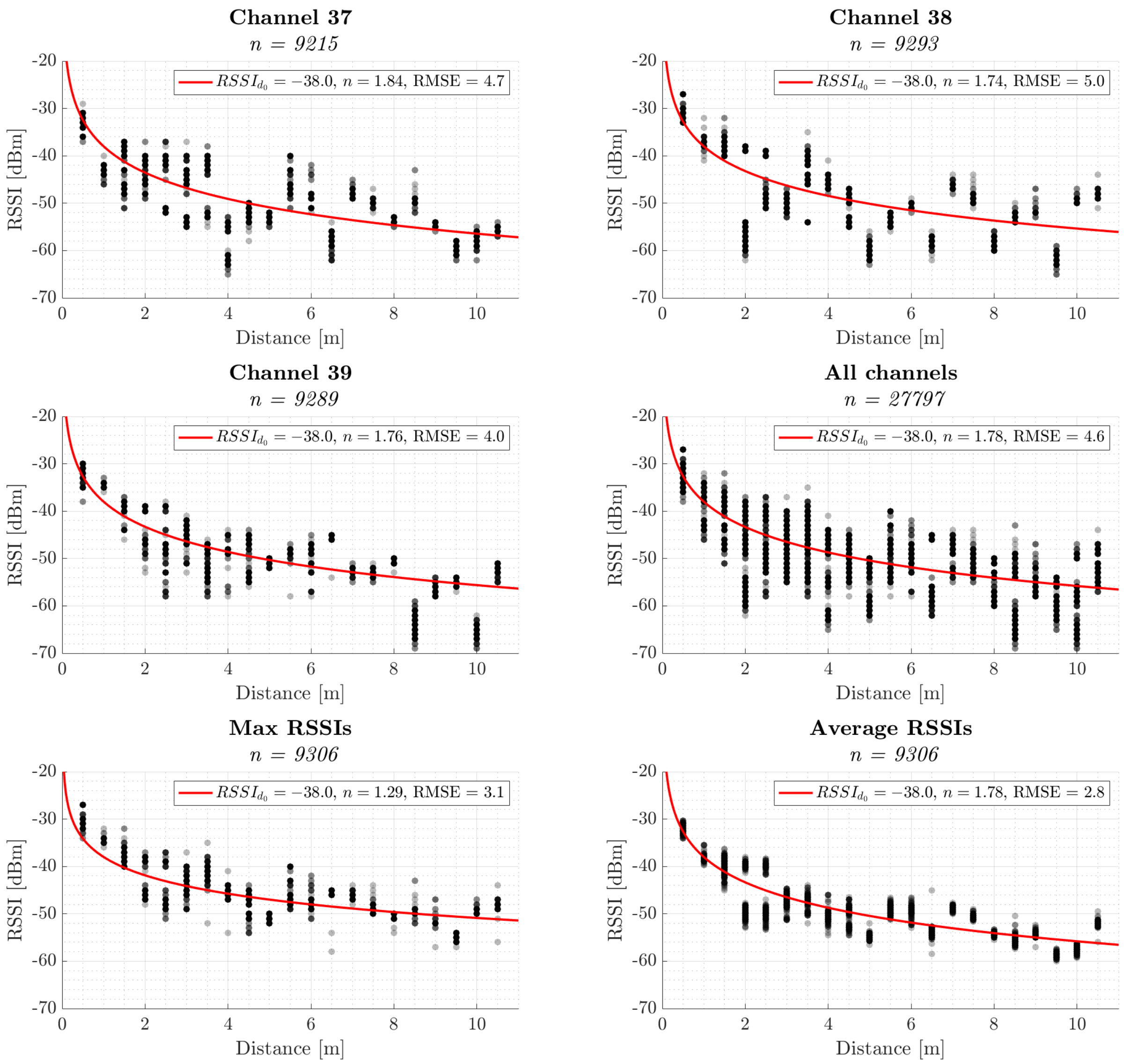


Figure 6.4: RSSI values and estimated parameter n in log-distance path loss model plotted for distances from $0.5m$ to $10.5m$ for each individual advertising channel, for the maximum RSSI among the three channels per advertising interval and average RSSI for each advertising interval.

Figure 6.5 shows RSSI - distance plots for calculated Euclidean distances from the positioning tests, where the nodes are placed in the ceiling and the tags on a lower height. Only max ans average values per advertising interval is shown here. For both the RMSE is higher than in the previously shown RSSI - distance plots, indicating that the RSSI values, unsurprisingly, vary more in a real platform deployment.

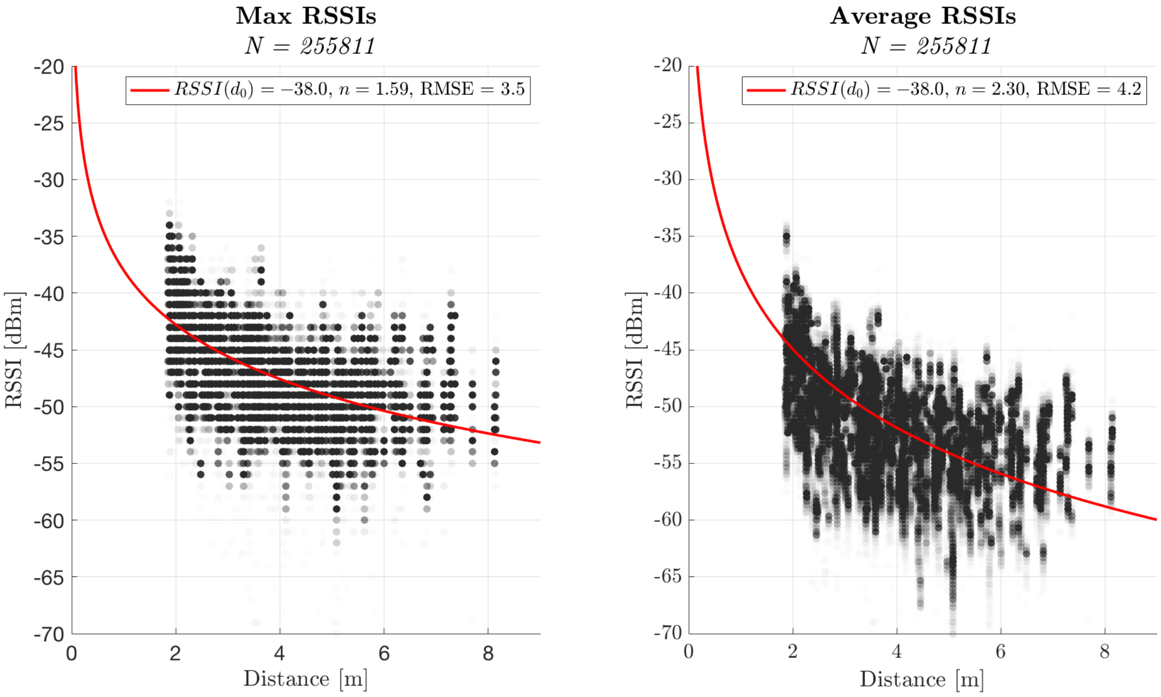


Figure 6.5: RSSI values plotted for calculated true distances between nods and tags for all positioning tests. Matlab's `nlinfit` used to estimate parameter $n = 1.59$ for log-distance path loss model. Resulting log-distance path loss model plotted in red using parameters $RSSI_{d_0} = -38.0$, $n = 1.59$.

6.2 Static tag positioning test results

For the positioning tests with static tags, there are two main areas of interest: the error between estimated tag position and the true tag position calculated as the Euclidean distance, and how long it takes before the platform can deliver a stable position estimate. The positioning algorithm shown in algorithm 1, page 60 has been applied to data sets stored in MySQL databases using Python programs described in section 3.4. All data from running the positioning algorithm has been stored in CSV files and further analyzed using Matlab R2018a.

Where nothing else is mentioned, the following configurations were used for the positioning algorithm:

- Log-distance path loss model, see eq. (2.1), using the max RSSI every advertising interval out of the ≤ 3 available channels if nothing else is specified
  - $RSSI_{d_0} = -38.0$
  - $n = 1.78$

- Kalman filter for RSSI with following configuration

$$Q = 0.65, R = 3.19 \tag{6.1}$$

- Using all available nodes, 8, in multilateration
- Kalman filter for positioning with 6 states, position and velocity in x-, y- and z-axes, with matrices as explained in section 2.4. The matrices are populated with the same values as shown in section 3.4.5.

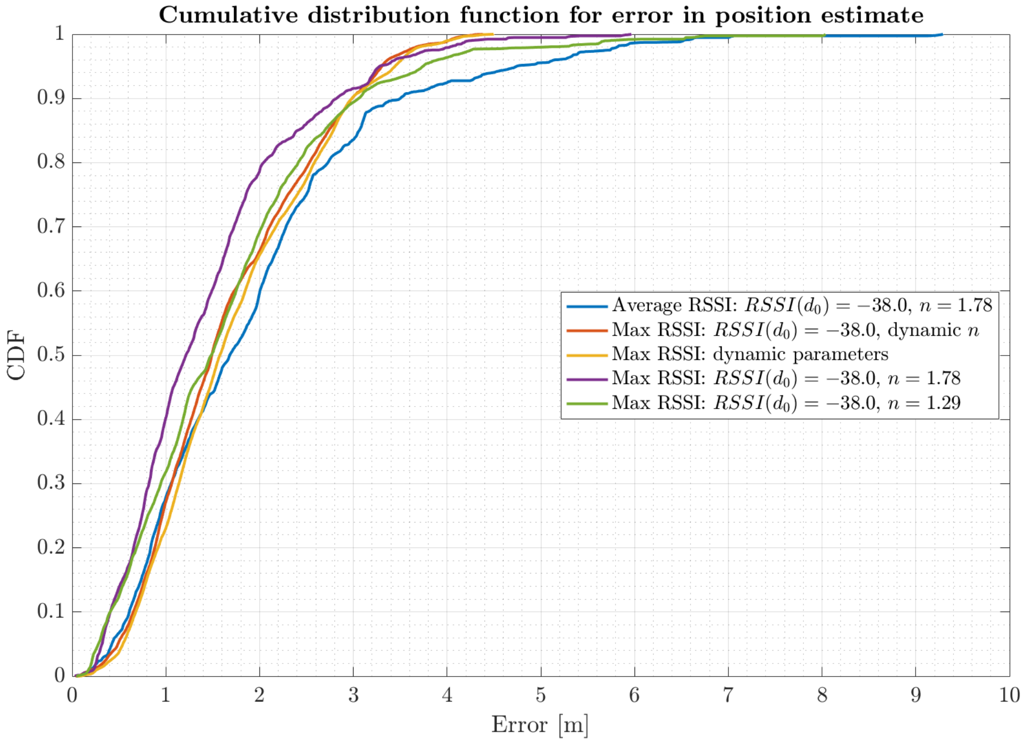


Figure 6.6: Cumulative distribution function for position estimates, with N = 418. Tags placed at height of 85cm in a grid with cell size 1.0m by 0.5m in the entire test room, rotated to point north, east, south and west relative to grid coordinate system at each spot.

Figure 6.6 shows a cumulative distribution function plot for 2D errors in position estimates with $N = 418$ distinct tests. The plotted functions tell how large proportion of the estimated positions that have an error of x or less. The errors are calculated from running the positioning algorithm on 447 different scenarios. Different log-distance path loss model parameters have been used to compare, based on the information presented in section 6.1. In addition, two dynamic approaches are tested. These are dynamic in the sense that they decide which log-distance parameters to use after the channel selection is done. One is based on the $RSSI_{d_0}$ and N estimates found seen in fig. 6.3, and the other is based on fig. 6.4 with statically set $RSSI_{d_0} = -38.0$ as is the mean of all RSSI measurements at $d_0 = 1.0$m. It's evident from the plot that using maximum RSSI per node per tag each advertising interval yields the best estimates for position.

Section 6.2 shows some statistical properties of the cumulative error functions. It's notable that the dynamic approaches have steeper curves and lower maximum errors with maximum of 1.5 meters lower than the static approaches. The best performing combination is max channel selection, $RSSI_{d_0} = -38.0$ and $n = 1.78$. This combination gives a mean error of 1.44m, and a median of 1.21m, with a standard deviation of 0.96m. Curiously it has a significantly larger maximum error than the dynamic channel selections.

	Min	**Max**	**Mean**	**Median**	**Std. dev.**
Max RSSI: $RSSI_{d_0} = -38.0$**,** $n = 1.78$	0.04	5.96	1.44	1.21	0.96
Max RSSI: dynamic $RSSI_{d_0}$ **and** n	0.06	4.49	1.74	1.58	0.90
Max RSSI: $RSSI_{d_0} = -38.0$**, dynamic** n	0.03	4.38	1.67	1.49	0.89
Max RSSI: $RSSI_{d_0} = -38.0$**,** $n = 1.29$	0.04	8.04	1.66	1.48	1.15
Average RSSI: $RSSI_{d_0} = -38.0$**,** $n = 1.78$	0.01	9.29	2.02	1.82	1.35

Table 6.1: Statistical properties for different log-distance path loss model parameters used in the positioning algorithm. $N = 418$.

In the subsequent tests, max RSSI is selected each advertising interval, and parameters used in log-distance path loss model are $RSSI_{d_0} = -38.0$ and $n = 1.78$ if nothing else is explicitly stated.

To be able to locate a device in a short amount of time is useful, and sometimes an important aspect of the use-case. As the positioning algorithm can only run another iteration and improve its position estimate every time it gets new information about a tag, the most interesting area is to look at how the accuracy of the system is affected by the number of advertising intervals it records a tag. A tag might broadcast once every second, every four seconds or five times a second. Therefore the time aspect is not as relevant as the number of advertising intervals and iterations the algorithm can run.

Figure 6.7 shows cumulative error functions for increasing number of advertising intervals and thereby algorithm iterations. Static tags are used, and the algorithm has no information about the tag before first iteration. This means that all Kalman filters are using initial values with high variance, and it's therefore expected that the position estimate uses some time to converge to its asymptotic error at each position. Note that the asymptotic error may be closer to the true tag position, but in some cases it's also farther from the true position, thus increasing the error for that particular position as number of iterations increases. Results from 302 position test were used to compile the plot.

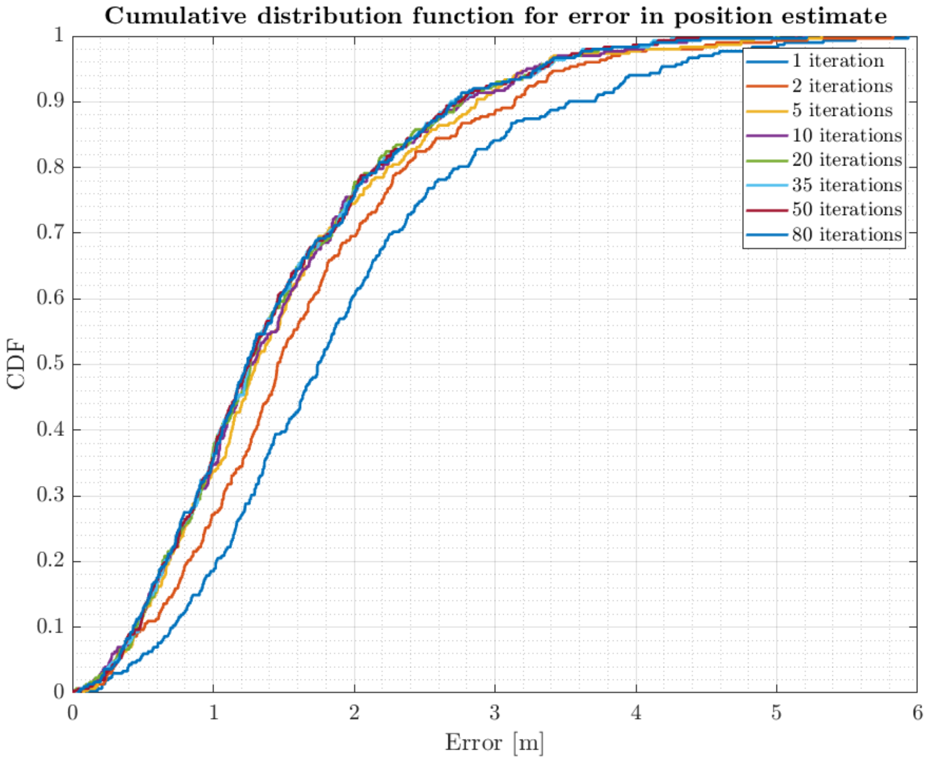


Figure 6.7: Cumulative distribution function for position estimates using an increasing number of advertising intervals allowing the same number of positioning algorithm iterations. 302 distinct tests with at least 80 iterations each are used to compile the plot.

