Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Pascal Pickel

Indoor 3D Positioning System

Using Bluetooth 5.1 Direction Finding

Bachelor's project in Computer Science
Supervisor: Tomas Holt
May 2021

NTNU
Kunnskap for en bedre verden

NORDIC SEMICONDUCTOR

Pascal Pickel

Indoor 3D Positioning System

Using Bluetooth 5.1 Direction Finding

Bachelor's project in Computer Science
Supervisor: Tomas Holt
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for en bedre verden

Preface

This project concludes my three years bachelor's degree programme of Engineering in Computer Science at *Norwegian University of Science and Technology (NTNU)*.

Selecting a bachelor project provided by *Nordic Semiconductor ASA (Nordic)* was a given as I have had the pleasure of being employed there throughout the entire three years, naturally forming a close connection with both the people working there and their products. Still, as I had been working mainly firmware related tasks, I felt the need to explore and develop skills for other areas, namely web and general front end development. As such, this project was a perfect fit allowing me to work with *Nordic*'s products while developing a desktop *Application (App)* using web technology.

I would like to express my gratitude to my supervisors Ketil Erichsen and Nicolai Berthelsen from *Nordic* and Tomas Holt from *NTNU* for their continuous support. I also would like to express my thanks to *Nordic* as a whole for giving me this opportunity and for offering me a suitable development and test environment, which was crucial for a satisfactory result.

Student Date

Task

The task, found in full in Appendix A, asks for a demonstration of a 3D *Indoor Positioning System (IPS)* using *Nordic*'s existing hardware and firmware solution, consisting of Bluetooth devices utilising a relatively new feature called Bluetooth Direction Finding.

The system I am tasked to develop is to consist of a software solution written in JavaScript on the React framework. While not specifically mentioned in the task description, the system will be launched through *Nordic*'s nRF Connect for Desktop Electron launcher and therefore visually resemble other nRF Connect for Desktop *Apps*. This enables me to utilise existing *Nordic* React components [1] and save time otherwise used on creating standard components such as a navigation bar.

The original task description gives some leeway in terms of requirements and one should therefore refer to the vision document in Appendix B and the requirements document in Appendix C for a more detailed look at these.

Abstract

This project demonstrates how to utilise Bluetooth Direction Finding, a new feature among the Bluetooth Core specifications [2], to create a 3D *IPS*. While an *IPS* is nothing new and exists in many different shapes and forms like through the use of WiFi, it is through Bluetooth Direction Finding that Bluetooth devices really find their place in location tracking systems. In an age where everyone is already equipped with a Bluetooth supporting device, namely a smartphone, a shopping centre navigation system is just one of the many systems that become possible through Bluetooth Direction Finding.

Nordic has an existing hardware solution that supports Bluetooth Direction Finding, consisting of nRF52833 *Development Kits (DKs)* and expansion boards with an antenna array. Through the distance between the individual antennas it is possible to calculate the direction of an incoming Bluetooth transmission. By using multiple such devices to track a common *tag*, a 3D position can be deduced.

The *App* was developed through an iterative process, not adhering to any specific frameworks due to the limited scope and singular project member, and is written in JavaScript on the React framework. Most of its features are related to the receiving and manipulation of serial data and user input handling, yet they all impact a scaled version of the system setup displayed in a 3D view port. This allowed for a simple and concise presentation of the features and progress during the weekly meetings, as there was always a visual effect related to the features.

The final product, while definitely leaving room for improvements, is solid and very satisfactory based on mine and most importantly the client's expectations. Outside of the main functionality, allowing for the display of a 3D real-time position, there are many smaller features added throughout the project period resulting in a generally well-rounded product.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AoA Angle of Arrival.

AoD Angle of Departure.

App Application.

BLE Bluetooth Low Energy.

CPU Central Prosessing Unit.

DK Development Kit.

IPS Indoor Positioning System.

LED Light-emitting diode.

NFC Near-field communication.

Nordic Nordic Semiconductor ASA.

NTNU Norwegian University of Science and Technology.

QOL Quality of life.

RSSI Received Signal Strength Indicator.

RTLS Real-time Location System.

SoC System on a Chip.

UI User Interface.

USB Universal Serial Bus.

Glossary

Black box A system who is only described in terms of input and output without any knowledge of it's internal workings.

Bluetooth Low Energy A wireless network technology intended to reduce the power consumption of classic Bluetooth but maintaining a similar communication range.

Bluetooth Mesh A standard based on *Bluetooth Low Energy (BLE)* detailing the many-to-many communications through Bluetooth radio.

Indoor Positioning System A system tracking the position of an object within an indoor area.

Locator Bluetooth device that tracks the direction of a Bluetooth transmission sent by a *tag*.

Quaternion A member of the quaternion number system commonly used to describe three-dimensional rotations.

Real-time Location System A system tracking the position of an object in real-time.

Received Signal Strength Indicator A indicator of the power present in a received radio signal transmission.

System on a Chip An integrated circuit that usually include but are not limited to a *Central Prosessing Unit (CPU)*, memory, input/output ports and secondary storage.

Tag Bluetooth device whose position is tracked in a Bluetooth Direction Finding system.

Unit vector Vector whose magnitude is equal to 1. For a 3D vector: $\sqrt{x^2+y^2+z^2} = 1$.

Zenith An imaginary point directly ”above”. Usually used in the context of astronomy and refers to the direction opposite to that of gravity.

1 Introduction

1.1 Background

With the recent addition of the Bluetooth Direction Finding feature in the Bluetooth 5.1 Core Specification [2], a new market has opened up for Bluetooth devices. *Nordic* has developed a sample *App* to showcase this feature by displaying the relative direction of a *tag* to a *locator device* in real-time. They are interested to see how useful this *App* is when used as a basis for customers to develop larger Bluetooth Direction Finding systems. Thus I have been tasked to act as a customer which received this sample *App* with the associated hardware and develop a 3D *IPS*. In this way, *Nordic* will also be able to figure out what could be easier, better documented or otherwise different for an improved customer experience. The finished product demonstrates a use of the Bluetooth Direction Finding feature and gives an estimate to the amount of work required of customers to develop such a system with *Nordic* products. Additionally it could serve as a sample *App* available to customers or as an internal reference for *Nordic* to develop other Bluetooth Direction Finding sample *Apps*.

1.2 Research Problem

The research problem is formulated as follows:

> *How to use Bluetooth Direction Finding to create an indoor 3D positioning system*

Bluetooth Direction Finding is still a lesser explored feature and as such there are not as many available projects publicly available, let alone step-by-step guides, and so while this project has quite specific software and hardware requirements, it remains a generic demonstration of how to utilise Bluetooth Direction Finding to create an *IPS*.

1.3 Report Structure

This report closely follows the structure outlined in NTNU TDAT3001's Bachelor's Thesis main report template. The report content's order and a short explanation are detailed in Table 1.

Introduction	Introduction of the project background, the research problem and report structure
Theory	Details the theory utilised for this project
Choice of Technology and Method	Lists and describes the choices of technology and methods used for this project
Results	Presents the scientific, engineering and administrative results of this project
Discussion	Discusses the results, it's advantages and disadvantages in relation to the research problem, the requirements and the choices made
Conclusion and future work	Concludes with the answers to the research problem and project requirements, based on the discussion and results. Includes recommendations for future work with this project.

Table 1: Report Structure

2 Theory

2.1 Spherical coordinates

Spherical coordinates describe a 3D position as specified by 3 values: the polar angle, the azimuthal angle and the radial distance. The definition for the 3 values can change from implementation to implementation but will be similar in use, namely

- Polar angle: Defines the angle from the zenith
- Azimuthal angle: Defines the angle from an axis parallel with the horizontal plane
- Radial distance: Distance from the origin in the direction given by the polar and azimuthal angle

A common alternative to the polar angles exists in the form of an elevation angle. This is the angle from the horizontal plane towards the zenith. It is equal to $90 - polarangle$.

In an environment where the sphere dimensions are static, or where you are only concerned with direction as in our system, the radial distance becomes redundant.

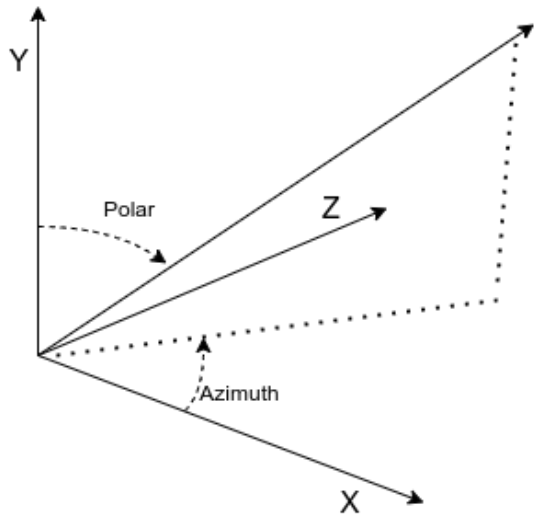


Figure 1: Example of azimuth and polar angle

2.1.1 Spherical coordinates to 3D unit vector

This calculation is taken from *Nordic*'s Direction Finding application [3]. It differs from my implementation only in that all mentions of y and z should be swapped. This is due to my application using positive z as the forward direction of the *locator device* instead of positive y. The calculation is presented with the forward direction being positive y.

To convert the azimuth and polar angle to a 3D unit vector we only need to calculate the x and z components of the vector because we know that the locator always detects devices in the forward/positive y direction.

We illustrate this calculation by building on the example given in Figure 1 but add additional markers for the x, y and z components of the vector, and for a line $a = \sqrt{x^2 + z^2}$.