There's a decrease in error as the number of iterations goes up. Figure 6.8 shows how the mean and median errors decrease as the number of algorithm iterations goes up. There's a relatively sharp decrease in error for the first iterations, but slope rapidly flattens. The mean estimate error can be reduced by 45.6cm and the median 44.1cm by increasing the iteration count from 1 to 80 iterations. However, as the figure shows, there's little gain in increasing to more than 5 intervals to have a stable estimate. Figure 6.9 shows the potential percentage in reduced error that can be achieved by increasing the iteration count. By increasing from 1 to 5 iterations, the estimate is in the average and median case has been reduced by 22% and 25%, respectively, and there's only 3% and 5% more to be gained by letting the algorithm run for more iterations.

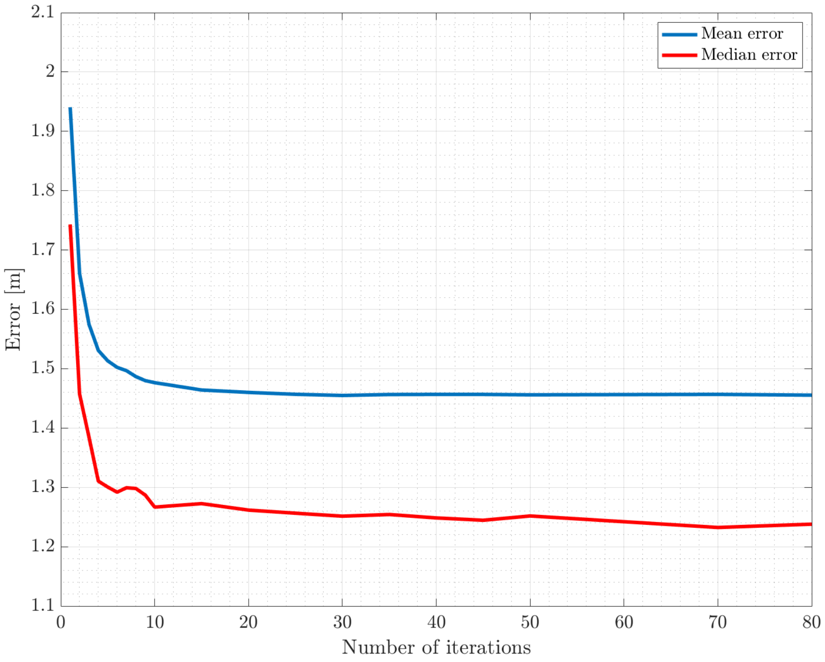


Figure 6.8: Mean and median errors for position estimates using positioning algorithm iterations in the range from 1 to 80.

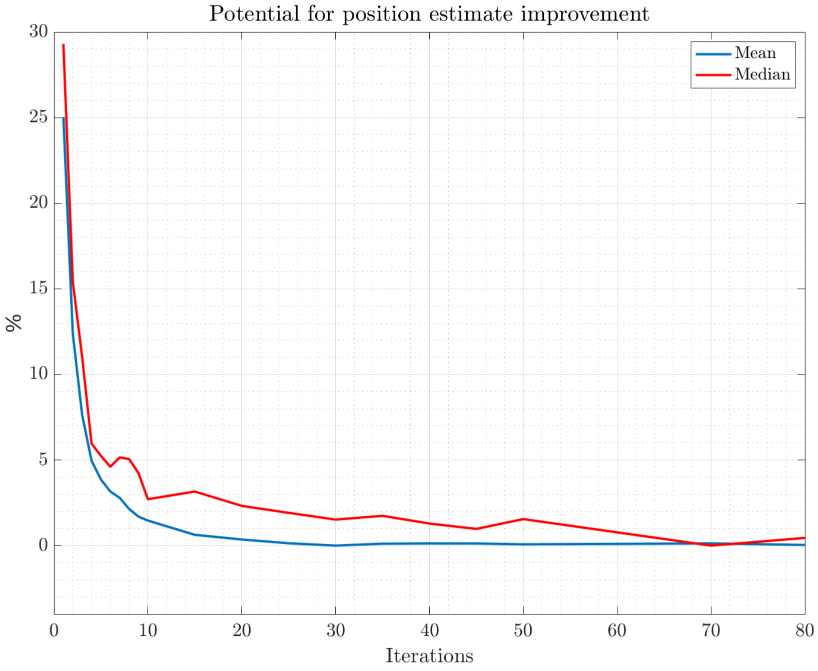


Figure 6.9: Potential for improvement for mean and median errors for position estimates using positioning algorithm iterations in the range from 1 to 80.

Testing of so-called self-positioning has also been conducted. Self-positioning is in this thesis defined as the procedure of a node acting as a tag and advertising, while the other nodes function as regular nodes. Positioning algorithm is then applied to the collected data to estimate the position of the advertising node. The cumulative error distribution for these tests can be seen in fig. 6.10, 79% of the position estimates have en error of 1.02m or less and 50% are within 75cm. In practice, this means that 7 out of the 8 nodes in total were estimated to be within 1 meter of their true position.

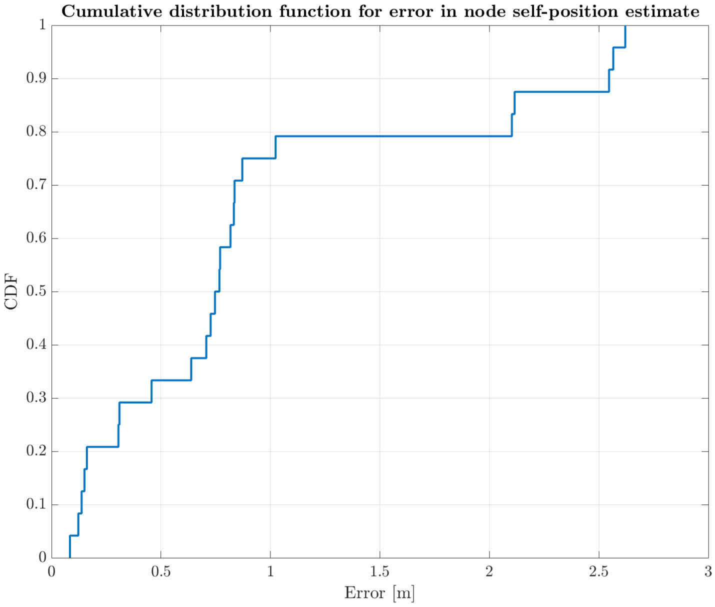


Figure 6.10: Cumulative error distribution for self-positioning.

6.3 Tracking of moving tag

Tracking of moving tags using RSSIs is a challenging task. The orientation of the board and thereby antenna may affect the result. Another major impact factor is the presence of persons near the tag. Human bodies effectively absorb radio waves, and also act as a reflector to a certain extent. This will impact the RSSI readings, and by extension the positioning estimates. Figure 6.11 shows an example of tracking a tag held in front of a person in 1.8 meter height, walking in a straight line illustrated by the yellow arrow with normal walking pace of about 1.2 m/s. Three different paths of position estimates are shown: based on raw RSSI values, filtered RSSI values and the combination of RSSI and position filtering. Raw RSSI yields very noisy estimates and the RSSI filter provides a clear improvement. Filtered RSSI and position is clearly the superior method. Although the maximum deviation of nearly 1 meter from the true path is pretty equal for all methods, the variance is very much improved using the position RSSI filter. The estimated path interestingly seems to follow the curvature of the left wall. The same trend is seen on most of the other recordings of the exact same path, but not all. The sample size is too small to say anything conclusive about this effect, but the tendency is noteworthy.

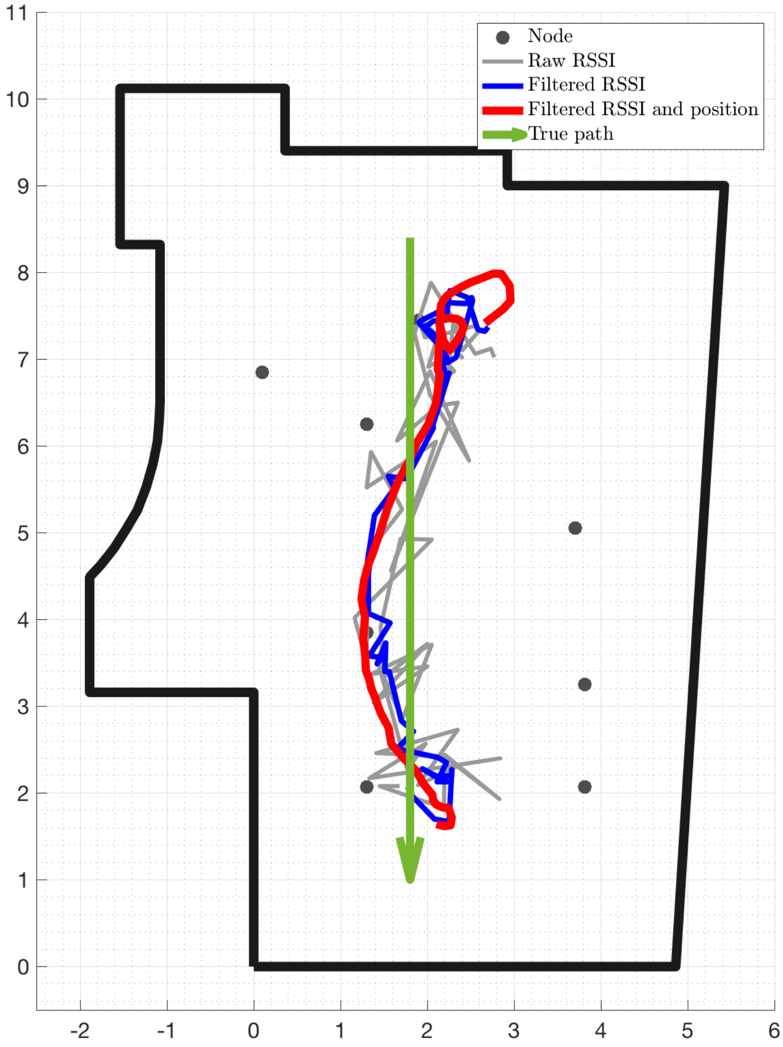


Figure 6.11: Tracking of moving tag. The plot shows the paths for position estimates using raw RSSI, filtered RSSI and the combination of filtered RSSI and filtered position. The tag is attached to a stand carried by a person holding it at 1.8 meter height, 1.5 meters from the body.

Figure 6.12 shows a similar scenario to the above, but where the tag is placed on a 85 cm high stand being pulled at walking speed, so there is no body nearby potentially hindering line of sight for the nodes. It's visible that instead of undershooting the position estimate in both ends of the path, it's now overshooting the position estimates relative to the true positions. Looking at the z-coordinate estimates, which is the height, in fig. 6.13, we can form a hypothesis as to why this is the case. The tag is in reality placed at 0.85 meter height, but towards both ends of the position is estimated to be at about 2 meters height. This undershoot in height relative to the nodes placed at 2.70 meters, might lead to an undesired overshoot in other axes, y in this case.

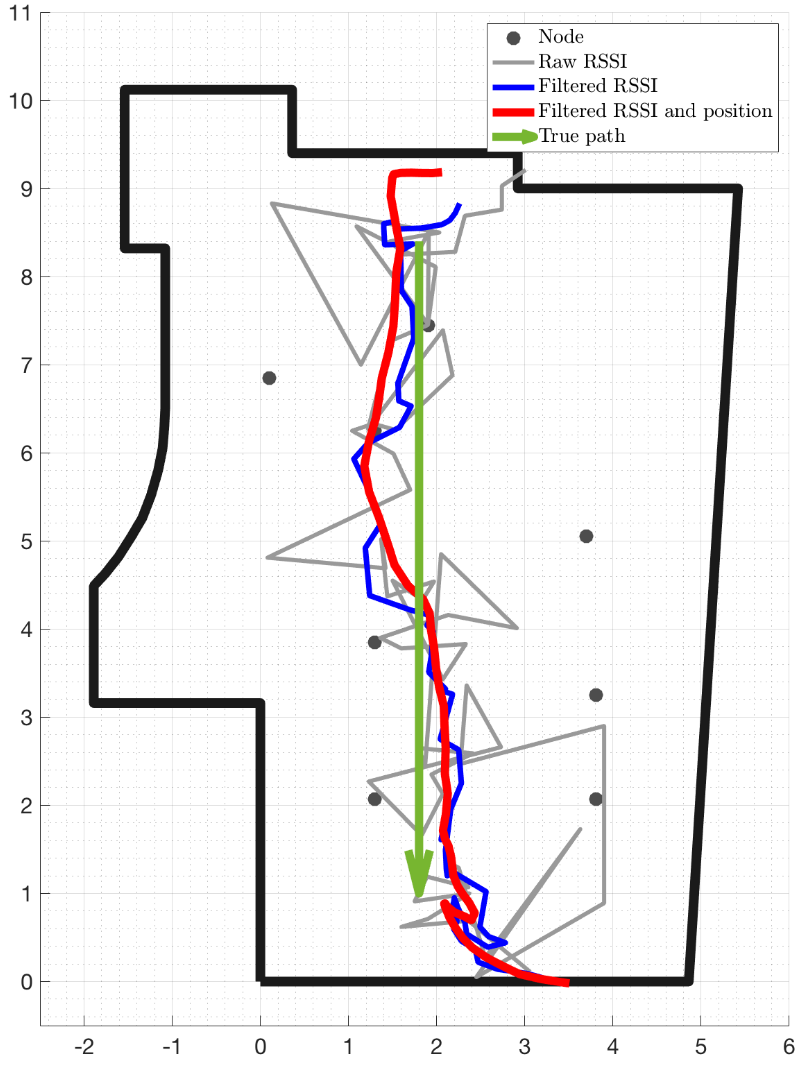


Figure 6.12: Tracking of tag attached to an 85 cm high stand being pulled along the noted path.

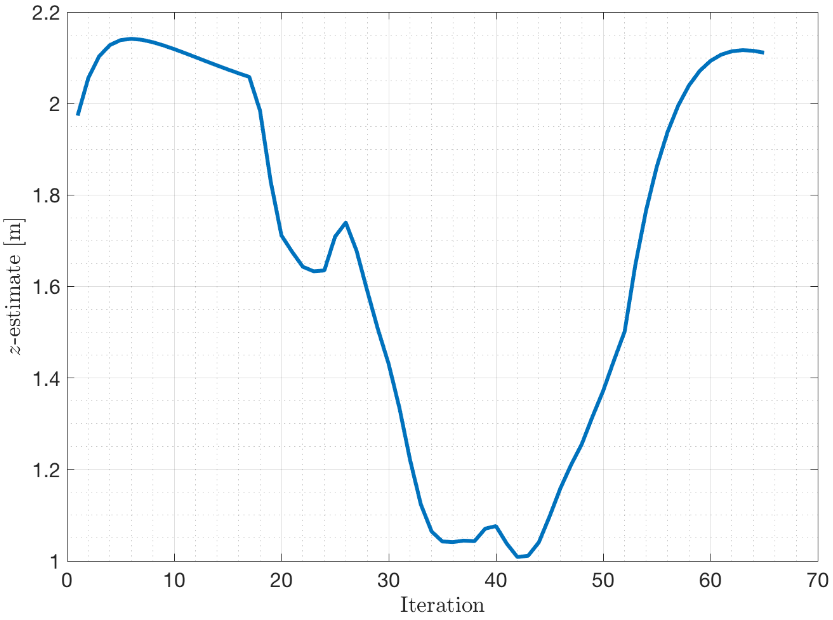


Figure 6.13: Z-position estimate for path estimate in fig. 6.12

Figure 6.14 shows a more complex path where the tag's orientation changed two times relative to the test site's coordinate system. It was carried vertically, with the antenna upwards and the top side of the board pointing ahead of the person carrying it, and therefore changed orientation relative to the coordinate system at each turn. The figure shows that the path estimate is quite close to the truth, but the error increases towards the end.

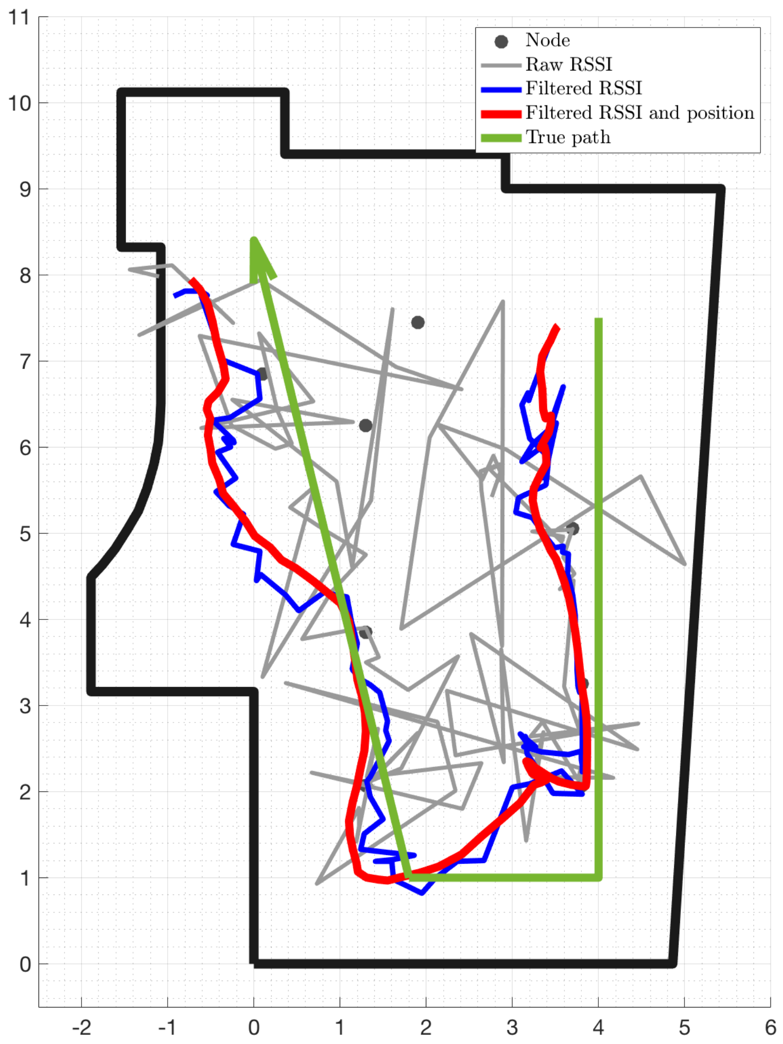


Figure 6.14: Path estimate for tag carried on a stand at 1.8 meter height. The figure shows that the position estimate at the worst is around 1.1 meter off from the true position.

Figure 6.15 shows the estimated paths of a person walking with a tag attached to the right pocket, as shown in fig. 5.5. The true path is again shown in yellow, and it's clear that the path estimate is skewed to the right of the true position. This is perhaps what one would expect, as the nodes to the left of the person will likely overshoot the distance estimates to the tag because there's no line of sight to the tag and therefore the RSSI will be negatively affected.

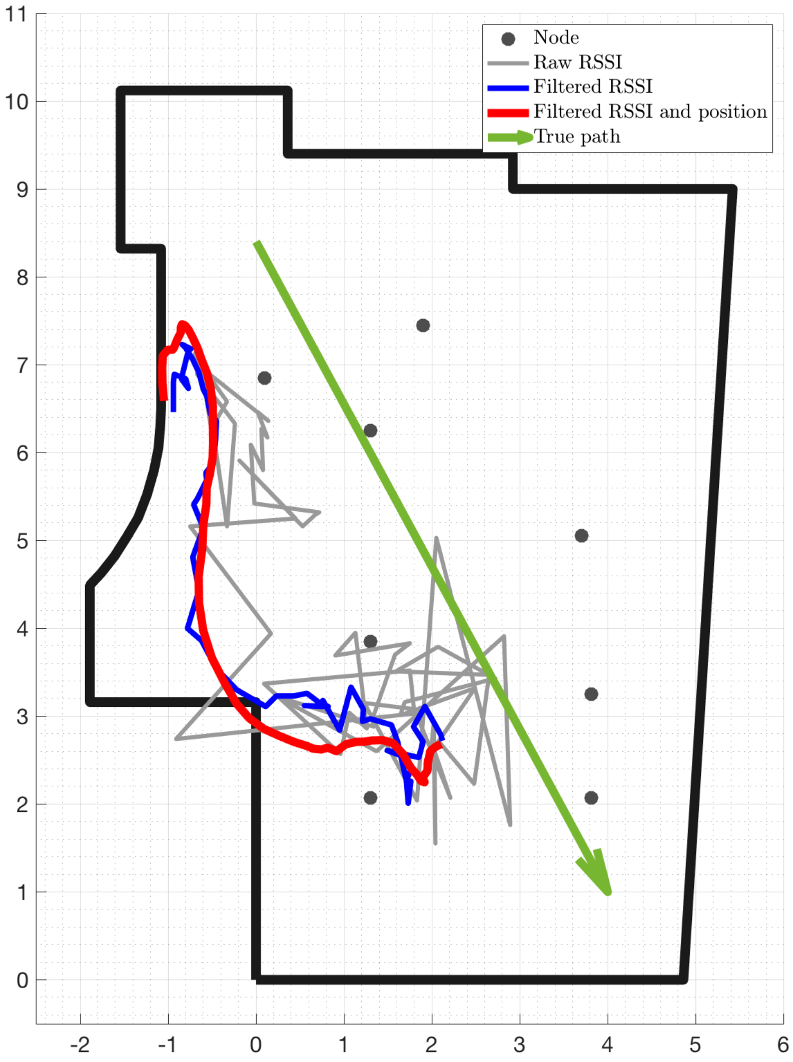


Figure 6.15: Tracking of tag attached to a person's pocket as shown in fig. 5.5.

An example of resulting path estimate for a tag hanging on the chest is shown in fig. 6.16. A similar effect as in fig. 6.15 is seen here. The body is now blocking line of sight for nodes directly behind the person carrying the tag, resulting in an overshoot in the estimate in the opposite direction. The direction of the estimate is however pretty accurate, suggesting that the nodes on the sides of the tag record normal RSSI values, or possibly equally and symmetrically affected and mostly altering the z-axis estimate not shown in the plot.

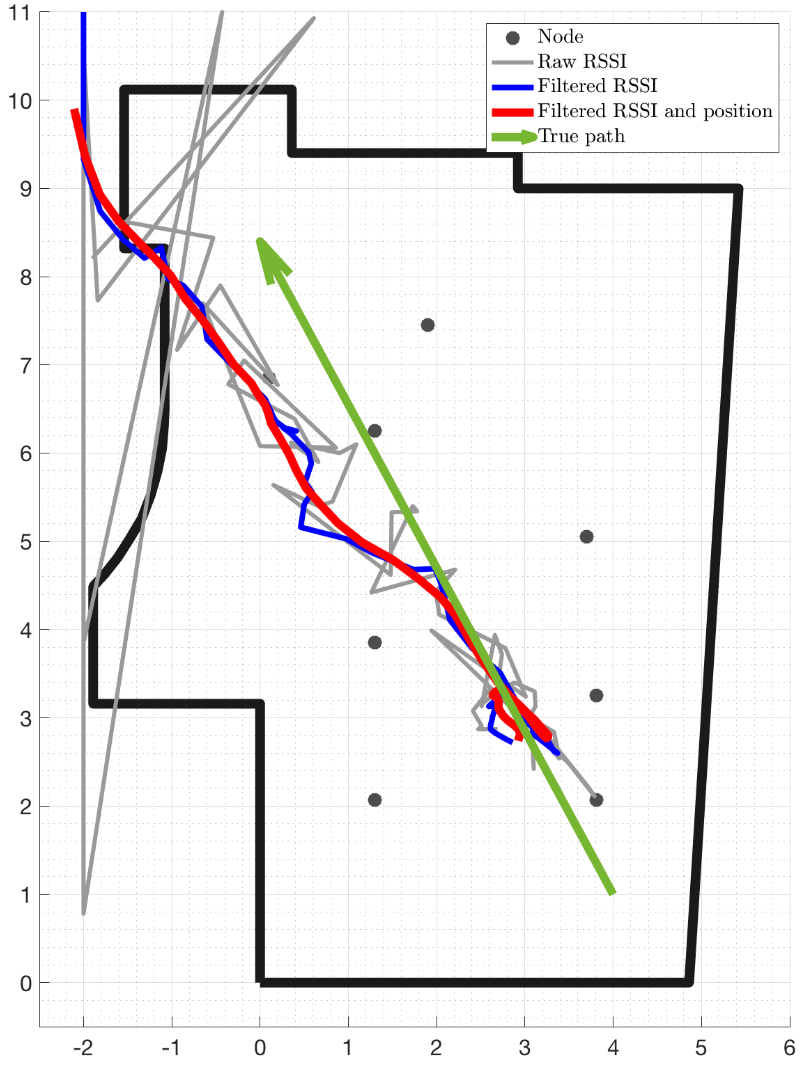


Figure 6.16: Path estimate for tag hanging on the chest. Note how the position estimate based on raw RSSI varies wildly as the person approaches the end of the path and all nodes are blocked from line of sight by the person's body, but the estimates based on the two different filter variants are less affected by the variations.

Figure 6.17 shows a path estimate for a tag attached to a person's pocket (as seen in fig. 5.5). It's evident that the estimates are less accurate when carrying the tag close to the body. The figure also shows how important the the Kalman filtering of both the RSSI and position is when the tag is close to the body. The path based on raw RSSI is very noisy, and without filtering, it would be hard to imagine where the person actually walked by looking at the path estimated by the raw RSSI data. The filtered paths are not good estimates, either, but they do come closer to the truth. Note that the tag is always placed on the right side of the body and the person is always walking straight ahead along the path lines, turning right when needed. Again there's a clear tendency that the path is skewed towards the tag side of the body, both when waling diagonally, when turning and walking parallel to the x-axis and when walking parallel to the y-axis.

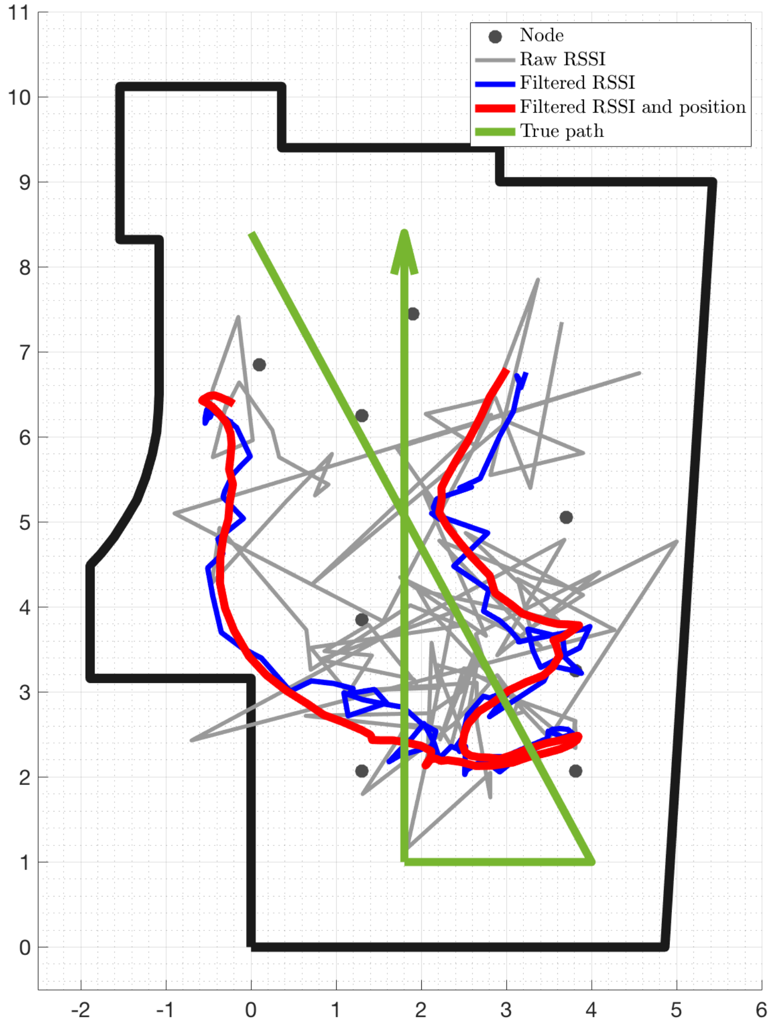


Figure 6.17: Path estimate for tag attached to a person's right pocket. The estimated paths start at the left hand side of the figure.

To be able to create cumulative error plots like those for static tags and more sophisticated statistics for moving tags, one would need strict timing of the movements, and the golden standard in this research field is using a remotely controlled vehicle. This has regrettably not been within the scope of this thesis.

7 | Evaluation and discussion

This chapter first evaluates the indoor positioning platform developed in the thesis, and then discusses the results and findings in the tests and analyses that have been conducted and presented in chapter 5 and chapter 6.

7.1 Platform evaluation

The indoor positioning platform has been tested both during development and under the formalized tests described in chapter 5. It's proven to "do its job" in terms of collecting data about tags that are being tracked and sending scan reports to the back-end computer in the desired format. It's not been performance tested and analyzed other than through normal usage. In this section, the hardware, microcontroller firmware and back-end software will be discussed with a qualitative approach, weighing back and forth how the choices during development have turned out and identifying potential for improvements.

7.1.1 Hardware

The platform hardware consists of the nodes, PoE network switch, back-end computer and the tags. The tags and back-end computer hardware has not been specified to be specific devices, but rather defined to fill a role. The tags have in this theses been dedicated development kits programmed with custom firmware for full control over their functionality. However, the same functionality could have been obtained by using a mobile phone with open-source software. Therefore, they will not be discussed further in the hardware section.

PoE network Switch

The PoE switch has functioned as intended, and powered all the nodes without any interruptions. It's also served as expected of a DHCP server and IP router, with noe remarkable events happening during testing or development. As there are no advanced or device specific configurations enabled in the switch, it's expected that the simple configuration described in section 3.2.3 would also be transferable to other switches in a larger scale implementation.

Anchor nodes

The node hardware has worked as intended during the testing, after a few issues were resolved during development. There is, however, room for improvement. Most important is that the antenna is a PCB antenna that seemingly does not record equal RSSI values when the board is rotated. Tests are needed to verify this, but empirically, a board pointing the antenna towards a tag will record a stronger signal than if pointing away. An external antenna with a more uniform characteristic would likely give more even RSSI values regardless of the nodes' orientation relative to the transmitting source.