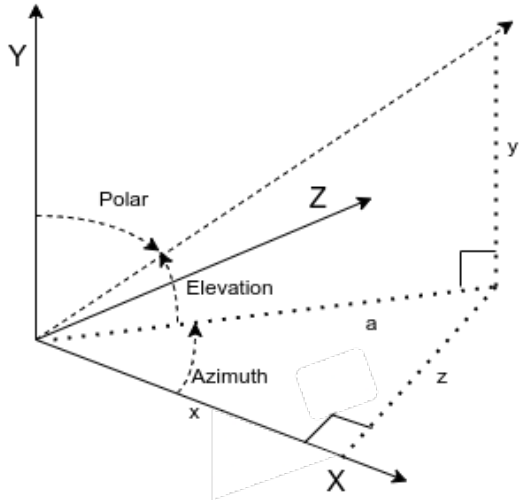


Figure 2: Spherical coordinate system with labeled x, y and z vector components

We denote the azimuthal angle as ϕ and the elevation angle as $\theta = 90\,\text{deg} -$ polar angle. Further, we define $a = \sqrt{x^2 + z^2}$ and find that:

$$\tan\theta = \frac{y}{a}, \cos\phi = \frac{x}{a}, \sin\phi = \frac{z}{a}$$

Because the *locator device* only registers *tags* in the forward direction, positive y in this example, we assume $y = 1$. Thus we get:

$$\tan\theta = \frac{1}{a}$$

We solve this equation for a:

$$(\tan\theta)^{-1} = (\frac{1}{a})^{-1}$$

$$\frac{1}{\tan\theta} = a$$

To solve for x, we substitute a and move $\tan\theta$:

$$\cos\phi = \frac{x}{a} = \frac{x}{\frac{1}{\tan\theta}} = x\tan\theta$$

$$\frac{\cos\phi}{\tan\theta} = x$$

We do the same thing for z:

$$\sin\phi = \frac{z}{a} = \frac{z}{\frac{1}{\tan\theta}} = z\tan\theta$$

$$\frac{\sin\phi}{\tan\theta} = z$$

Now we normalize the vector $\vec{v} = [x, y, z]$:

$$\frac{\vec{v}}{\sqrt{x^2 + y^2 + z^2}}$$

The result is our *unit vector*.

2.2 Nearest points to skew lines

Skew lines is a term used to describe lines in three-dimensional geometry which neither intersect nor are parallel to each other.

This formula makes use of a pair of skew lines, though they can certainly intersect, to determine a pair of points on each line that are closest to each other [4]. We first express the skew lines as vectors, where p denotes the origin, t a scalar value and d the direction:

$$Line1 : v_1 = p_1 + t_1 d_1$$

$$Line2 : v_2 = p_2 + t_2 d_2$$

The cross product $n = d_1 \times d_2$ forms a plane by translating Line 1 and Line 2 along it which contains p_2 and p_1 respectively. n is perpendicular to $n_2 = d_2 \times n$ and $n_1 = d_1 \times n$

The intersection point of Line 1 and Line 2 with this plane is the closest point to Line 2 and Line 1 respectively.

The point on Line 1, closest to Line 2 is given by:

$$c_1 = p_1 + \frac{(p_2 - p_1) \cdot n_2}{d_1 \cdot n_2} d_1$$

The point on Line 2, closest to Line 1 is given by:

$$c_1 = p_1 + \frac{(p_2 - p_1) \cdot n_2}{d_1 \cdot n_2} d_1$$

c_1 and c_2 now form the shortest line segment between Line 1 and Line 2.

By averaging these two points, we get the closest point to both lines.

2.3 Ray casting

Ray casting or ray tracing is a rendering technique mostly utilised for calculating lighting by simulating the path of light rays from a source. This enables a very realistic display of light, as it works by simulating the individual light rays from a light source and observing the possible collisions with other geometry. It can however be quite resource demanding unless used in a more isolated fashion. An example of this can be found in video games, where a single or very few rays are cast to determine whether a line of sight exists between the origin and target coordinates.

2.4 Euler angles

Euler angle is the name for 3 angles which describe an object's rotation within a fixed coordinate system. These angles will rotate the object around its local axes as defined by a chosen sequence like x-y-z or y-x-z.

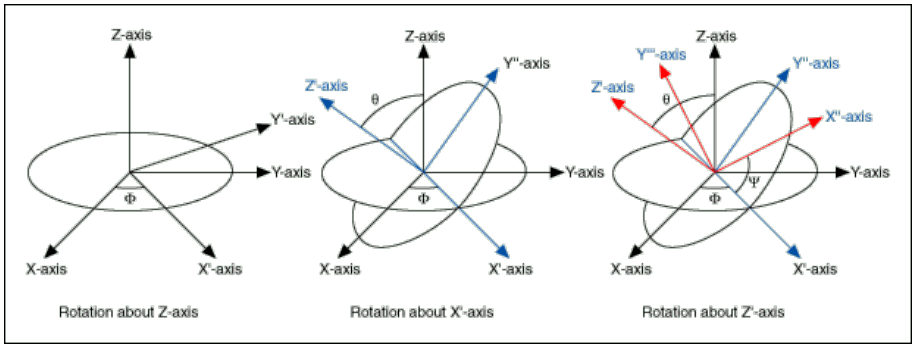


Figure 3: Example of a z-x-z euler rotation [5]

While euler angles suffer from a pitfall referred to as gimbal lock [6], resulting in the loss of a degree of freedom, it does not pose a problem for this project and will therefore only be described briefly. To illustrate how a gimbal lock could manifest, we define an x-y-z euler rotation where x and z values are of no importance. The y value should result in a rotation such that the z axis lines up with the current x axis, letting the z value rotate around the same axis as did the x value and thereby losing one degree of freedom.

2.5 Bluetooth

Bluetooth is an open standard for wireless technology [7].

2.5.1 Bluetooth Direction Finding

Bluetooth Direction Finding is a feature introduced during 2019 in the Bluetooth 5.1 Core specification [2] which allows devices to determine the direction of a Bluetooth transmission. Though this has been done previously through an approximation based on *Received Signal Strength Indicator (RSSI)*, Bluetooth Direction Finding does this with a highly enhanced level of accuracy [8]. This significantly improves previous use cases and introduces new ones, such as *Real-time Location Systems (RTLSs)* and *IPSs* used for asset tracking in warehouses or logistics and indoor navigation.

It is made possible through not only software, but also requires specialised hardware in the form of an antenna array. Due to the distance between the individual antennas an angular phase-shift occurs allowing for the calculation of various values. Out of these values, the polar and azimuth angle are of interest for this project.

Angle of Arrival (AoA) and *Angle of Departure (AoD)* are the two concepts on which Bluetooth Direction Finding is based on. They are very similar in that they require a device with an antenna array, a *locator device*, and a device who's relative direction is to be determined, a *tag*. Though, depending on whether *AoA* or *AoD* is used, the locator will be on the receiving or transmitting side respectively.

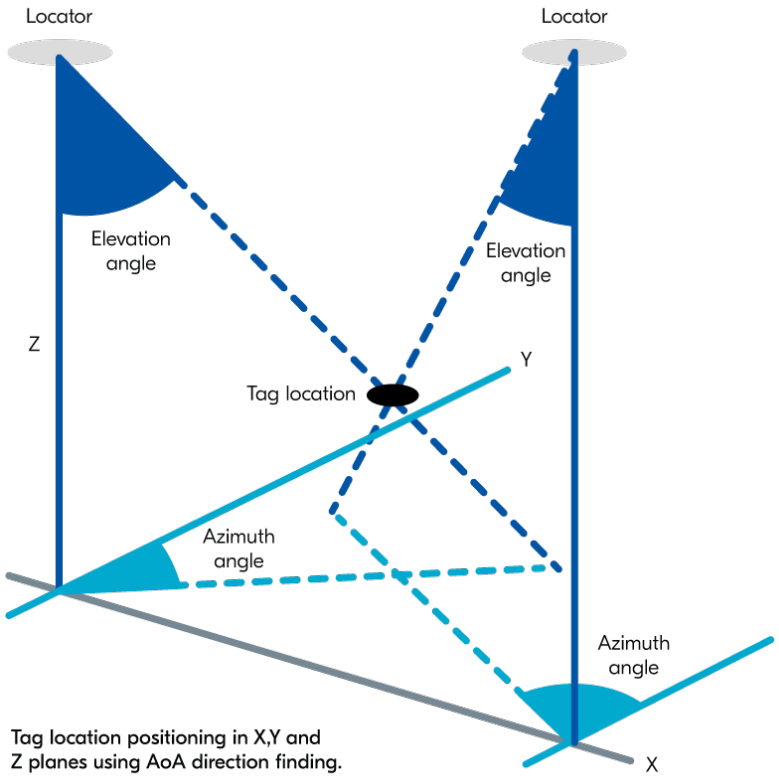


Figure 4: *AoA* Bluetooth Direction Finding with two *locator devices* and a *tag* [8]

Their use case also differs slightly as *AoD* requires more complex *tags* from a hardware and software perspective. Smartphones, though yet to offer Bluetooth Direction Finding support, would be a perfect fit.

3 Choice of Technology and Method

3.1 Hardware

3.1.1 nRF52833 Development Kit

The nRF52833 *DK* [9] is a single-board development kit used for developing firmware applications on the nRF52833 *System on a Chip (SoC)*. It has support for *BLE*, *Bluetooth Mesh* and *Near-field communication (NFC)* applications to name a few but most importantly for this project, it supports Bluetooth Direction Finding.

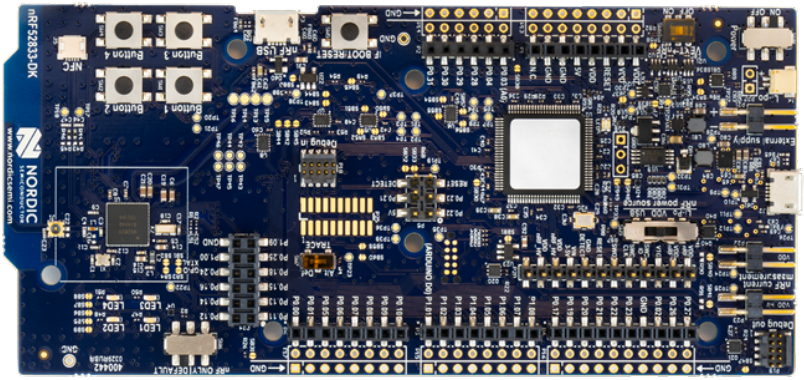


Figure 5: nRF52833 Development Kit

It has a variety of connectors and other hardware features such as *Light-emitting diodes (LEDs)* and buttons. The connectors have an extension out the back of the *DK* used to connect to an expansion board holding an antenna array. Additionally the *Universal Serial Bus (USB)* connector is used to supply the board with power and as the communication medium with the Indoor 3D Positioning application.