The buck converter's required external components on the node PCB has shown susceptible to failure. Several times during development the smoothing inductor and freewheeling diode have had to be changed on multiple boards. Failure on these component have in some cases led to over-current in other parts of the circuits and damage to even more components. Troubleshooting has been at times complex and time-consuming when other components have been damaged, and thus changing the 3.3V buck converter to a less error prone substitute would be preferable.

Wiznet's W5500 Ethernet chip has proven reliable for all its tasks. However there were compatibility issues with the Cisco Catalyst 3560-CG PoE network switch with regards to the impedance on the TX and RX line pairs. Using Wiznet's recommended 33Ω resistor on each line, only occasionally was the chip able to negotiate common transmission parameters with the switch while the PoE current draw was very high causing overheating of the PCB and is some cases damage to components. Other switches than the PoE switch worked flawlessly when powering the nodes externally. Further investigation and discussions with technical staff at Wiznet showed that 3.3Ω was a more optimal line impedance. After changing to this configuration, no issues have been seen with the Ethernet autonegotiation, and the nodes have connected to the switch as expected.

nRF52840 Engineering revision A was one of the earliest previews of the nRF52840 on the market. There were several issues with this chip revision affecting relevant functionality to this project, as listed in section 3.2.1.1. However, these could be solved in firmware and have not been affecting platform's functionality.

The high-power LED on the boards have turned out to be *very* high-power. They've been used with very low duty-cycles by default. Changing to a lower power version in a future node PCS revision seems reasonable.

7.1.2 Firmware

Firmware for nodes, tags and a dedicated time synchronization controller has been developed. The time synchronization firmware was implemented into the node application as a convenience, and is as such considered more of a module. Time synchronization has worked as intended, and the interval has been possible to set via the node command system.

The tag firmware that's been running on nRF52840 Preview Development Kits has also worked as expected. It would however have been a useful feature during testing to be able to change the advertisement interval using the buttons on the kit. For all tests it's been static, but in a future revision it would be interesting to see more of the behavior when altering the advertisement interval, in particular if it is to be used for real-time tracking.

Where the tag and time sync firmware may be considered isolated, passive systems, the nodes have been active and interacting with other entities over Ethernet, and thus have a more complex firmware. As far as functionality goes, the node firmware has fulfilled the requirements; it has established wired network connectivity, scanned for tags using the radio in BLE mode, created scan reports, formatted them correctly as JSON strings and sent them in UDP datagrams to the back-end computer for further processing. The scan reports have been correctly received by the computer. The command system described in section 3.3.4.2 has enabled remote control of one or multiple nodes, and been a very useful tool during testing. It's been expanded with more functionality underway. In its current state it's a very useful feature, although not a well-designed system. As it's expanded, it's become evident that it would benefit from using an underlying application protocol for standardization and easier implementation.

The advantages of implementing such an application protocol are likely to be grater than the drawbacks in terms of increased complexity of implementation and network overhead at this point. In early stages of the project, it was considered to use MQTT or its more lightweight cousin MQTT-SN as application protocol, but this was put aside to allow more time spent on other, more central parts of the platform, as it was seen as strictly necessary with such a protocol. Although not necessary for the nodes to function, MQTT or similar certainly would help keeping the implementation clean and scalable. MQTT's hierarchical topic structure would be a welcome feature when addressing nodes one by one or in groups. It could also be considered to send scan reports over MQTT or similar well. While making the design cleaner, an application protocol adds complexity to the back-end computer that would need to act as a server for the protocol.

Scanning of tags and creating scan reports has worked well, with one possible exception: no LPEs (Long Packet Error) have been detected. This *may* be correct, but in nearly 1 million advertisements that have been collected during testing, not a single LPE is recorded. It seems unlikely, and may point to weaknesses in the implementation of this feature. Long packet errors occur when more bits are received during packet reception than what is theoretically possible for a valid packet within the bounds of the platform, and it was included in the firmware to avoid occupying the radio module with long packets that are known to be incorrect and thus terminate the reception. It may just be that the test site was in an environment with very little interfering 2.4 GHz radio traffic, which it was, and thus LPE would be an extremely rare event. It's hard to confirm this, as LPE statistics are not seen in the literature. Further investigation into this possible issue is required before it's possible to draw reliable conclusions.

7.1.3 Processing software

This software has been running on a computer, in the thesis called back-end computer. It's been written in Python, using `socket` module for network functionality. The most important task has been to communicate with the nodes, receive scan reports from them and store the data to a database. Optionally, it has run real-time or delayed tracking of assets using either the live stream of data from the nodes or fetching data from the database.

The basic functionality og node communication and database storage has worked flawlessly during testing. Regrettably, there hasn't been conducted any substantial investigation into the capacity of the program with regards to the amount of data it can handle at the same time. It would be interesting to see how it scales when the node and tag number increases drastically.

Processing of data, both in real-time and with database-fetching, has worked acceptably, but with significant delays when the tag number increased experimentally. No in-depth analysis has been conducted into this, but a superficial investigation shows that it likely relates to the rendering of graphics showing where the tags are at any moment. However, the software is in no way optimized for performance, so it's also possible, not to say likely, that a full redesign of this system would be preferred for future use. The current software has done its job per se, but it's not sufficiently optimized in terms of code quality, not sufficiently modular and fault-tolerant by design and hence error prone.

Although Python programming language has worked well in the work with this thesis, multi-threading using Python or alternative languages could be looked into. In particular Go or similar languages with its light-weight goroutines and communication channels seem like a tempting alternative. Each active node could have its dedicated goroutine receiving and processing all data. The main thread could handle the UDP socket and graphics rendering and not much else. Intuitively this sounds interesting from a performance perspective when considering scalability of the system. It would at least be useful to review the options.

7.2 Positioning algorithm discussion

Positioning and tracking has been done in a 3D space, as the nodes and tags always are located in different horizontal planes. However, doing the positioning tests in 3D space is the real-life situation and it's not realistic to install the nodes in the same height as the tags are expected to be. Because of this, it's expected that the positioning performs less well than is the case in other research where the nodes and tags are typically installed in the same z-plane. On the other hand, considering that the nodes have PCB antennas, it's also expected that having the tags on a different z-plane will cause more evenly measured RSSI values than if the antennas were placed on the same z-plane.

7.2.1 RSSI - distance relationship

The RSSI - distance relationship was investigated and the test results served as the basis for determining the parameters used in the log-distance path loss model in the positioning algorithm. The RSSI values for various distances in figs. 6.1, 6.3 and 6.5 illustrate well how variable RSSI values can be for any given distance. For example, from a sample of one RSSI value of -50 dBm, one can't say with certainty if the device that sent the message is located one or 10 meters away from the receiver. This is main weakness when using RSSI to estimate distance. No matter how well the model for inferring distance from RSSI is, it doesn't help much if the RSSI in itself is inherently unreliable.

It was found in the results that the best overall parameters for the log-distance path loss model was $RSSI_{d_0} = -38.0$ and $n = 1.78$, based on regression in Matlab. These were based on the RSSI - distance tests only. It was also shown that using parameters based on distances derived from the positioning tests somewhat surprisingly did not increase the positioning accuracy. The results of these tests are also comparable to similar research projects, such as [2, 40, 32].

7.2.2 Positioning of static tag

The positioning algorithm performance is largely summed up in fig. 6.6, 77, and section 6.2, 78, which show the cumulative errors for various variants of the algorithm and their statistical properties. The results show that 90% of the position estimates are within 2.8 meters of error from the true position. 80% of the position estimates are within 2 meters of error, and 40% are within 1 meter of error. They also show that performing the algorithm with max channel RSSI is better than mean RSSI with the same parameters for log-distance path loss model. The best results with a mean error is 1.44m, with a median of 1.21m when using max channel selection, $RSSI_{d_0} = -38.0$ and $n = 1.78$. From section 6.2, one can also see that dynamic parameters for log-distance path loss model performs worse in the mean case, but has a smaller maximum error than static parameters, as seen in section 6.2.

Another aspect of the positioning algorithm that was tested, is how fast it can estimate a device's location with some minimum degree of certainty. In this context, "speed" is measured in number of advertising intervals, not seconds. Advertising intervals may range from tens of milliseconds to seconds, so the time aspect becomes relative. For each advertising interval, the positioning algorithm executes one time, so the number of algorithm iterations is equal to the number of advertising intervals recorded by the nodes. As shown in fig. 6.7, the sigmoid curve for the cumulative error function is gradually moved to the left when the number of iterations increases, indicating lower overall error for increasing number of iterations. It's also evident from figs. 6.7 and 6.8 that the accuracy gain when increasing the number of iterations rapidly diminishes when the number exceeds 5 iterations. By letting the number of iterations go from 5 to 80, the gain in mean accuracy is just 4% and 5% in the average and median cases, respectively. In other words waiting more than five advertising intervals for a position estimate for a tag will probably not improve the estimate more than 5%.

So there's an interesting trade-off to be made with the advertising interval and positioning accuracy. For a static tag, the trade-off is likely easy. The gain in accuracy is not sufficient enough to defend a sub-second advertising interval. For a moving object, on the other hand, the trade-off might be more interesting. Depending on the accuracy demands of the given use-case, a lower advertising interval might be required to achieve sufficient accuracy. Self-positioning tests, where one node acts as a tag and is being located, showed that nearly 80% of the position estimates were within 1 meter of the true node position. 7 out of the 8 nodes were estimated to be closer to their actual position than any other node position. The results are interesting, but should be handled with care. They're based on the assumption that only one node is at an unknown located at a time. If that was the case, self-positioning wouldn't be a very useful feature, as it would be easy to match the locations and nodes without it. Perhaps more interesting and useful is the fact that position estimates are generally far more accurate in the self-positioning tests than the other static tag tests. Intuition tells us that practical positioning in two dimensions is simpler than in three, and these results backs up that belief.

The results are comparable in terms of accuracy to what's found in other research projects using BLE and RSSI such as [2]. Other solutions report impressive results of mean errors in the range of 0.5m to 1.0m, which is far better than this thesis was able to achieve [31]. However, as other reports have shown, accuracy varies vastly from environment environment when using RSSI as measurement input, so one should be careful when comparing reults both from this thesis and other reports.

The results are also far less accurate than the best indoor positioning solutions on the market that utilize other technologies for their distance estimates, but they're not really comparable as the require expensive equipment, big installations and use fundamentally different technology [18, 37]. These have also proven to achieve more stable results when used in various different environments than RSSI-based systems.

Compared to the systems in [18], the mean error of the proposed platform and algorithm in this thesis performs pretty average. However, the platform has a dense node distribution and the testing space and without furniture was of limited size, so the validity of comparing the systems is questionable. To really compare to other systems, participation in Microsoft's annual localization competition or similar where systems are directly compared in the exact same setup seems to be the best option.

7.2.3 Tracking of moving tag

Test results for moving tags can be split into two different groups: tag moving without nearby obstructions and tag carried close to human body.

Figures 6.11, 6.12 and 6.14 show the results of moving tag in three different scenarios where there was no obstruction to the direct line of sight between the tag and nodes. The strict timing and movement accuracy requirements for doing statistical analysis on the moving tags data limits this to a qualitative and visual discussion of the results.

A general observation is that the Kalman filters perform well. It's clear by looking at the three different path estimates in each plot that filtering is important to reduce noise. In particular when the tag is carried close to the body, the position estimates based on raw RSSIs tend to be very inaccurate. There's not been put a lot of effort into tuning the Kalman filter as this has not been an important focus of the thesis, but it seems that the filters are tuned to a satisfying level regardless of that.

Comparison of figs. 6.11 and 6.12 shows that the tag being carried at 1.8 meter (fig. 6.11) height undershoots the path in both ends, whereas the tag being pulled on a stand (fig. 6.11) undershoots in both ends. Looking at the z-index estimates for The deviation from the true path is similar, with a comparable curvature on the estimated paths roughly following the curvature of the wall on the plot's left hand side. The same artifact is noticeable in several other, but not all, recordings that are not shown in the report. It may be coincidental, but it might also be related to multipath effects. More investigation into this using more accurate methods and perhaps in different environments is needed.

Tracked paths of moving tag attached to body, either right side pocket or hanging around the neck on the chest are shown in figs. 6.15 to 6.17. Unsurprisingly, the RSSI measurements and hence the position estimates are affected by a body absorbing, blocking and reflecting the radio waves from the tag. The tendency is that nodes that are blocked from direct line of sight measure a significantly lower RSSI than nodes with free line of sight, leading to a path estimate skewed away from the blocked nodes. When the tag is attached to a pocket on the right side of the body, the path is estimated roughly 2-3 meters too far to the person's right hand side. The same applies to a tag hanging on the chest; the path is skewed 2 to 3 meters too far ahead, while the left/right estimate is equal to or lower than 1.5 meters. It's hard to compensate for these effects, and the results show the weaknesses of estimating RSSI from distance. Since there's no way to know where on a body, if at all, a tag is attached, one cannot simply offset the position estimate to correct the position error.

It's hard to compare the results from the moving tag tests with other research results, as there are no objective metrics available, tests are conducted differently, and the indoor environments can vary greatly in terms of furniture, materials on walls, and so on. Advanced equipment is needed to achieve reliable results in such tests, which was also one of the main motives when Microsoft started to invite researchers to participate in an indoor localization competition where the environment and measurement equipment was the same for all systems and participants. However, the results from the moving tag tests are, in the author of this thesis' humble and biased opinion, promising. Visually comparing the estimated paths to the true paths show that the proposed positioning algorithm with Kalman filtering follow the true paths reasonably well.

8 | Conclusion

This thesis has presented an indoor positioning platform, its theoretical background, implementation and practical testing results. The platform consists of nRF52840 based boards with Ethernet connectivity, a Power over Ethernet switch and a back-end computer tracking tags broadcasting BLE advertisement packets. To achieve positioning and tracking, RSSI values for each received packet has been recorded along with other metadata, and sent over Ethernet from the nodes to a computer for processing. The platform can be considered positioning algorithm agnostic, as it allows various radio modes to be used, and it provides raw data that any algorithm can be applied to.

This thesis has presented a positioning algorithm that uses multilateration as its position estimator. The algorithm and its practical implementation has been described in detail. RSSI data has been collected from the nodes and the positioning algorithm has been applied to the data set. As RSSI is inherently unstable and unreliable in unfiltered form, a Kalman filter has been used to filter the RSSI values. Log-distance path loss model has been used to estimate distances using RSSI values. To estimate an asset's location, multilateration with n nodes has been used with non-linear least squares algorithm. A Kalman filter has been applied to the positioning estimates.

The complete solution has been deployed in an indoor testing area and several positioning tests have been carried out. Test data has been gathered and analyzed both in real-time and by post-processing. It's been shown that the platform and positioning algorithm can locate assets with a mean error of 1.44m and median error of 1.21m. The estimation error less than 2.5 meters for 85% of the time in 2-dimensional space. Tests that try to locate a node in the same z-plane as the other nodes, called self-positioning, has shown that nearly 80% of the time, the mean error is equal to or less than 1 meter.

The implemented positioning algorithm's test results have shown that it needs very few iterations to find a stable position estimate for an object that's being tracked. In the average and median case, the initial position estimate can only be improved by 4% and 5%, respectively, by increasing the number of iterations to more than 5. From 1 to 5 iterations there's a 22% and 25% reduction in error for the average and median case. An iteration of the algorithm is equal to one BLE advertising interval. The fact that the implementation of the algorithm can be considered as fast in terms of iterations needed for a good estimate means that tracked devices can have longer advertising intervals and thus use less battery.