3.1.2 Antenna Array Expansion Board

This antenna array expansion board is developed by Nordic Semiconductor and is used in combination with the nRF52833 *DK* to enable support for Bluetooth Direction Finding. It holds 12 individual antennas spread evenly around the frame of the board and a connector for a specialised cable connecting to the nRF52833 *DK*.

3.2 Development process and technology

3.2.1 Development Process

The development process employed for this bachelor project was iterative. It can be compared to scrum though as there was only a singular project member and the project scope was relatively small, a full fledged scrum solution was not appropriate. Nevertheless, some elements from scrum, like sprints and sprint reviews, which naturally came to be due to a weekly meeting schedule, were used for a smooth development process. These meetings included a presentation and evaluation of the previous weeks' work after which tasks to be completed during the next sprint were decided upon. This way, the client was consistently up to date with the project status and changes to features and requirements could be tackled immediately.

3.2.2 Typescript

Typescript builds on JavaScript and lets you specify the shape/type of objects. This allows for better documented code and the early detection of type errors, compared to JavaScript where types have to be inferred from usage and one usually first comes across type errors during execution.

3.2.3 three.js

Three.js is a 3D JavaScript library used to create the 3D view port and objects in the project. This allows for a realistic representation of the system setup with 1 meter in real life being scaled to 1 unit in the view port. Additionally, it allows for a simple way to adjust the perspective based on user needs by rotating in 3D space.

3.2.4 react-three-fiber

React-three-fiber is a helper library for React which wraps three.js object into React components instead of having to define the objects manually and load them into the scene. This makes for more readable and consistent code in a React project.

3.2.5 zustand

Zustand is a state management solution written in JavaScript. While the nRF shared accessories already implement Redux as a state management solution [1] which I can

easily hook into, I ultimately decided to use zustand. This decision was mostly based on an issue regarding the React context in react-three-fiber [10]. While this issue can be resolved with a workaround, zustand works as is and was easier for me to work with than Redux.

3.3 Calculations

3.3.1 Positioning formula

While there were many different options to determine a position, I chose the formula for the nearest point to skew lines, detailed in subsection 2.2. The main alternative in consideration was ray casting. Three.js has an existing implementation of this technique which allows for the detection and locating of *Bluetooth Mesh* collisions. I ultimately decided against the use of this approach for several reasons:

- It requires references to the *Bluetooth Mesh*es to check for collisions. This in turn would force me to store references of relevant *Bluetooth Mesh*es, pass them to their respective component and transfer them to the component running the position calculations. While this wouldn't be difficult to implement, it does decrease the readability of the code.
- It would require the vectors to be represented by cylinders as one dimensional lines cannot guarantee a collision which creates unnecessary geometry.
- It is logical to assume that collisions will be detected on the *Bluetooth Mesh* surface. Due to the cylinder *Bluetooth Mesh*es having a certain radius, the the detected collision coordinates become unusable without further manipulation. That is because we are interested in the distance between the actual vectors, the cylinder center, not any points on the surface. This does not make it impossible to determine a position, but it does require a substantial amount of additional work. Do take note that I have not been able to test and confirm that collision is indeed registered on the surface rather than center.
- Given the offset between vector and surface of the cylinder, as discussed in the above point, determining the error and calibrating the system also becomes more difficult or at the very least requires significantly more work.
- Last but not least, this approach will be more resource intensive due to running similar calculations as the approach above, in addition to using and rendering 3D objects.

It is therefore doubtful whether this approach is worth implementing when the above formula for determining the closest point to skew lines does not have any of these drawbacks.

As this formula only considers 2 vectors at a time, it needs to be run $\frac{n!}{2(n-2)!}$ times, where n is the number of locator devices, before averaging the results.

This calculation is run 60 times per second though one can afford to run it at a lesser interval given the message frequency of the locator device averaging at around 3 times per second. As *locator devices* aren't synced and an additional linear interpolation is applied to the direction data, updating at 60 frames per second simply ensures a smooth and correct display of the position.

3.3.2 Rotation of device graphics

While the rotation of the device graphics seems like a fairly irrelevant feature, it was necessary to have an intuitive way for the user to set these variables. I initially used x-y-z Euler angles, though it quickly became apparent that this was too complex for such a simple feature due to the shift in local axes. While a local axes indicator for each device would be a valid solution to this problem, it does not simplify the actual rotation control and still suffers from a potential gimbal lock (briefly explained in subsection 2.4. Rotating around the local axes one by one was also tested, though was similarly confusing. Another consideration was to use *quaternions*, though they were also too complex and required the user to have an understanding for these.

Finally I settled on a solution that resembles a spherical coordinate system but uses y-x-z Euler angles in the background, allowing for the added rotation around its direction vector (Figure 6). Devices by default had an orientation towards positive z. The y value is comparable to the azimuthal angle, adjusting the orientation in parallel with the horizontal plane, with a range of 0-359 degrees. X became comparable to the elevation angle, which while similar to the polar angle defines the angle from the horizontal plane instead of defining the angle from a fixed *zenith* direction, with ±90 degrees of motion. Z denotes the rotation around the direction vector created by the y/azimuth and x/elevation manipulations.

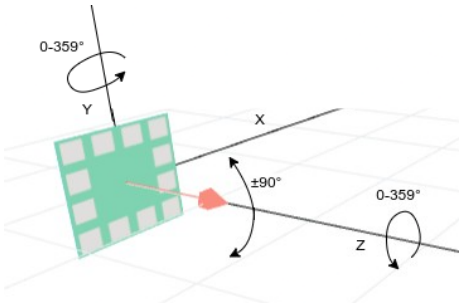


Figure 6: Device rotation

3.3.3 Calibration

The calibration process used for this project is quite simple and the user is guided through these steps by a wizard:

1. Detect which *locator devices* detect which corners based on the coverage angle and their respective position and orientation.
2. Instruct the user to place the *tag* at one of the detected corners and record *locator device* data for a period of time. Repeat for all detected corners.
3. Determine the offset between received and expected data.
4. Save the offset, received data for each device.

During reading the determined offset values are then applied as follows:

1. Determine a scalar based on the distance between the received data and saved received data.
2. Apply this scalar to the offset.
3. Apply the scaled offset to the received data from step 1.

4 Results

4.1 Scientific

4.1.1 Hardware

The hardware is made up of two or more *locator devices* and one *tag*, both of which are *Nordic*'s nRF52833 *DKs* though the *locator devices* include an additional antenna array expansion board each.

The hardware and firmware is used without any modifications, although the firmware must be written to the devices before use. This can be done through the *App*, though only for the *locator devices*.

4.1.2 Calculation

To calculate the 3D position with the direction data received from the *locator devices*, the application follows this formula:

1. The received direction data is first transformed from a spherical coordinate into a 3D coordinate. This is detailed in 2.1.1. This coordinate is used as a *unit vector* as no magnitudes are present.

2. These *unit vectors* are currently pointed in a local direction based on the *locator device*' orientation. We therefore apply the same Euler rotation as is applied to the respective *locator device*. This calculation is performed by the three.js library and results in the direction vectors having values relative to the global axes.

3. The transformed and rotated *unit vectors* are now passed in pairs through the formula detailed in 2.2. The results are averaged for a final 3D coordinate.

4.1.3 Design

The *App* design is based on other nRF Connect for Desktop *Apps*, making use of *Nordic*'s shared React components to have a consistent *Nordic* look.

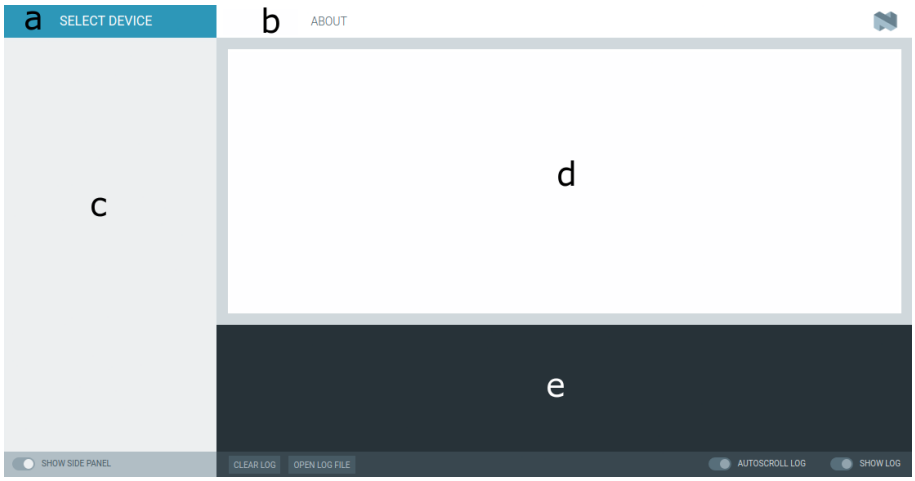


Figure 7: Basic nRF Connect for Desktop *App* made with shared *Nordic* React components (a: DeviceSelector, b: NavBar with ”About” pane and Nordic Logo, c: Sidepanel, d: Main, bottom: Log)

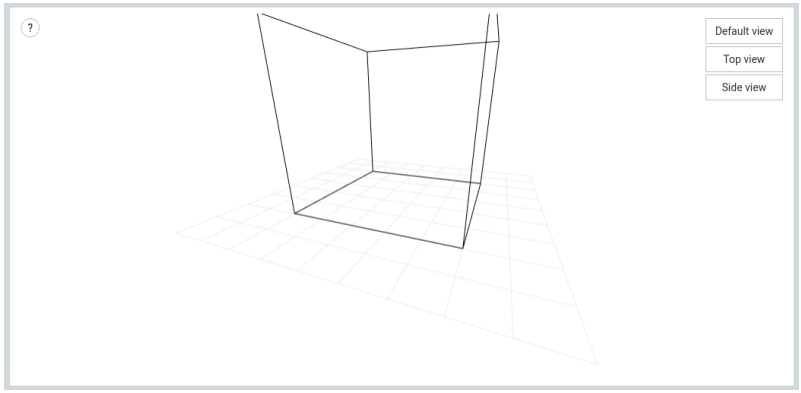


Figure 8: Main view port, Help (?) top-left and perspective presets top-right

In our *App* we have an additional ”Position” pane which displays the 3D view port containing a box, representing the room in which the system is setup, on top a grid which serves as the ground Figure 8. We also have a ”Readings” pane whose use will be discussed towards the end of subsection 4.2.