The goal of the thesis was to develop a versatile indoor positioning platform, implement a positioning algorithm on the platform, test the whole system and analyze and evaluate it. The results and evaluation show that a fully operational platform has been developed, a positioning algorithm has been implemented and it can be considered fully functional. Thus, the goals of the thesis are achieved.

9 | Future work

This chapter briefly presents some aspects of the positioning platform and positioning algorithm that there was not room for within the scope of this thesis, and that would be interesting to look more into.

- An interesting area to look more into is self-positioning of the nodes on a floor plan of a building. The solution in this project requires manual placement of the nodes in a coordinate system, which again demands knowledge of the environment. One can imagine a system where a few anchor nodes' positions are defined in a floor plan, and from that the positions of all the other nodes can be estimated. An option is to define all possible node placements in the floor plan, and then infer from the collected data which node is most likely to be positioned at the various available locations.

- To expand on the thought of using floor plans, it would be very interesting to use floor plans as an input in the positioning algorithm. Perhaps multipath effects can be alleviated by looking at the nodes' placement relative to walls and even depending on the wall materials. This would have to somehow be automated to minimize manual work and make the platform resilient to changes in the environment and deployment in differing environments

- Add the option to use to wireless collection of the data from the nodes. This is already possible with the hardware used in this project as a footprint for the Wi-Fi module WizFi310 is already in place. A possible solution could be to scan for tags most of the time, and in short bursts send the collected information to the back-end using Wi-Fi. A challenge is to avoid radio interference as the radios of Wi-Fi and nRF52840 both utilize the 2.4 GHz band.

- Large data sets can be recorded using the positioning platform. It would be interesting to apply machine learning techniques to the data to investigate if there are better ways to use the available data to estimate distances and by extension, positions.

- The already possibility in the platform for firmware upgrades using the Ethernet network should be expanded on and automated. An application should be made that easily programs all nodes without any manual work. There already exists a command for the case, and should mostly be to allow this command to make the node enter the bootloader's firmware update mode.

- It should be looked more into how the time synchronization signal can be used in the algorithm to discover potential temporal factors that may affect the precision of the system. Consider adding a clock with higher resolution that perhaps can enable the use of time-of-flight or similar techniques in distance estimation.

Bibliography

[1] Microsoft indoor localization competition – ipsn 2018. https://www.microsoft.com/en-us/research/event/microsoft-indoor-localization-competition-ipsn-2018/, Apr 2018.

[2] V. Cantón Paterna, A. Calveras Augé, J. Paradells Aspas, and M. A. Pérez Bullones. A bluetooth low energy indoor positioning system with channel diversity, weighted trilateration and kalman filtering. *Sensors (Basel, Switzerland)*, 17(12), December 2017.

[3] Cisco. Multipath and diversity. https://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/27147-multipath.html, January 2018. [Online; accessed 19-May-2018].

[4] T. S. community. scipy.optimize.least_squares. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html, 2018.

[5] R. Faragher and R. Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+'14)*, pages 201–210, 2014.

[6] R. Faragher and R. Harle. Location fingerprinting with bluetooth low energy beacons. *IEEE Journal on Selected Areas in Communications*, 33(11):2418–2428, Nov 2015.

[7] Q. Fu and G. Retscher. Active rfid trilateration and location fingerprinting based on rssi for pedestrian navigation. *The Journal of Navigation*, 62(2):323–340, 2009.

[8] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[9] B. S. I. Group. Bluetooth core specification v5.0. Technical report, Bluetooth SIG, 2016.

[10] J. T. Guggedal. Multiprotokoll gateway. 2017.

[11] Hans Elfberg. Sensor beacon. https://github.com/NordicPlayground/solar_sensor_beacon, 2017. [Online; accessed 13-May-2018].

[12] Hung Bui. Bluetooth smart and the nordic's softdevices - part 1 gap advertising. https://devzone.nordicsemi.com/b/blog/posts/bluetooth-smart-and-the-nordics-softdevices-part-1, 2015. [Online; accessed 15-May-2018].

[13] IEEE. Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 3: Csma/cd access method and physical layer specifications amendment 3: Data terminal equipment (dte) power via the media dependent interface (mdi) enhancements. *IEEE Std 802.3at-2009 (Amendment to IEEE Std 802.3-2008)*, pages 1–137, Oct 2009.

[14] IEEE. Ieee standard for ethernet. *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, pages 1–4017, March 2016.

[15] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

[16] R. Labbe. Filterpy - kalman filters and other optimal and non-optimal estimation filters in python. https://github.com/rlabbe/filterpy, 2018.

[17] R. Labbe. Kalman and bayesian filters in python. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python, 2018.

[18] D. Lymberopoulos and J. Liu. The microsoft indoor localization competition: Experiences and lessons learned. *IEEE Signal Processing Magazine*, 34(5):125–140, Sept 2017.

[19] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen. A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned. In *Proceedings of the 14th international conference on information processing in sensor networks*, pages 178–189. ACM, 2015.

[20] MazeMap. Indoor positioning and blue dot with the mazemap platform. https://www.mazemap.com/indoor-positioning, 2018. [Online; accessed 19-May-2018].

[21] A. A. Panyov, A. A. Golovan, and A. S. Smirnov. Indoor positioning using wi-fi fingerprinting pedestrian dead reckoning and aided ins. In *2014 International Symposium on Inertial Sensors and Systems (ISISS)*, pages 1–2, Feb 2014.

[22] Python. general python faq. 2018. [Online; accessed 11-May-2018].

[23] N. Rajagopal, J. Miller, K. K. R. Kumar, A. Luong, and A. Rowe. Welcome to my world: demystifying multi-user ar with the cloud: demo abstract. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 146–147. IEEE Press, 2018.

[24] T. S. Rappaport. *Wireless communications principles and practices.* Prentice-Hall, 2002.

[25] Coursa retail. In-store shopping behavior and shopper location analytics from coursa retail. `http://www.coursaretail.com/`, 2018. [Online; accessed 19-May-2018].

[26] Nordic Semiconductor. nrf52840 dk. `https://www.nordicsemi.com/eng/Products/nRF52840-DK`, 2017. [Online; accessed 19-May-2018].

[27] Nordic Semiconductor. nordic semiconductor infocenter - sdk 14.2.0 introduction. 2017. [Online; accessed 13-May-2018].

[28] Nordic Semiconductor. nrf52840 engineering a errata v1.5. `http://infocenter.nordicsemi.com/pdf/nRF52840_Engineering_A_Errata_v1.5.pdf`, 2018. [Online; accessed 19-May-2018].

[29] Nordic Semiconductor. nRF52840 Product Specification v1.0, 2018.

[30] Nordic Semiconductor. S140 softdevice. `https://www.nordicsemi.com/eng/Products/S140-SoftDevice`, 2018. [Online; accessed 06-March-2018].

[31] M. E. Rida, F. Liu, Y. Jadi, A. A. A. Algawhari, and A. Askourih. Indoor location position based on bluetooth signal strength. In *2015 2nd International Conference on Information Science and Control Engineering*, pages 769–773, April 2015.

[32] M. E. Rusli, M. Ali, N. Jamil, and M. M. Din. An improved indoor positioning algorithm based on rssi-trilateration technique for internet of things (iot). In *2016 International Conference on Computer and Communication Engineering (ICCCE)*, pages 72–77, July 2016.

[33] J. Röbesaat, P. Zhang, M. Abdelaal, and O. Theel. An improved ble indoor localization with kalman-based fusion: An experimental study. *Sensors (Switzerland)*, 17(5), May 2017.

[34] A. Savvides, C.-C. Han, and M. B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 166–179. ACM, 2001.

[35] SEGGER. Segger embedded studio. `https://www.segger.com/products/development-tools/embedded-studio/`, 2018. [Online; accessed 06-April-2018].

[36] B. SIG. Generic access profile. `https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile`, 2016. [Online; accessed 15-May-2018].

[37] B. Silva, Z. Pang, J. Åkerberg, J. Neander, and G. Hancke. Experimental study of uwb-based high precision localization for industrial applications. In *2014 IEEE International Conference on Ultra-WideBand (ICUWB)*, pages 280–285, Sept 2014.

[38] S. Tomažič and I. Škrjanc. Bluetooth localization based on fuzzy models and particle swarm optimization. In *Indoor Positioning and Indoor Navigation (IPIN), 2017 International Conference on*, pages 1–8. IEEE, 2017.

[39] I. T. Union. P.1238 : Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 300 mhz to 100 ghz, 2017.

[40] Y. Wang, Q. Ye, J. Cheng, and L. Wang. Rssi-based bluetooth indoor localization. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 165–171, Dec 2015.

[41] Wikipedia contributors. Kalman filter — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Kalman_filter&oldid=843991639`, 2018. [Online; accessed 13-May-2018].

[42] Wikipedia contributors. Talk: Multilateration. `https://en.wikipedia.org/w/index.php?title=Talk:Multilateration&oldid=822192892#Multilateration_vs_Trilateration`, 2018. [Online; accessed 19-May-2018].

[43] WIZnet. W5500 datasheet v1.0.6. 2013.

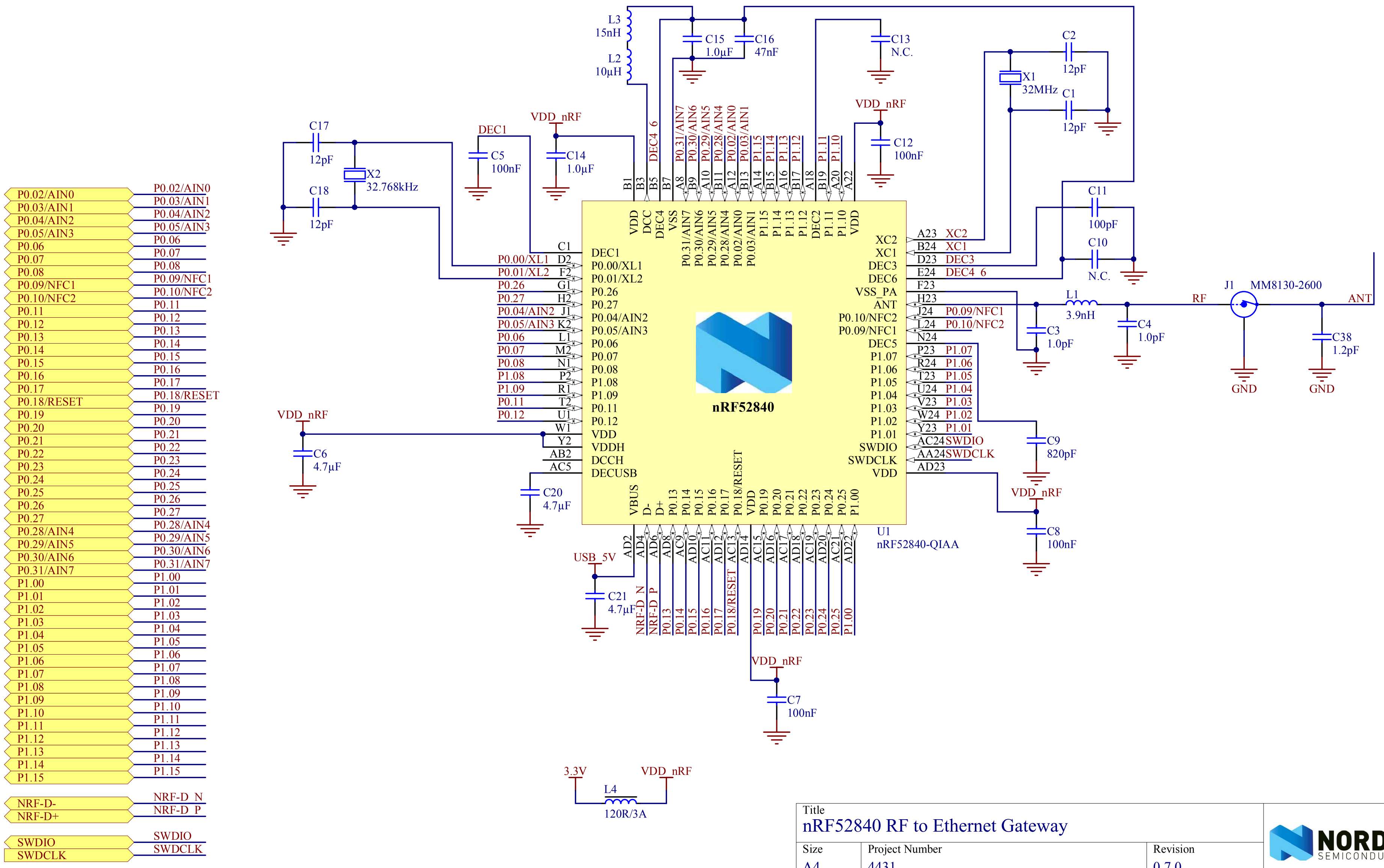
[44] Wiznet. Wizble52840 ethernet. `https://github.com/Wiznet/WIZBLE52840_ETHERNET`, 2017.

[45] Yael Landau, Boaz Ben-Moshe, Itiel Rosenberg. Steps – an accurate relative positioning method for first-responders. 2018.

[46] Y. Zhou. An efficient least-squares trilateration algorithm for mobile robot localization. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3474–3479. IEEE, 2009.

A | Node schematics and PCB layout

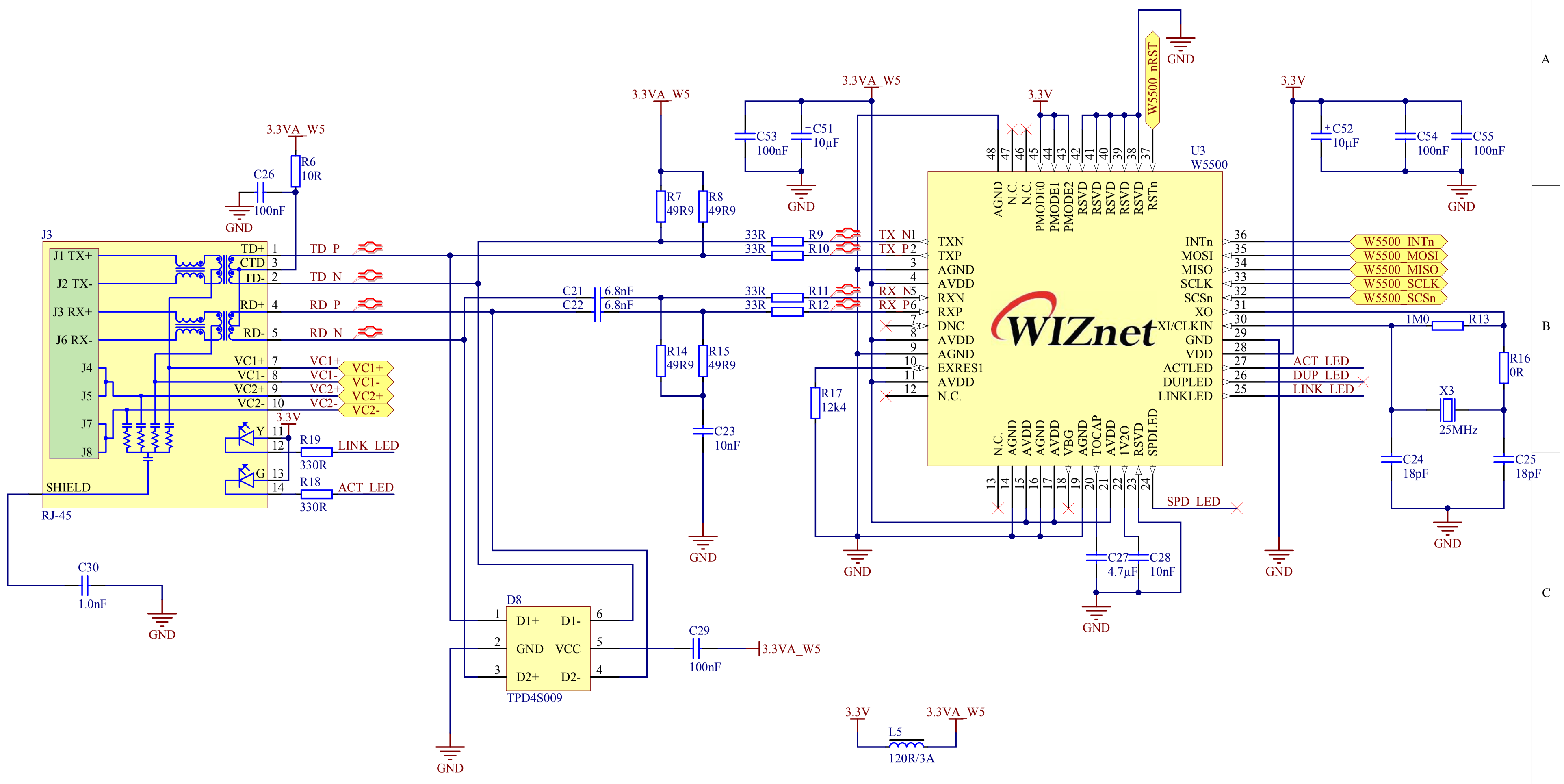
The following pages show the schematics and PCB design for the node boards in this thesis, called PCA20036.