The Sidepanel (Figure 9) is now filled with settings. We have various toggles whose use will be illustrated later in this section. Additionally the ”Room” section allows the user to edit the length, height and depth of the room box graphic. At the bottom we can see an empty ”Devices” section which will also be explained later in this section. Changes to any of these are immediately reflected within the 3D view port. All the settings displayed here in addition to the device settings detailed later are persisted through the library electron-store and are automatically restored upon startup.

The ”Select Device” option will extend a list of connected devices which we can add to our application.

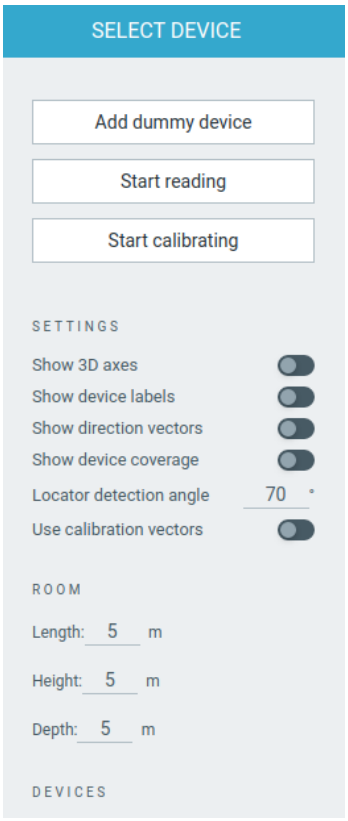
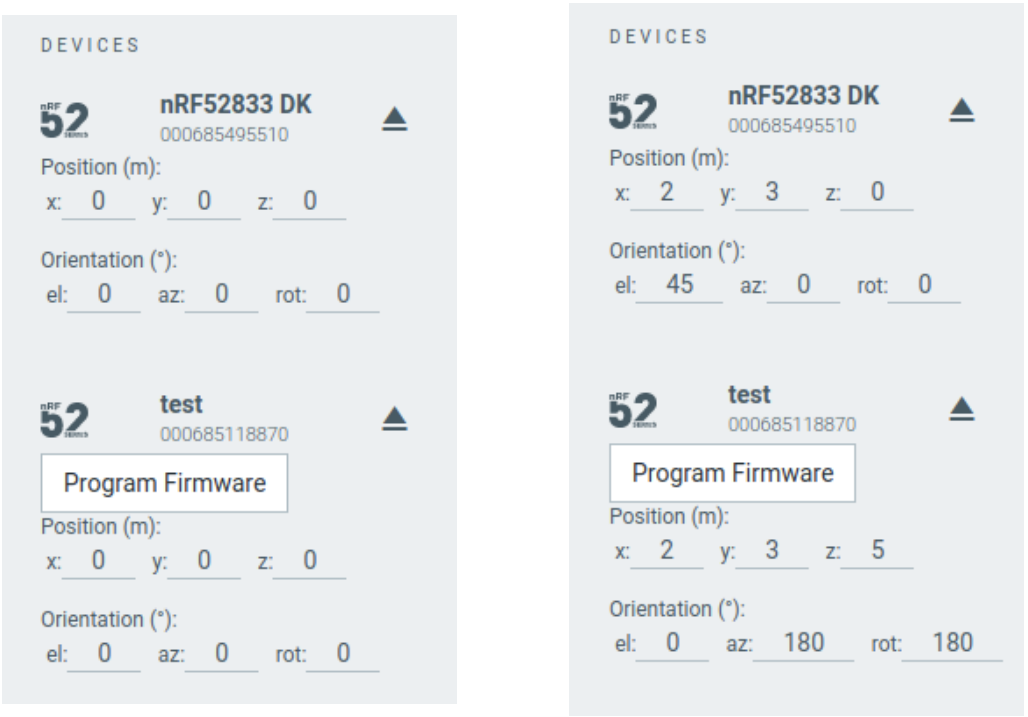


Figure 9: Sidepanel

Figure 10: Display of connected devices

Upon selecting a device, it is added to the Device section in the sidepanel (Figure 11) and also as a graphical element within the view port (Figure 12). Each device has its logo, name and serial number displayed and can be removed by selecting the eject option (right). The edit fields below a device item can be changed to adjust the device position and orientation. Additionally there is the option to ”Program Firmware” for the device called ”test”. This feature detects devices without the expected firmware and allows the user to program them. The red arrows seen in sticking out of the device graphic reveal the device’s forward direction.

The main feature, tracking a 3D position, is made possibly by having two or more *locator devices* connected and selecting the ”Start reading” option. This will display a red dot, indicating the position, with a shadow and two grey horizontal lines for better visual indication of the x, y and z coordinates.



(a) Default device settings

(b) Edited device position and orientation

Figure 11: Device items in sidepanel

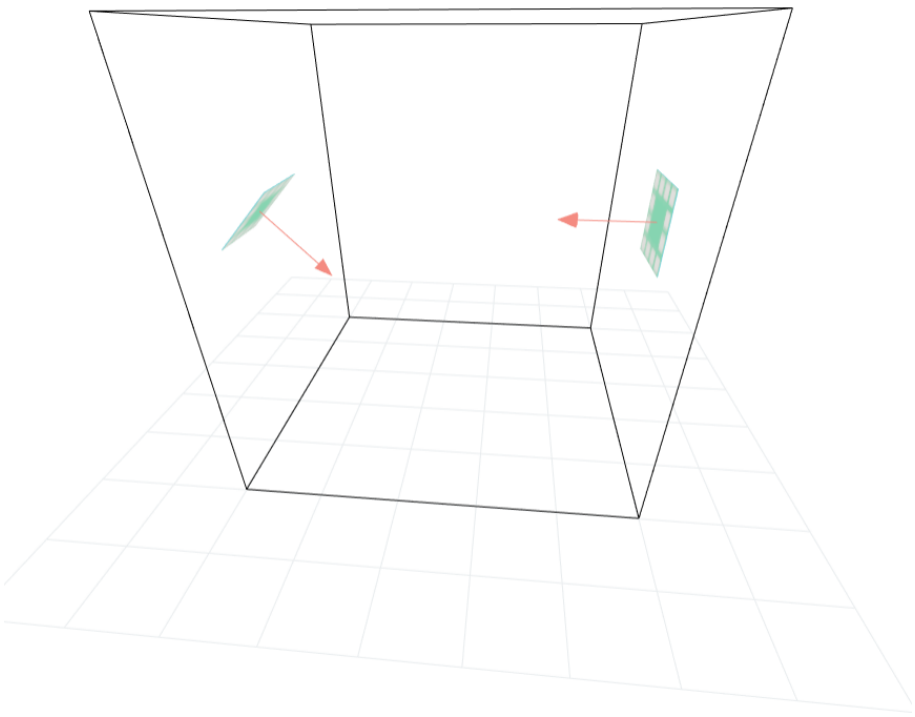


Figure 12: Devices displayed in the 3D view port

Calibration is done by selecting the ”Start calibrating” option. The user is prompted to place the tag at a position indicated by a blinking, blue dot which will always be in one of the bottom corners (Figure 14). Once the tag is placed and the user selects ”Tag placed”

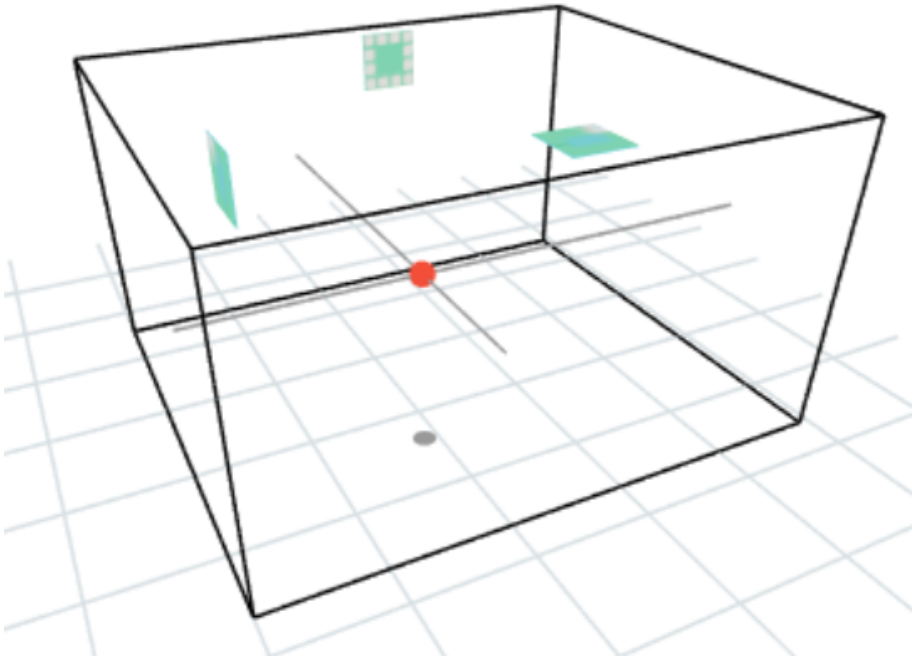


Figure 13: Tracked *tag* position

Place the tag at the indicated position and click the 'Tag placed' button

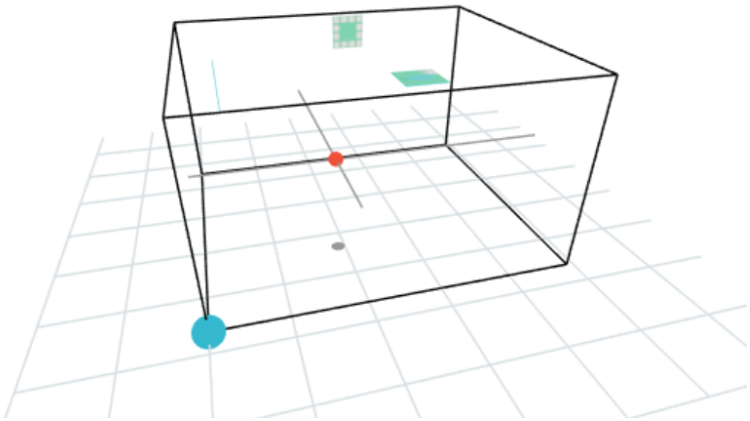


Figure 14: Calibration viewport

(Figure 15), the procedure will be repeated for all corners in the detected area until finally calibration values are calculated from the collected data.