Title
nRF52840 RF to Ethernet Gateway

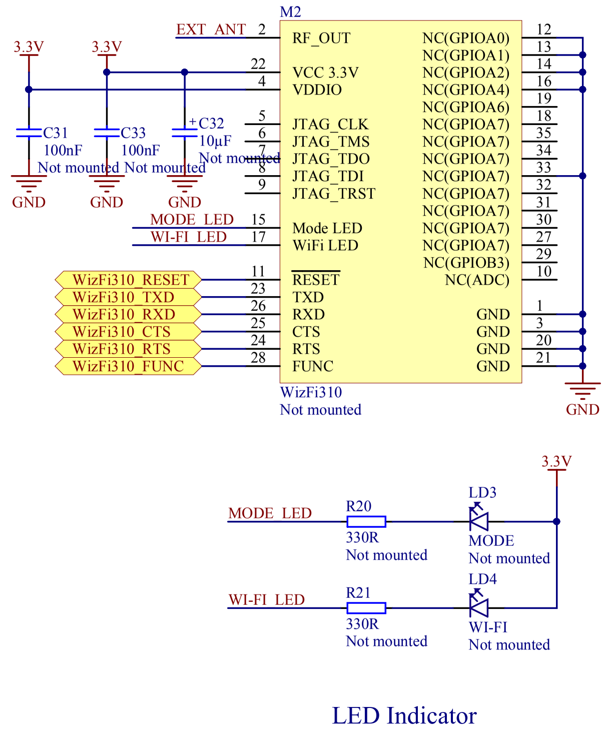
Size	Project Number	Revision
A4	4431	0.7.0

Date: 25.10.2017
Sheet 1 of 8
File: pca20036_nrf52840.SchDoc
Drawn By: RUBR

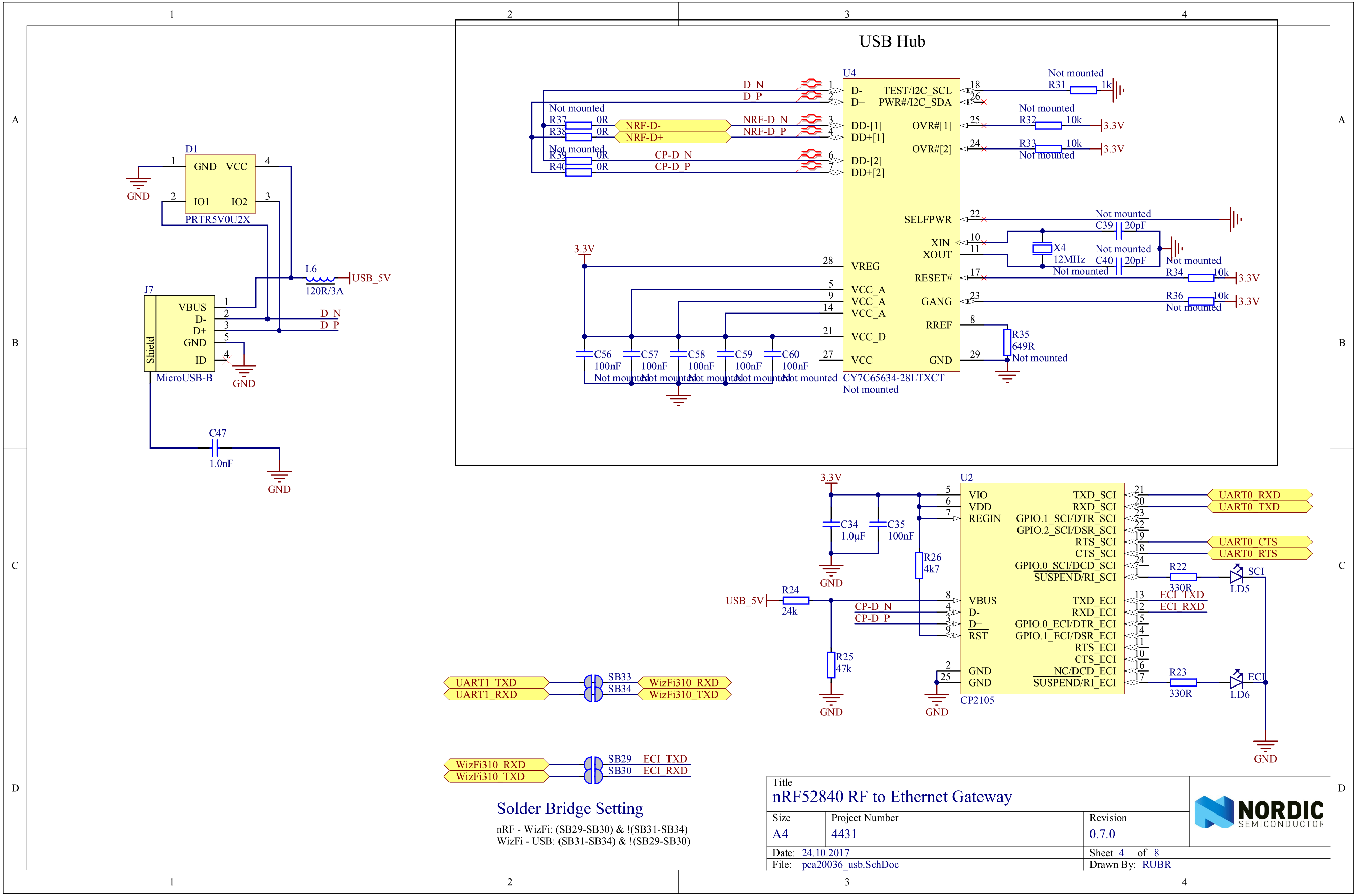


Title		
nRF52840 RF to Ethernet Gateway		
Size	Project Number	Revision
A4	4431	0.7.0
Date: 09.10.2017		Sheet 2 of 8
File: pca20036_w5500.SchDoc		Drawn By: RUBR

NORDIC SEMICONDUCTOR



Title		
nRF52840 RF to Ethernet Gateway		
Size A4	Project Number 4431	Revision 0.7.0
Date: 14.10.2017		Sheet 3 of 8
File: pca20036_wizfi310.SchDoc		Drawn By: RUBR



Solder Bridge Setting

nRF - WizFi: (SB29-SB30) & !(SB31-SB34)
WizFi - USB: (SB31-SB34) & !(SB29-SB30)

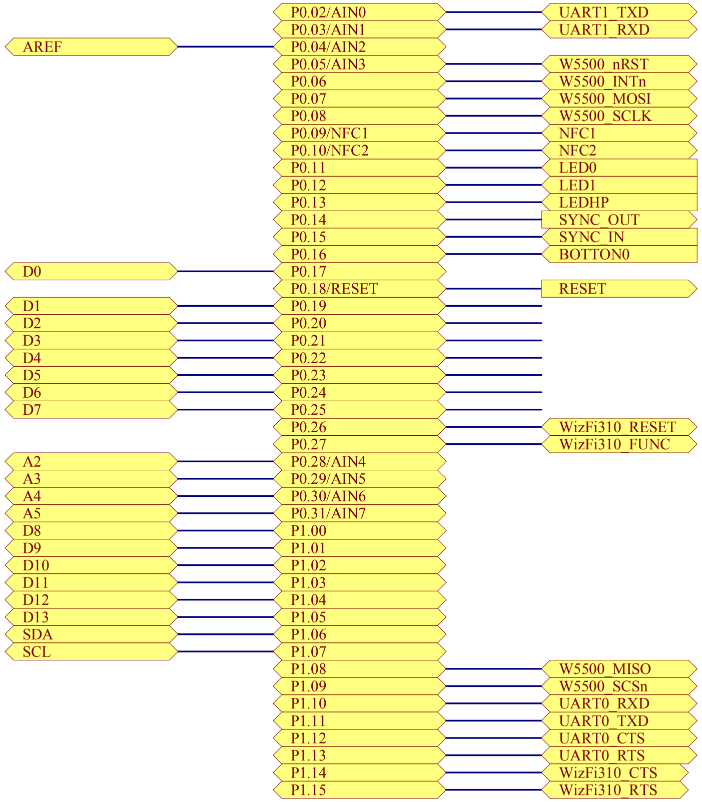
Title
nRF52840 RF to Ethernet Gateway

Size	Project Number	Revision
A4	4431	0.7.0

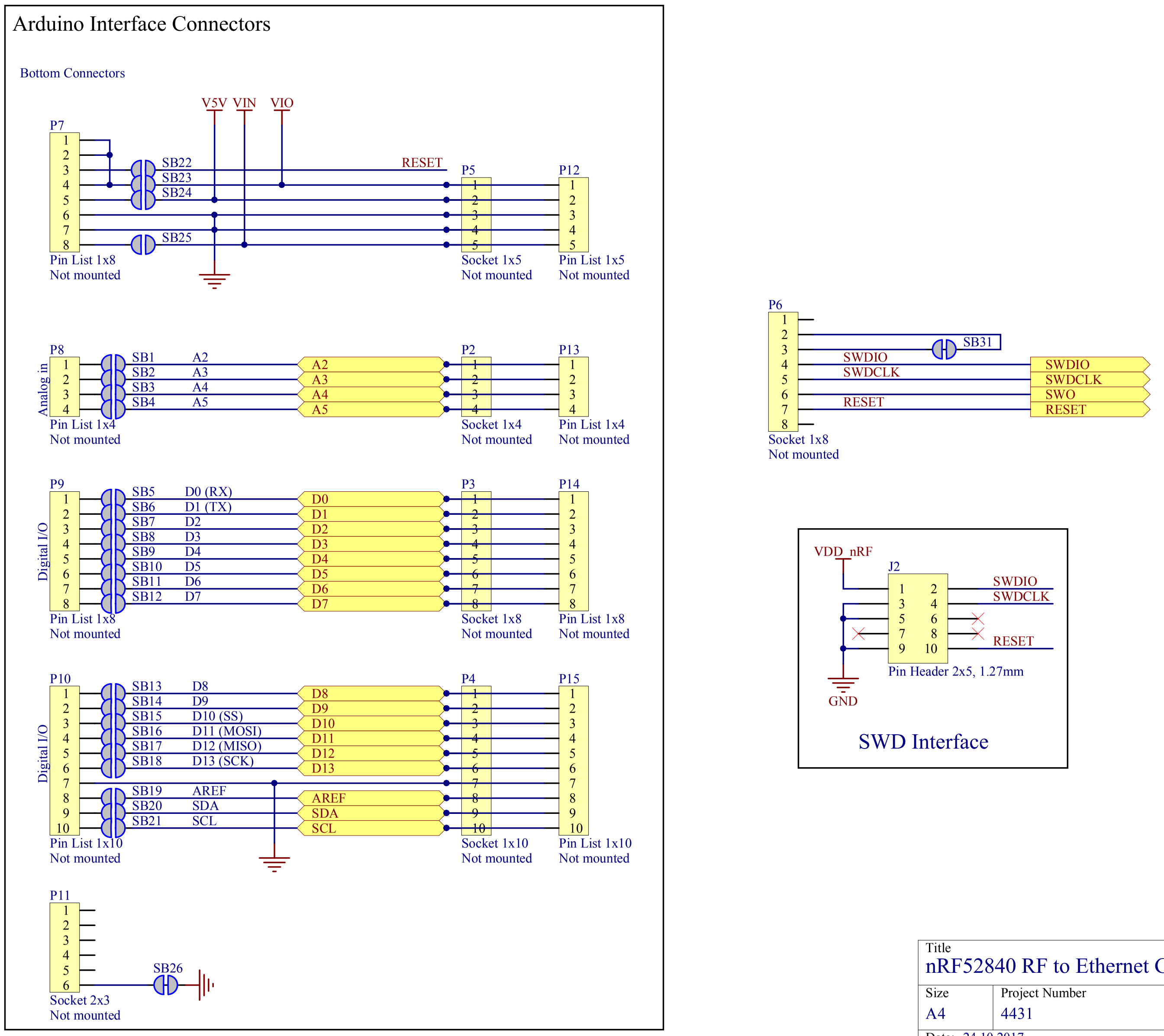
Date: 24.10.2017
Sheet 4 of 8
File: pca20036_usb.SchDoc
Drawn By: RUBR

NORDIC SEMICONDUCTOR

GPIO pin mapping



Title			
nRF52840 RF to Ethernet Gateway			
Size A4	Project Number 4431	Revision 0.7.0	
Date: 20.10.2017		Sheet 5 of 8	
File: pca20036_pin_map.SchDoc		Drawn By: RUBR	