The user can then choose to apply these calibration values when tracking the 3D position via the ”Use calibration vectors” toggle in the Settings section of the sidepanel.

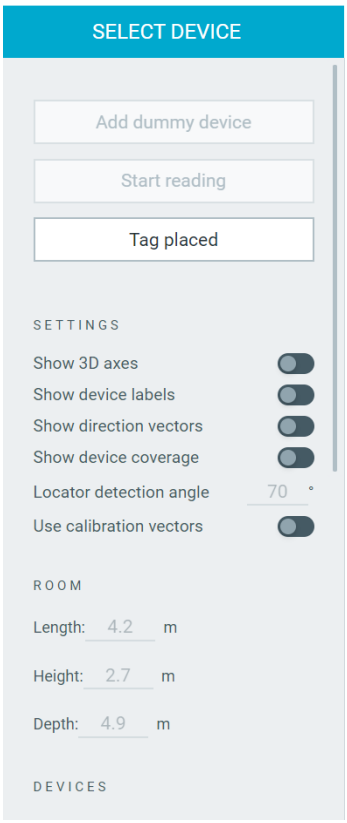


Figure 15: Sidepanel during calibration

4.2 Engineering

The vision document found in Appendix B, lists the required features of this project. Though as there were no specific requirements outside the bare minimum for an indoor 3D positioning system, these have already been covered by the previous chapter detailing the scientific results. As such, I will only cover the additional requirements which were added during the course of the project here. These can be found in the requirement document in Appendix C.

Control of the view port camera can be done by holding and dragging the mouse around or using the w, a, s and d keys. This functionally rotates the geometry around the origin/center. Optionally one can use the 3 buttons to the right to set the perspective. Zooming within a certain range is also possible with the z and x keys. These controls are detailed in a window which can be accessed by selecting the help (?) option (Figure 16). Optionally one can use the perspective buttons or their respective key (1, 2 and 3) (Figure 8).

A reference for the axes can be displayed which helps to position the devices (Figure 17). This can be especially helpful as one rotates around the view port because x and z axes are difficult to tell apart in the default view.

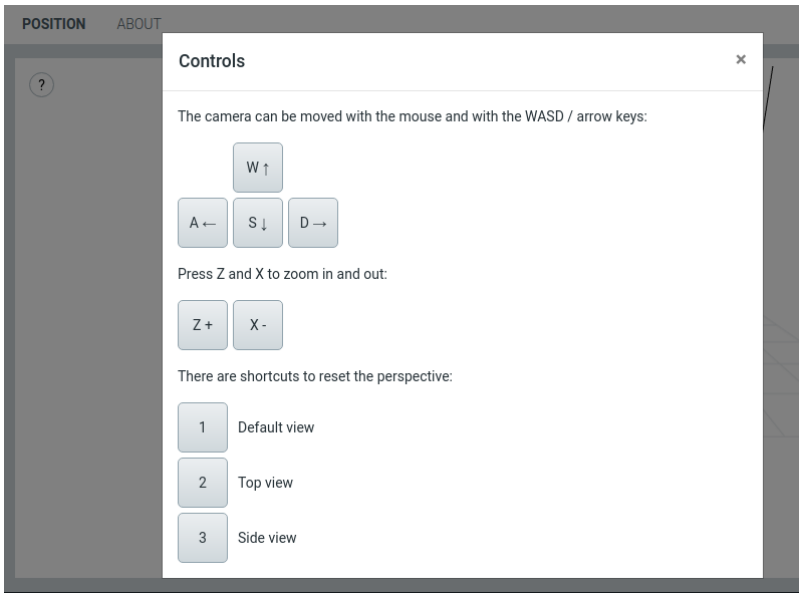


Figure 16: Control help screen

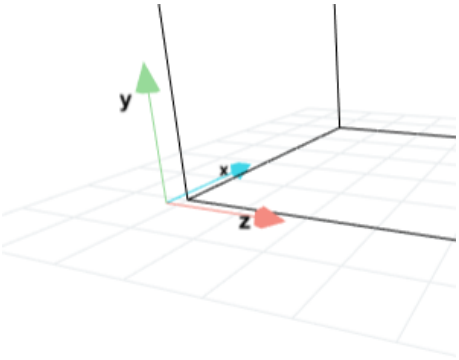
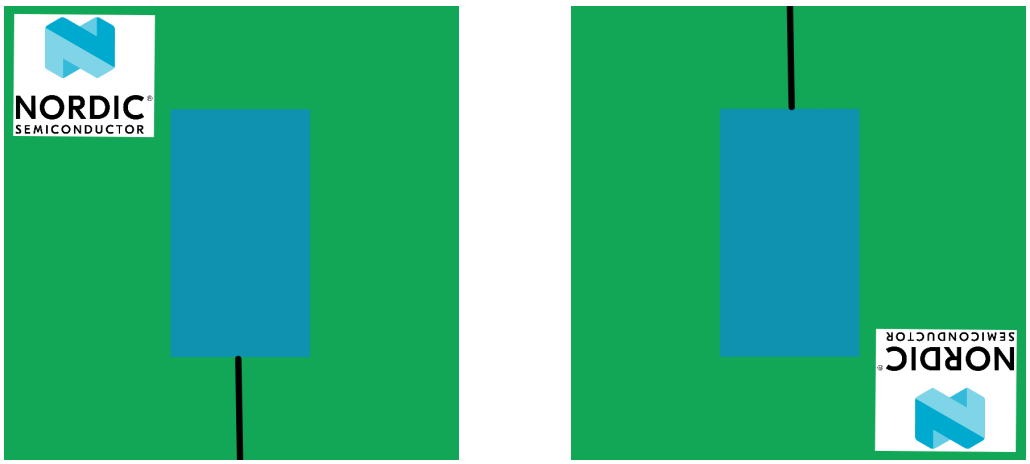


Figure 17: Global axes reference



(a) Back of device graphic with default rotation (b) Back of device graphic with 180° rotation

Figure 18: Back of device graphic

As devices have an ”up” and ”down” direction and can be rotated around their forward direction, it is necessary to have a reference. This is implemented in the form of images in

the devicegraphics showing the rough design (Figure 18). The circuit board, logo and *USB* cable connecting to it are visible. By default the *USB* cable points downwards (negative y).

A rename feature has been implemented for better identification of similar devices (Figure 19). This feature comes with the DeviceSelector component from *Nordic*'s shared React components, but I have added support for the nickname display in the Sidepanel.

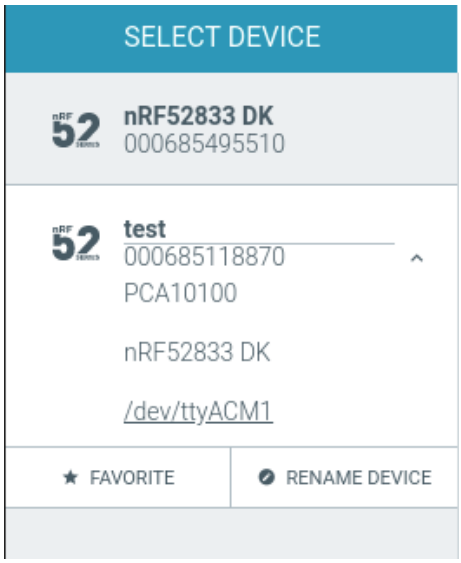


Figure 19: Renaming a device

Additionally one can turn on the label setting which displays the device name above the device graphics in the viewport (Figure 20). This helps with the identification of the device graphics, especially in a larger, more device populated setup.

To give an idea of the coverage of devices, one can choose to turn on the display of coverage. This will replace the red arrow with a semi-transparent cone indicating the space covered by the locator antennas (Figure 21). The cone angle can be adjusted from the sidepanel settings as not all devices will cover the same amount of space.

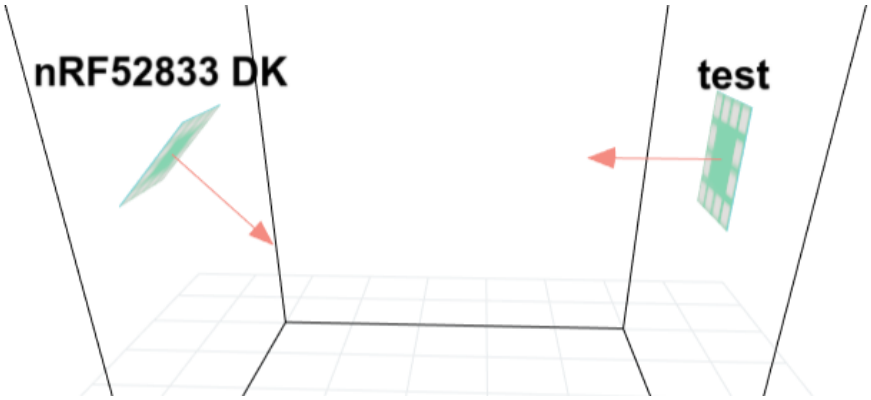
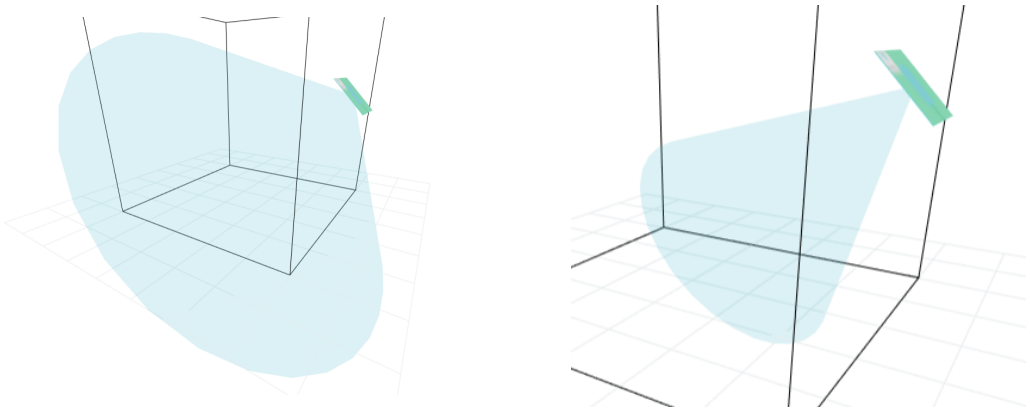


Figure 20: Device graphics with labels

The *locator devices* transmit direction data through serial communication and uses the following configuration:

- Baud rate: 115200
- Parity: None



(a) Coverage indicator

(b) Smaller coverage indicator

Figure 21: Coverage indicator

- Stop bits: 1
- Data bits: 8

The protocol used for the data transmission is quite simple and follows the following pattern:

1. A message starts with the string ”DF_BEGIN” followed by a newline
2. Now various identifiers in the form of two capital letter followed by ”:”, the corresponding value and finally a newline are transmitted
3. The message ends upon receiving the string ”DF_END” followed by a newline

There are only two identifiers which are handled by the *App*, ”KE” and ”KA”. They are the filtered elevation and azimuth values respectively.

It should be noted that there can be a variable number of carriage returns at the end of each line. Therefore instead of comparing for string equality, the *App* checks for inclusion.

A temporary measure implemented during development when only a single device was available was turned into a proper feature. It allows for the visualization of the direction vector of each individual *locator device* by selecting the ”Show direction vectors” option in the sidepanel.

Figure 22: Dummy device

A developer oriented feature was also implemented allowing for the system to be

used and tested without the need of a full
system setup. This is done by adding dummy devices, they act and are displayed like normal *locator devices* except for not being able to detect the *tag* and therefore indicating a static *tag* direction.

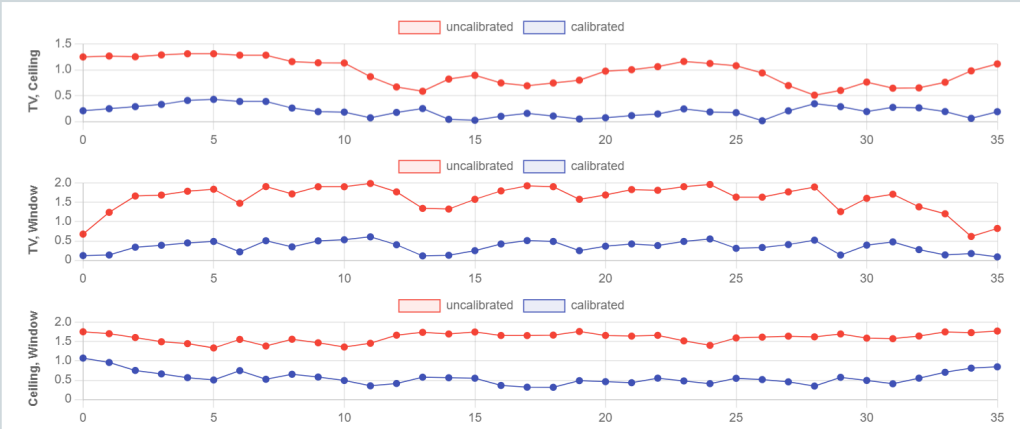


Figure 23: Graph display of statistical difference between uncalibrated and calibrated direction vectors

A developer focused feature implemented at the very end of the project, plots the shortest distance between a pair of vectors (Figure 23). This is used to illustrates the difference in accuracy between uncalibrated and calibrated direction vectors.

The vision document lists a feature which would allow for the tracking of multiple *tags* at the same time. This was not implemented.

To have a closer look at the system or for interest in the exact implementation of any of these features, one should refer to the system document in Appendix D, to get started, or directly head over to the source code (Appendix F).

4.3 Administrative

4.3.1 Project Plan

The project can be roughly divided into 4 sprints of different lengths and can be studied more closely in the project manual in Appendix E:

1. The focus here was to get familiar with the administrative part of the project, mainly by becoming familiar with and preparing various documents. Most of the time was spent on gaining a thorough understanding of the framework and the few core libraries to be utilised for this project.

2. This sprint's goal was to implement the *App User Interface (UI)* and functionality, not including the reading and handling of serial data from the *locator devices*.

3. This sprint focused on implementing the handling and manipulation of serial data from the *locator devices* and to calculate a 3D position. While this was expected to be the most time consuming and difficult task and sprint due to lack of experience, it was implemented quickly and allowed for a lot of time to work on the main report.

4. The last sprint just saw to finishing the system document and main report, as well as minor improvements to the *App* through new features and adjustments. As there was a good amount of time remaining though, calibration was also worked on. Although no requirements regarding the 3D position's accuracy were specified, there was definitely room for improvement despite the 3D position being stable and accurate enough for a demonstration and proof of concept.

4.3.2 Time Management

The amount of hours required for the bachelor project was expected to be at around 500. This requirement was satisfied with the total hours measuring at around 503 hours total. The distribution of hours can be seen from from Table 2 though for a more detailed study of time management, refer to the project manual found in Appendix E.

Activity	Hours used
Documentation	116
Programming	325
Research	43
Meetings	11
Other	8
Total	**503**

Table 2: Time distribution per activity

5 Discussion

5.1 Scientific

5.1.1 Hardware

The hardware to be used was already decided upon before the start of the project and while this may suggest the solution to be non-generic, that isn't the case. While certainly some small adjustments may have to be made for proper integration, any hardware is supported so long as it's data adheres to the protocol used by the current firmware. Apart from the start and end identifier, only an azimuth and elevation value needs to be transmitted as all other values are ignored.

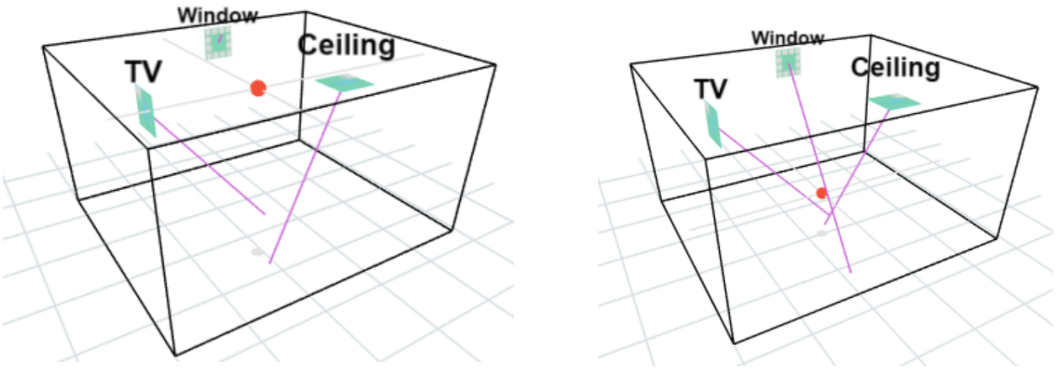
While being a working *IPS* implementation based on Bluetooth Direction Finding and therefore being a valid solution to the research problem, this flexibility of hardware allows one to think of the *locator devices* as *black box*es making it a generically applicable solution.

5.1.2 Calculation

The calculations work as intended and demonstrate that Bluetooth Direction Finding can be used in an *IPS* to determine a 3D position.

That being said, there is certainly a lot of room for improvement which is to be expected given the limited scope, manpower and previous experience of similar systems. Additionally there are many errors sources such as the reliance on user input for both the position and orientation which nearly guarantees an error in calculation by default. Other sources of errors, both intermittent and consistent, mainly originate from hardware instabilities and include but may not be limited to antenna polarization, radio signal reflection and noise.

To combat the consistent error caused by inaccuracies of the user input and stationary objects creating noise or reflecting radio signals, an attempt at a calibration system was made. This was done by having the user place the *tag* at known locations and calculating an offset which would then be used to adjust the incoming direction data. Sadly, this does also require human interaction and will therefore not be as accurate as it could be. To curb this, it was decided to place the *tag* in the room corners, which are easy to reference both by the *App* and the user. Sadly, this has it's own shortcomings as the room shape is assumed to be rectangular shape based on its representation in the *App*. Therefore, a proper implementation of this calibration process would require support for a more complex room representation. Given the goal of this project being a demonstration and



(a) Tracking *tag* position using uncalibrated direction vectors

(b) Tracking *tag* position using calibrated direction vectors

Figure 24: Example of uncalibrated vs calibrated

considering that work on calibration started fairly late without a guarantee to be functional in time, a more complex room implementation was not worked on. Given all these drawbacks, the calibration system did turn out to be beneficial and have a positive impact on the accuracy of the *tag* position. To illustrate this, I will use an example for which the statistical difference has been given during a feature showcase earlier (Figure 23). It is a fairly extreme example making use of the test setup used towards the end of the project (Figure 24). We can see a highly inaccurate direction determined by the ”Window” *locator device*, which I was only able to guess was due to radio signal reflection or noise. Applying the calibration values has a clearly positive effect for all the *locator devices* involved.

A spike filter was also considered, though there was not enough time left. The main goal of this feature was to filter and/or restrict temporary errors caused by noise. This would have been done by comparing the incoming *AoA* to the previous or an average of the last few previous ones. If the new *AoA* would change more than one would expect from a human movement it would either discard or scale it down significantly. This would result in an overall increase in stability of the *tag* position.

5.1.3 Design

The design implementation process was very linear without any memorable problems occurring as functionality was prioritised over the look of the *App*. Therefore the *UI* may feel simplistic at times and not allow for a lot of customizing as for example creating more complex room shapes. Still, it demonstrates a possible presentation of an *IPS* based on Bluetooth Direction Finding.

5.2 Engineering

The requirements given in the vision document, Appendix B, were satisfied as soon as the *App* was able to calculate and display a 3D position. Some additional features, such as calibration and support for multiple *tags*, are mentioned. Calibration was implemented and covered in subsubsection 5.1.2. The multiple *tags* feature was initially thought of as a stretch goal, as it was not supported by the firmware. While it was possible to implement this, *Nordic* had expressed a desire to use the their firmware with no modifications and was therefore not implemented. Also, given the work spent on calibration, there was no time available to work on this feature.

All other required features which were detailed in the requirement document, Appendix C, were satisfied. Though satisfied, some of these features are somewhat lacking. On such feature was the coverage simulation feature which turned out to not be very useful due to a very large space covered which resulted in it being more of a ”visual nuisance” than anything else. Initially, I made an effort to restrict the cone to within the box which would have been a perfect solution. Sadly, none of the available libraries which calculated intersections and other set operations on geometry were outdated and did not display the result properly. Working on my own implementation for this was simply not worth the time for a minor *Quality of life (QOL)* feature.