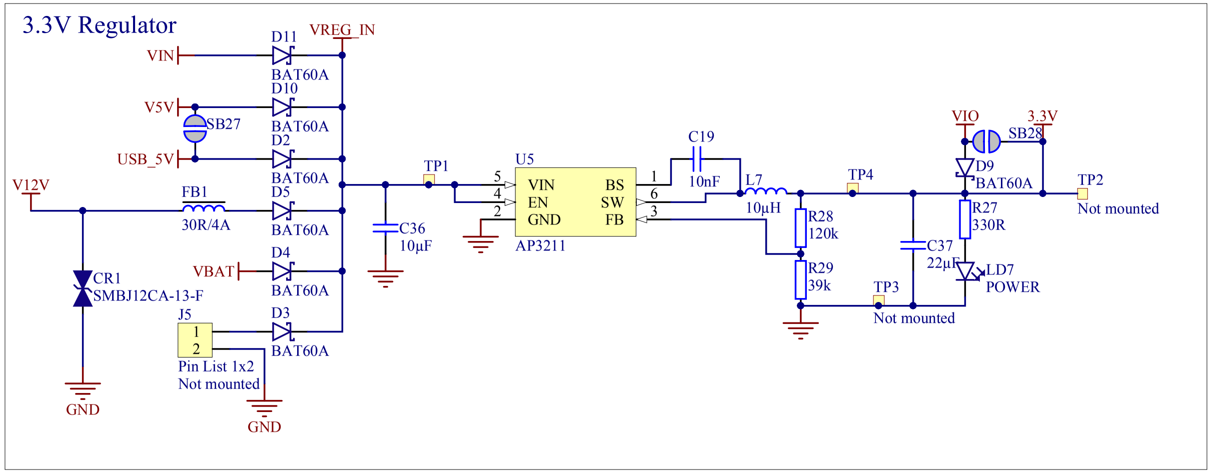


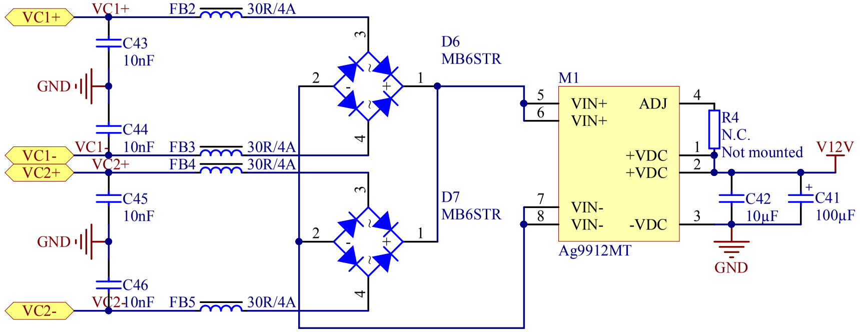
Title
nRF52840 RF to Ethernet Gateway

Size	Project Number	Revision
A4	4431	0.7.0

Date: 24.10.2017
File: pca20036_dk_interface.SchDoc
Sheet 6 of 8
Drawn By: RUBR

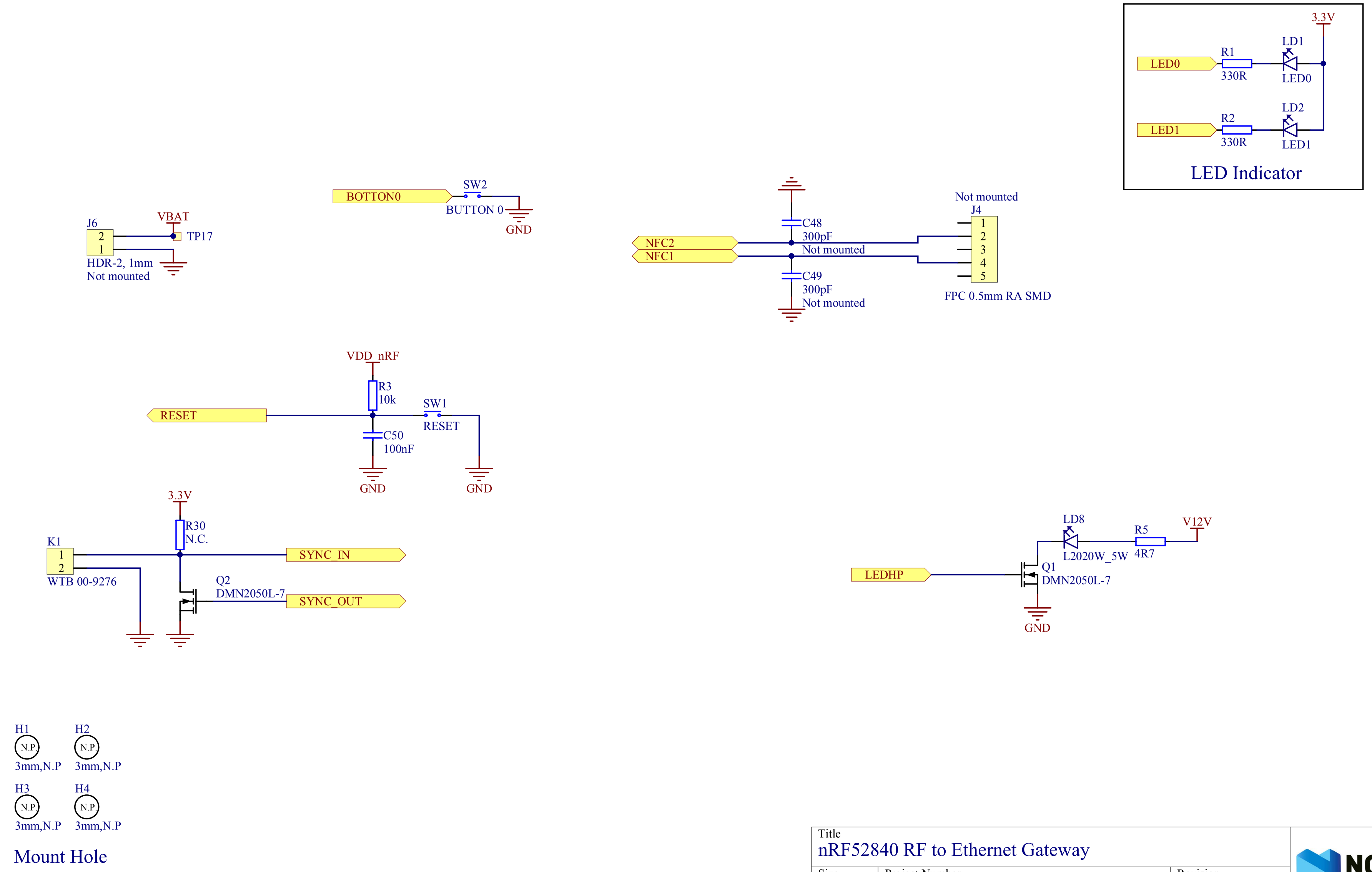
3.3V Regulator



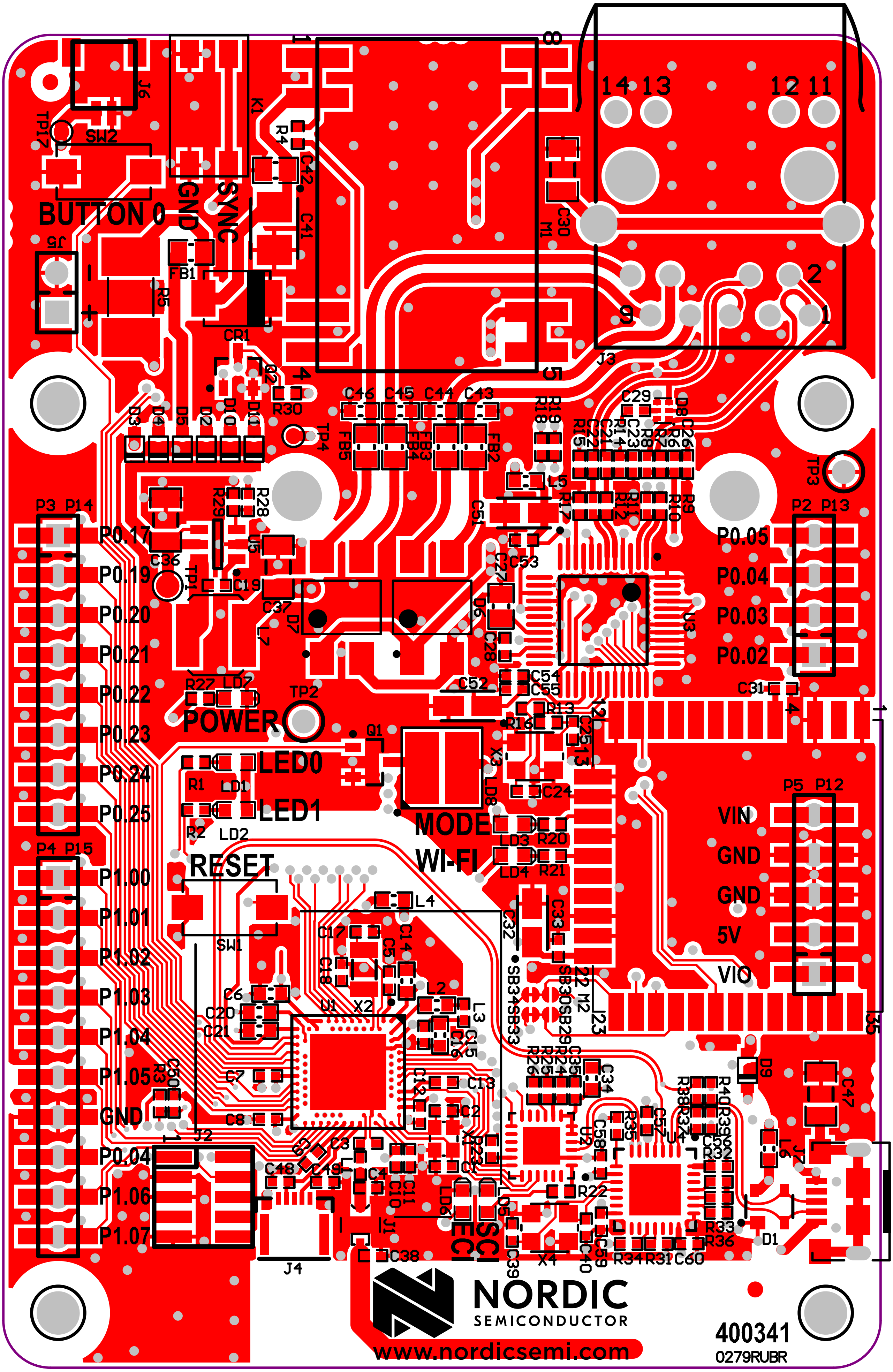


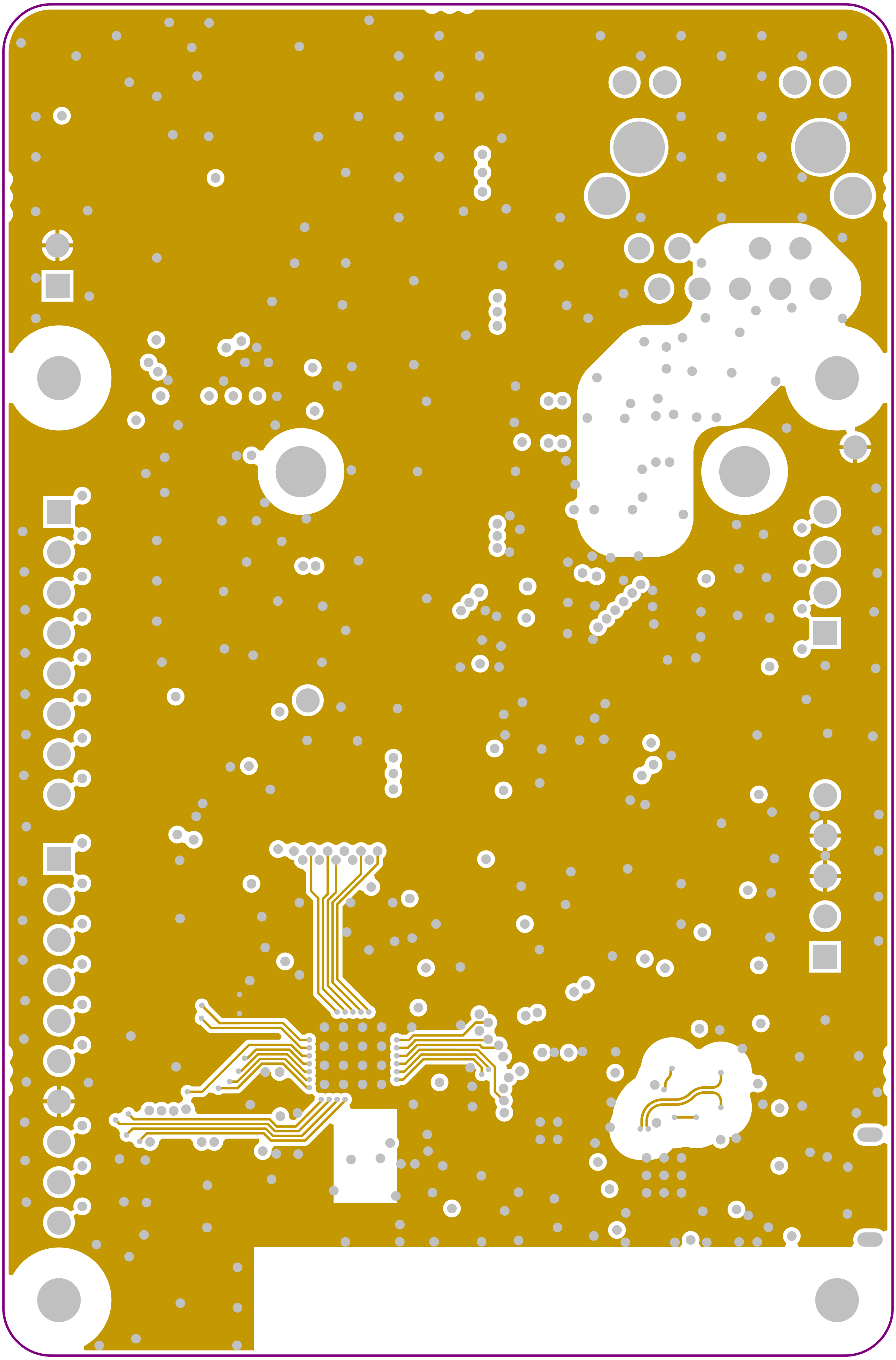
Title
nRF52840 RF to Ethernet Gateway

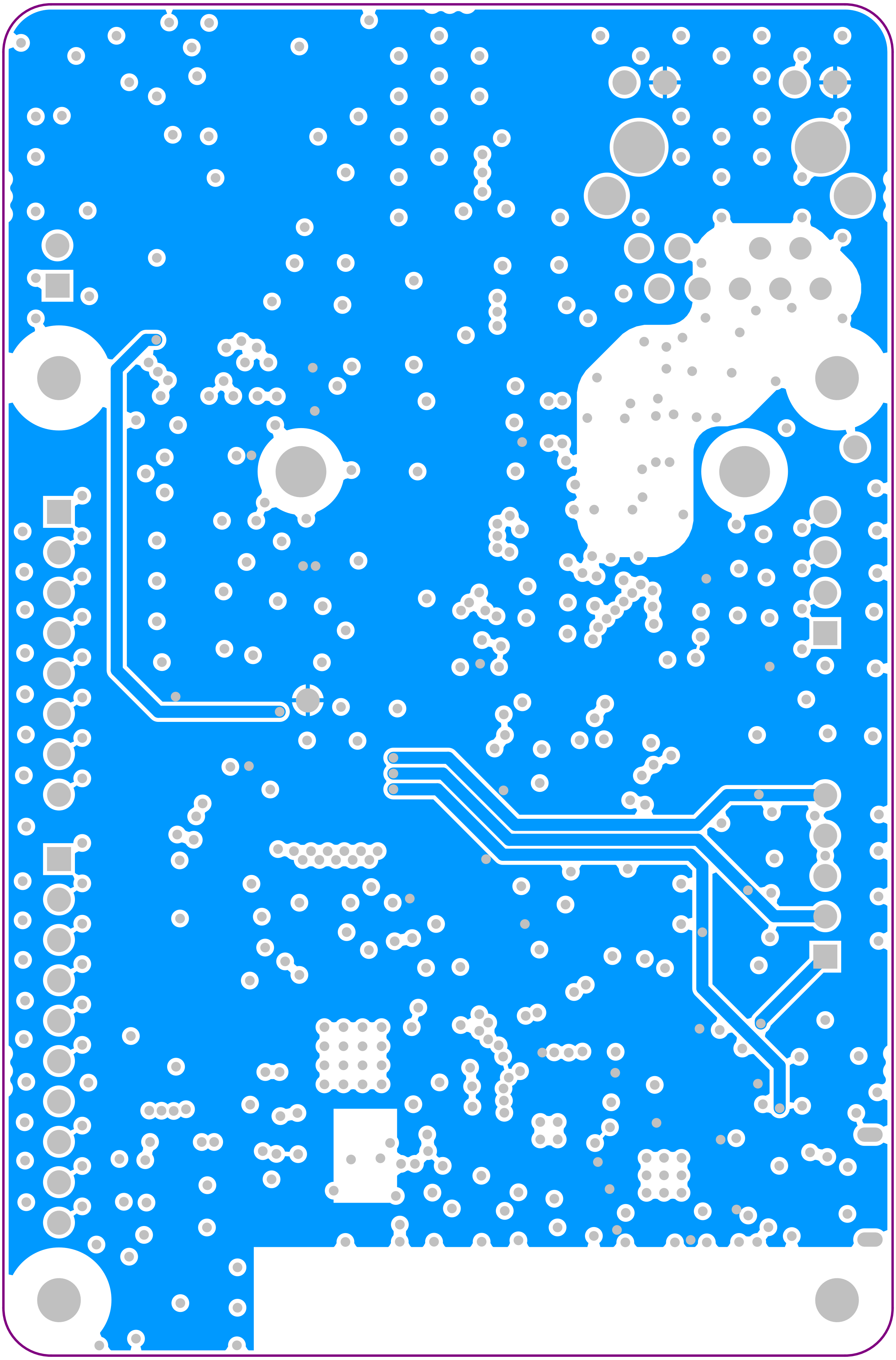
Size	Project Number	Revision
A4	4431	0.7.0
Date: 24.10.2017		Sheet 7 of 8
File: pca20036_power_supply.SchDoc		Drawn By: RUBR