There are also two known bugs which were not handled. The first is a known bug among nRF for Connect *Apps* which concerns the data transmission to stop almost immediately upon start. This can be handled by refreshing the *App* through the use of CTRL+R. I found that the *locator devices* had to be connected and turned on before starting the *App*, though this is not a confirmed workaround. The second bug also affects another nRF for Connect *App*, the Direction Finding *App* which I used as a reference, though it is fairly minor for this project. I noticed that when focusing an edit field in the side panel and pressing one of the four direction keys (W, A, S, D and the arrow keys) it registered the key down event but not the key up event resulting in a continuous rotation. I was able to circumvent this behaviour by registering when the edit fields were focused and temporarily turning off the perspective controls. Sadly, there is one more edit field which I am unable to interact with directly as it is imported from the *Nordic*’s shared React components, the DeviceSelector. As this component is included through the navigation bar it would require me to redo two entire components for a fix. Given that it is not a major bug which can be fixed with a CTRL+R, to reload the app but apply the rename nonetheless, or by pressing CTRL+(same direction key).

Even with the problems mentioned above, *Nordic* was very satisfied with the result and wants to continue working with the product after the end of the project.

5.3 Administrative

Due to the relative freedom of the task and being the singular developer of the project a full-fledged development process was not implemented. While this allowed for a high degree of flexibility and steady progression, especially given the weekly meeting schedule, it made it impossible to properly document the process. Additionally, there was no documentation done regarding testing. This was due to testing being a heavily integrated part of the development process through manual testing of a live *App* instance after nearly every change.

And so although working alone gives a lot of leeway, as one is up to date with every single detail of the project, and has worked out well, I have experienced the limitations of such an approach and the necessity of a proper development process and documentation. As such, I would implement a formal process in any upcoming projects.

Despite this taken into account, the progress was smooth and steady allowing for quick and easy changes. Involving the client this much in the process was also a major factor for a satisfactory result.

5.4 Own effort and learning

The project has been a great opportunity for learning, both in terms of learning about new concepts and technologies but most importantly for better understanding my own work habits.

React and typescript, while familiar was something I quite enjoyed though never truly felt comfortable with until now. Having to get familiar with other's code was a truly valuable experience as I underestimated it's difficulty initially. Last but not least, it was quite fun to work with a 3D environment.

There is no doubt in my mind that there is a better, more visually appealing solution for many of the features I implemented but that that is not to say that I am any less pleased with the results. While some features were naturally more fun to work on than others, I gave it my all and had mind set on satisfying both the client and my own conscience.

5.5 Ethics

While there are many ethical concern within this field of work, I was luckily not as affected and could work with less concerns regarding this. Covid-19 restrictions were still in place, forcing me to work mostly from home, though I was luckily already quite used

to this beforehand and it was therefore not a serious cause for concern. Though during the end of the project we set up a testing environment within the *Nordic* offices during which certain safety measures like social distancing had to be taken into account. This neither posed a problem and the testing environment was set up easily.

Although no sensitive user data would be handled, when working within *Nordic*'s office I gained access to the internal network. As I was already an employee at the time of the project, a Non-Disclosure Agreement was previously signed, voiding this concern.

6 Conlusion and future work

6.1 Conclusion

Based on the results it is clear that Bluetooth Direction Finding has a lot of potential for being used in an *IPS*. This is especially true when one considers how effective of a demonstration was created given the limitations of time and experience. This project utilised a simple room layout with fewer signal disturbances compared to what one would expect of larger *IPSs*, allowing me to use simpler calculations than would be required of a larger, more complex *IPS*.

Bluetooth Direction Finding is still a fairly young feature and there are not as many products and solutions on the market, but seeing the results it is clear that this will not be the case for long.

6.2 Further work

While being a solid *App* on its own, there is a more than enough room for improvements.

A useful and theoretically simple feature to add is allowing different communication mediums between the *locator devices* and *App*, allowing for more flexibility and removing the necessity for *USB* cables.

Other major improvements, though requiring more work, would be allowing for a more complex room design and a more advanced calibration implementation. The first being more visually oriented though also increasing the complexity of the calibration process. Minimizing the error caused by individual *locator devices* by for example a spike filter may also lead to an increase in average accuracy and stability.

From here on there are many possible *QOL* improvements like adjusting the coverage feature as mentioned in subsection 4.2 or a more intuitive device position/orientation control, to name a few.

If one is to use *Nordic*'s hardware solution, it should be mentioned that the firmware is still actively being adjusted and updated. Though not requiring too much change if the protocol truly changes, it could lead to some confusion. It is also possible to edit the firmware as *Nordic*'s has most of their customer relevant code available on github [11].

Bibliography

[1] Nordic Semiconductor ASA. 'Shared commodities for developing nrf connect for desktop', [Online]. Available: https://github.com/NordicSemiconductor/pc-nrfconnect-shared (visited on 9th Feb. 2021).

[2] Bluetooth Special Interest Group. 'Bluetooth core specification version 5.1 feature overview', [Online]. Available: https://www.bluetooth.com/bluetooth-resources/bluetooth-core-specification-v5-1-feature-overview/ (visited on 7th Feb. 2021).

[3] Nordic Semiconductor ASA. 'Spherical coordinates to 3d unit vector calculation', [Online]. Available: https://github.com/NordicSemiconductor/pc-nrfconnect-directionfinding/blob/master/src/actions/frameActions.ts#L74 (visited on 12th Apr. 2021).

[4] Wikipedia. 'Nearest points to skew lines', [Online]. Available: https://en.wikipedia.org/wiki/Skew_lines#Nearest_points (visited on 28th Mar. 2021).

[5] NI. '3d cartesian coordinate rotation (euler)', [Online]. Available: https://zone.ni.com/reference/en-XX/help/371361R-01/gmath/3d_cartesian_coordinate_rotation_euler/ (visited on 9th Apr. 2021).

[6] Redshift Labs. 'Understanding euler angles', [Online]. Available: http://www.chrobotics.com/library/understanding-euler-angles (visited on 22nd Mar. 2021).

[7] Bluetooth Special Interst Group. 'Bluetooth specification list', [Online]. Available: https://www.bluetooth.com/specifications/specs/ (visited on 20th Apr. 2021).

[8] Nordic Semiconductor ASA. 'Blueooth direction finding', [Online]. Available: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/Direction-finding (visited on 7th Feb. 2021).

[9] ——, 'Nrf52833 dk', [Online]. Available: https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF52833-DK (visited on 10th Feb. 2021).

[10] Poimandres. 'React-three-fiber consuming context from a foreign provider', [Online]. Available: https://github.com/pmndrs/react-three-fiber/blob/master/markdown/api.md#consuming-context-from-a-foreign-provider (visited on 20th Feb. 2021).

[11] Nordic Semiconductor ASA. 'Nordic semiconductor github account', [Online]. Available: https://github.com/NordicSemiconductor (visited on 17th Apr. 2021).

Appendix

A Original Task Description

Arbeidstittel:	Indoor 3D Positioning System

Hensikten med oppgaven:
Create a demonstration of how to track objects/people in 3D space using existing hardware.

Kort beskrivelse av oppgaveforslag:
Use existing azimuth and elevation output from multiple antennas and use this to calculate position in 3D space. Figure out how to position hardware and calibrate system.

Oppgaven passer for (kryss av de(t) som passer og skriv evt. en kommentar til oss):	- Bacheloroppgave
Skal oppgaven utføres av bestemte studenter? (der avtalt) Fyll i så fall inn studentenes navn	Pascal Pickel
Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:	Ja
Oppgaven passer best for, antall studenter:	- 2 - 3
Opplysninger om oppgavestiller	
Er du fra bedrift/virksomhet eller er du student med en egendefinert/selvlaget oppgave?	- Bedrift/virksomhet
Navn på bedrift/organisasjon/student:	Nordic Semiconductor ASA
Addresse	Otto Nielsens veg 12
Postnummer	7052
Poststed	Trondheim
Navn på kontaktperson/veileder:	Ketil Erichsen
Telefon:	47088747
Epost:	ketil.erichsen@nordicsemi.no
Navn på kontaktperson 2/veileder 2:	Nicolai Berthelsen
Epost kontaktperson 2/veileder 2:	Nicolai.Berthelsen@nordicsemi.no

Utfyllende kommentarer til hva oppgaven gjelder:
Direction Finding makes it possible to locate objects in 3D space. Nordic has created hardware and software to enable customers to create complete systems. The task is for the students to act as customers and create a complete system for direction finding.

The main task will be understanding/describing the system and writing a software application. The application will be written in JavaScript on the React framework.

For interested students there might also be a task related to digital signal processing.

B Vision Document

022

Indoor 3D Positioning System Vision

Version 1.0

Revision History

Date	Version	Description	Author
16/jan/2021	0.1	Started chapter 1, 2,and 3	Pascal Pickel
20/jan/2021	0.1	Finished chapter 1 and 3 Starter chapter 4 and 5	Pascal Pickel
22/jan/2021	0.1	Finished first draft	Pascal Pickel
25/jan/2021	1.0	Language change	Pascal Pickel

Table of Contents

1. Introduction

1.1 Purpose

This documents outlines the vision for a indoors 3D positioning system. The purpose of the document is to:

- Identify stakeholders and users
- Identify and describe the problem
- Propose a solution
- Specify requirements and constraints for the solution

1.2 Definitions, acronyms and abbreviations

AoA – Angle of Arrival
Locator – Stationary unit that uses an antenna array to derive the AoA (and others)
Tag – Movable unit whose relative direction/position is being tracked by (a) locator(s)
nRF Connect for Desktop – A launcher for the application of this system (and others)

1.3 References

https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-desktop
https://www.nordicsemi.com/Products/Low-power-short-range-wireless/Direction-finding

2. Positioning

2.1 Problem statement

The problem of	lack of sampleproducts for an indoor 3D positioning system based on Bluetooth Direction Finding
affects	customers
causing the impact of	customers having to develop their own solutions without reference material
A successful solution would provide	a nRFConnect for Desktop app that displays the real-time 3D positiong of an AoA tag in an area covered by a number of AoA locators. The app calibrates the system to allow tracking in the customers unique environment.

2.2 Product position statement

For	customers
who	wish to use an indoor positioning system or to test out the capabilities of Bluetooth Direction Finding
The indoor 3D positioning system	is a software product
that	by using AoA locators and a calibration process, displays the real-time 3D position of an AoA tag inside a room.