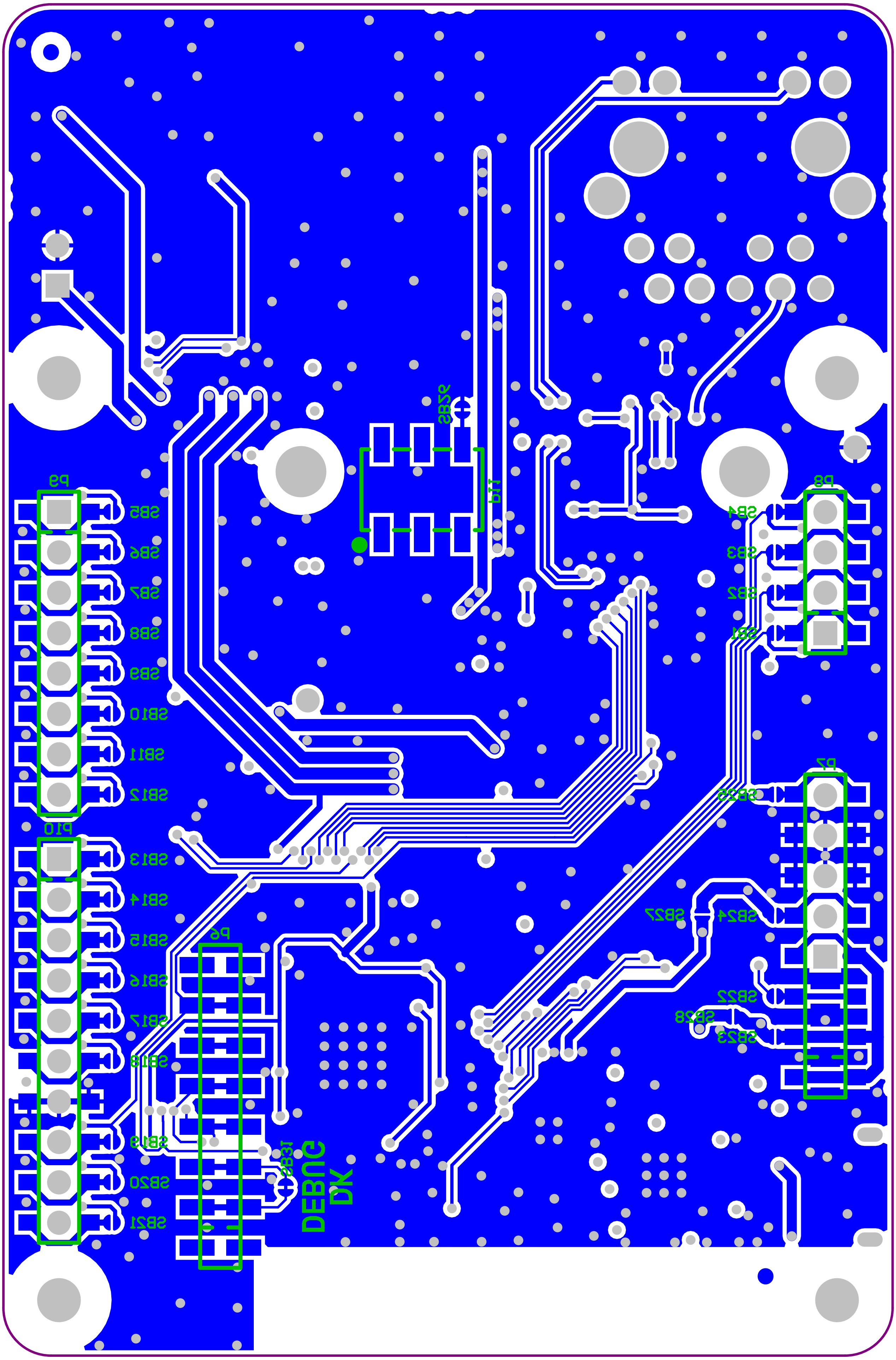


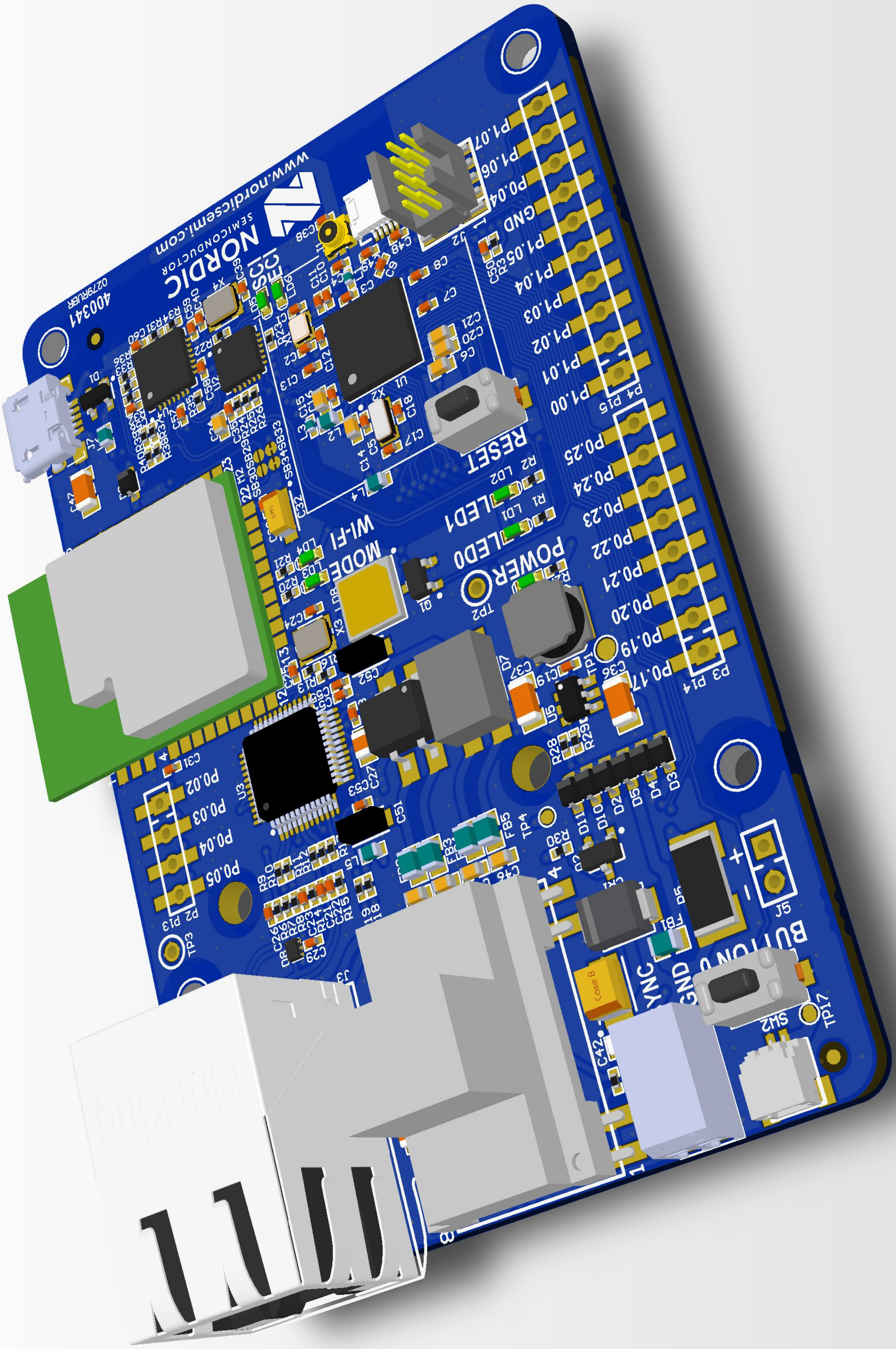
Title		
nRF52840 RF to Ethernet Gateway		
Size A4	Project Number 4431	Revision 0.7.0
Date: 23.10.2017		Sheet 8 of 8
File: pca20036_etc.SchDoc		Drawn By: RUBR


BUTTON 0
GND
SYNC
J6
SW2
TP17
K1
J5
FB1
CR1
J3
14 13
12 11
M1
C30
C41
C42
R4
R5
POWER
TP2
LED0
LED1
MODE
WI-FI
RESET
SW1
LD1
LD2
LD3
LD4
LD7
R1
R2
R27
Q1
Q2
X2
X3
X4
U1
U2
U3
U5
TP1
TP3
TP4
FB2
FB3
FB4
FB5
C43
C44
C45
C46
C29
P3 P14
P4 P15
P2 P13
P5 P12
P0.17
P0.19
P0.20
P0.21
P0.22
P0.23
P0.24
P0.25
P1.00
P1.01
P1.02
P1.03
P1.04
P1.05
GND
P0.04
P1.06
P1.07
P0.05
P0.04
P0.03
P0.02
VIN
GND
GND
5V
VIO
J2
J4
J7
D1
D9
L1
L2
L3
L4
L5
L6
L7
C47
C38
C39
C40
SB34
SB33
SB30
SB29
NORDIC
SEMICONDUCTOR
www.nordicsemi.com
400341
0279RUBR






P9
P10
P6
P7
P8
U7
SB1
SB2
SB3
SB4
SB5
SB6
SB7
SB8
SB9
SB10
SB11
SB12
SB13
SB14
SB15
SB16
SB17
SB18
SB19
SB20
SB21
SB22
SB23
SB24
SB25
SB26
SB27
SB28
SB29
SB31
DEBUG
DK


NORDIC
SEMICONDUCTOR
www.nordicsemi.com
400341
RESET
LED1
LED0
POWER
WI-FI
MODE
BUTTON
P1.07
P1.06
P0.04
GND
P1.05
P1.04
P1.03
P1.02
P1.01
P1.00
P0.25
P0.24
P0.23
P0.22
P0.21
P0.20
P0.19
P0.17
P0.02
P0.03
P0.04
P0.05
J5
SW2
TP17

B | Command system

This appendix contains all the commands that are defined in the command system that can be used to remotely control the node in the indoor positioning platform. The commands are defined in the firmware file `commands.h`. Commands are sent from a computer to all nodes.

All commands have the format shown in table B.1.

Field type	Byte index	Byte length	Default value
Prefix	0	16	`CONTROL_COMMAND:`
Command	16	1	
Payload length	17	1	
Payload	18	-	

Table B.1: Control command message format

Byte index refers to the index of the string that's being sent from the computer to the nodes. The available commands that can be used are shown in table B.2, 122.

Cmd.	Defined name	Payload	Description
1	`WHOAMI_START`	-	Starts to send WHO AM I messages containing the node's MAC and IP every main loop iteration
2	`WHOAMI_STOP`	-	Stops sending WHO AM I messages
10	`CMD_SERVER_IP_BROADCAST`	<IP>:<port>	Broadcast message from back-end computer containing IP and port for nodes to contact it
11	`CMD_NEW_SERVER_IP`	<IP>:<port>	Sent if computer obtains new IP address or changes port number
12	`CMD_NEW_FIRMWARE`	-	The node can download new firmware via TFTP
13	`CMD_NEW_ACCESS_ADDRESS`	<6 byte address>	Sets new BLE access address
14	`CMD_ADVERTISING_START`	-	The nodes starts advertising
15	`CMD_ADVERTISING_STOP`	-	Stop advertising
40	`CMD_ALL_HPLED_ON`	-	Switches on all the nodes' HP LEDs
41	`CMD_ALL_HPLED_OFF`	-	Switches off all the nodes' HP LEDs
42	`CMD_ALL_HPLED_DEFAULT`		Sets HP LED duty cycle to default
43	`CMD_ALL_HPLED_NEW_DEFAULT`	<units of 1/10%>	Sets new default HP LED duty cycle
44	`CMD_ALL_HPLED_CUSTOM`	<units of 1/10%>	Sets custom HP LED duty cycle
50	`CMD_SINGLE_HPLED_ON`	-	Switches on single node's HP LED
51	`CMD_SINGLE_OFF`	-	Switches off single node's HP LED
52	`CMD_SINGLE_HPLED_DEFAULT`	-	Sets single node's HP LED to default
53	`CMD_SINGLE_HPLED_CUSTOM`	<units of 1/10%>	Sets single node's HP LED to custom value
60	`CMD_SINGLE_ADVERTISING_ON`	-	Single node starts advertising
61	`CMD_SINGLE_ADVERTISING_OFF`	-	Single node stops advertising
70	`CMD_SYNC_NODE_SET`	<IP address>	One node gets the role as synchronization signal controller
71	`CMD_SYNC_SET_INTERVAL`	<units of 100ms>	Sets new synchronization signal interval

Table B.2: Control commands

C | Codebase overview

This appendix provides and overview of the codebase in this thesis. The code is attached to the delivery, and can also be found online in two public repositories, with both sources having the same content and folder structure:

- **Firmware**: `https://github.com/jtguggedal/positioning_firmware`
- **Back-end software**: `https://github.com/jtguggedal/positioning_backend`

C.1 Firmware

The firmware is split into four folders:

- `node`: contains the node firmware
- `node_bootloader_SDK14.2`: contains the node bootloader
- `tag`: contains the tag firmware used during testing
- `time_syncer`: contains firmware for PCA10056 development kit that can act as an external time synchronizer

All the folders contain the source and `main.c` files needed for the different applications. They also contain one `.emProject` file that is a project file for SEGGER Embedded Studio. To compile the project, download SEGGER Embedded Studio, and open the project file. As all dependencies are in the repository or available in Nordic Semiconductor's nRF5 SDK 14.2, the firmware should then be able to compile without errors and warnings.

C.2 Back-end software

The back-end software for listening for scan reports, storing them to database and apply the positioning algorithm is also provided. The most central files and modules are the following:

- `node_listener.py`: this is the simplest possible back-end program. It regularly sends out command to the nodes with its IP address and port number and listens to a UDP socket for incoming scan reports. When scan reports come, they can optionally be stored to database.

- `db/`: folder that contains SQL code to create the necessary database and tables for the back-end software.

- `experiments/positioning_test.py`: this program was used during testing. It does the same as the node listener, but can in addition run the positioning algorithm in real-time.

- `utils/setup.py`: runs a configuration setup program that allows the user to enter the true node positions and store them to a configuration file. This file can later be used with other scripts to automatically provide node information if real-time or post-processing will happen.

- `calc/`: helper module that implements log-distance path loss model, trilateration and multilateration.

- `positioning/`: module that implements position, node and tag classes. The position and tag classes implements respective Kalman filters.

- `analysis/log_distance_parameters.py`: analysis program used to estimate log-distance path loss model parameters experimentally.

- `analysis/position_estimation.py`: program used to fetch data from database and apply positioning algorithm. Can save estimated positions to CSV files for further analysis in other programs, such as Matlab.