3. Stakeholder and user description

3.1 Stakeholder summary

Name	Description	Responsibilities
Nordic Semiconductor	Client	Decides the requirements and requirements of the product
NTNU	Supervisor and publisher	Gives feedback on the process, documentation and scope of the project

3.2 User summary

Name	Description	Responsibilities	Stakeholder
Business customers	End user	Uses the product to see capabilities of Bluetooth Direction Finding and as a reference design for further development	Self
Private customers	End user	Uses the product as a reference design for further development	Self

3.3 User environment

- 3(+) AoA locators are placed so they cover a common area
- Aoa locators are connected to the computer running nRFConnect for Desktop and the positioning app
- When the AoA tag is within the area covered by the locators, its real-time 3D position will be shown in the app

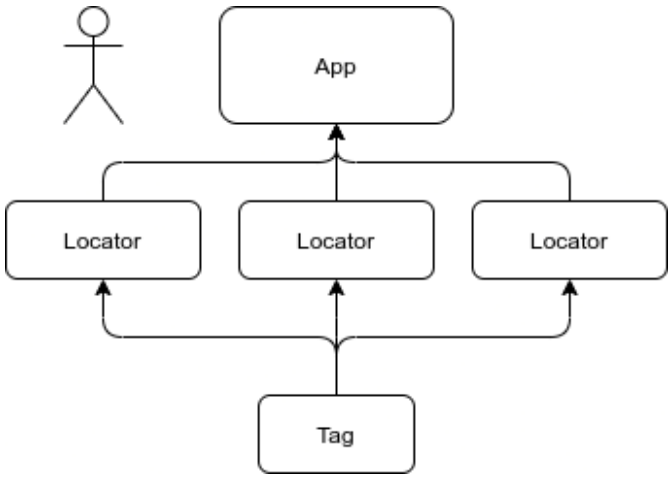
3.4 Key stakeholder or user needs

Need	Priority	Concerns	Current solution	Proposed solutions
Easy to use	High	The ability for users without knowledge of the system to set it up , test and use it	None	A plug and play experience by necessary firmware being pre-installed on the units and an intuitive app with clear

				instructions for the use of the system
Easy to calibrate	High	The ability for the user to use the system without needing exact and manual measurements of the setup	None	The user is instructeed to place the tag in positions relative to the locators. An algorithm calculates the position and orientiatopm of the locators.
Scalable	Low-Medium	The ability to use the system as is with more locators and tags	None	Algorithm and data that supports a dynamic number of tags (1+) and locators (3+)

4. Product overview

4.1 Product perspective



5. Product features

5.1 App features

- Start application
- Stop application
- Add locator units
- Remove locator units
- Start calibration
- Cancel calibration
- Select 2D/3D perspective

6. Other product requirements

6.1 System requirements

The system must be able to run nRF Connect for Desktop and have connectivity options for the locators.

C Requirement Document

22

Indoor 3D positioning system Requirements

Versjon 1.0

Revision History

Date	Version	Description	Author
29/01/2021	0.1	Added user stories and surface level sequence diagram	Pascal Pickel
19/02/2021	0.1	Added wireframes	Pascal Pickel
25/02/2021	0.1	Add more user stories	Pascal Pickel
01/03/2021	0.1	Add and edit user stories	Pascal Pickel
19/03/2021	1.0	Add supplementary specifications	Pascal Pickel

Table of Contents

1. Introduction

This document specifies the requirements to be met by the system and describes it's interactive options.

2. Requirements

2.1 User Stories

2.1.1 User

As a user
I wish to select connected devices
So that the app sets up and uses the correct devices

- If I am a user, I can select which devices to use
- I can add connected devices by choosing "Select device" and selecting my device from a list
- I cannot select devices that are incompatible with the firmware or are already selected
- I am prompted to install the correct firmware upon selection or cancel the selection process

As a user
I wish to remove devices from use
So that unwanted device data is not being used by the app

- If I am a user, I can remove devices from the «connected device» list
- I can remove a specific connected device by choosing the «eject» option beside it's entry in the list

As a user
I wish to calibrate the system
So that the tags correct 3D position can be obtained

- If I am a user, I can calibrate the system
- I can start the calibration process by selecting the «calibration» option
- After starting the calibration process, the app instructs me on how to complete the calibration progress

As a user
I wish to see the 3D position of the tag in a specific perspective
So that I can better see the position based on my needs

- If I am a user, I can change the perspective
- I can change the perspective by clicking and dragging the 3D view or selecting template persectives with the appropriate UI buttons or keyboard keys
- I can see the appropriate keyboard keys to control the perspective by selecting the help («?») button

As a user
I wish to set the room dimensions
So that I can view a representation of system environment with accurate proportions

- If I am a user, I can set the room length, height and depth
- I can change the dimensions by focusing the appropriate field and entering the desired values

As a user
I wish to set the coverage angle as seen from the device
So that I have an accurate representation of the devices coverage

- If I am a user, I can set the device coverage angle
- I can change the device coverage angle by focusing the appropriate field and entering the desired value

As a user
I wish to see a simulation of the coverage of a device
So that I can view a representation of the coverage in my environment

- If I am a user, I can toggle the «show device coverage» option
- I can toggle the «show device coverage» option by selecting it
- When toggled on, I am able to see the coverage within the room based on the devices position and angle therein

As a user
I wish to change the device name
So that I can identify devices easier and differentiate similar devices without the use of serial numbers
- If I am a user, I can edit the device name and enter a custom nickname
- I can select the device name and edit the text

As a user
I wish to see the device labels in the 3d viewport
So that I can accurately identify the devices
- If I am a user, I can toggle the «show device labels» option
- I can toggle the «show device labels» option by selecting it
- When toggled on, I am able to see the the device name or nickname hover above the appropriate device in the 3d viewport

As a user
I wish to see the viewport axes
So that I can easily determine the orientation and edit the device position
- If I am a user, I can toggle the «show axes» option
- I can toggle the «show axes» option by selecting it
- When toggled on, I am able to see the the global 3d viewport axes as arrows

As a user
I wish to avoid having to reconfigure the system settings after restarting the app
So that I can use my previously used setup without spending time reconfiguring known information
- If I am a user, I can start the application and continue using the settings I had set during the last session
- If I do not have previously connected devices connected, they will not be imported from the last session

2.1.2 Developer

As a developer
I wish to add dummy devices
So that I can use and test the application without the need of a physical setup
- If I am a developer/user, I can add a dummy device to the application
- I can add a dummy device by selecting the “Add dummy device” option

As a developer
I wish to see the difference between raw and calibrated vectors
So that I can easily troubleshoot and test changes to the reading/calibration algorithm
- If I am a developer/user, I see the raw vectors and raw position displayed alongside the calibrated version
- I can view the raw and calibrated vectors/position by having the “Show direction vectors”, “Use calibration vectors” and “Show raw vectors” options selected.

As a developer
I wish to see the difference between raw and calibrated vectors statistically
So that I can easily troubleshoot and test changes to the reading/calibration algorithm
- If I am a developer/user, I can see the respective error between a pair of vectors in a graph
- I can view the real-time graph by selecting the “Readings” pane

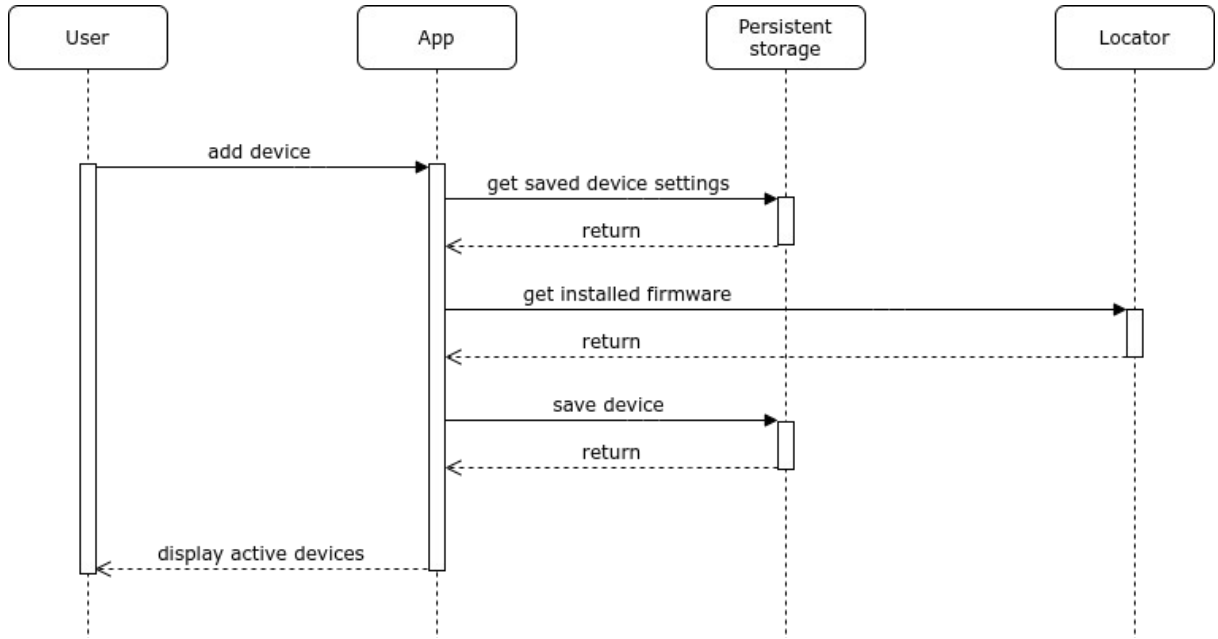
2.2 Supplementary Specifications

- 3+ locator devices. These are expected to be nRF52833 DK with an antenna array from Nordic Semiconductor ASA, whose firmware comes with the Indoor 3D Positioning application.
- A tag device which is also expected to be a nRF52833 DK from Nordic Semiconductor ASA. The firmware for this must be requested from Nordic Semiconductor ASA.
- A USB cable and power supply for the tag device as the lithium battery shipped with the device does not meet the energy demands of the firmware.
- nRF Connect for Desktop version 3.7.0
- The system running the Indoor 3D Positioning application needs as many usb ports as there are locator devices.
- USB cables to connect the locator device to the system running the Indoor 3D Positioning application.

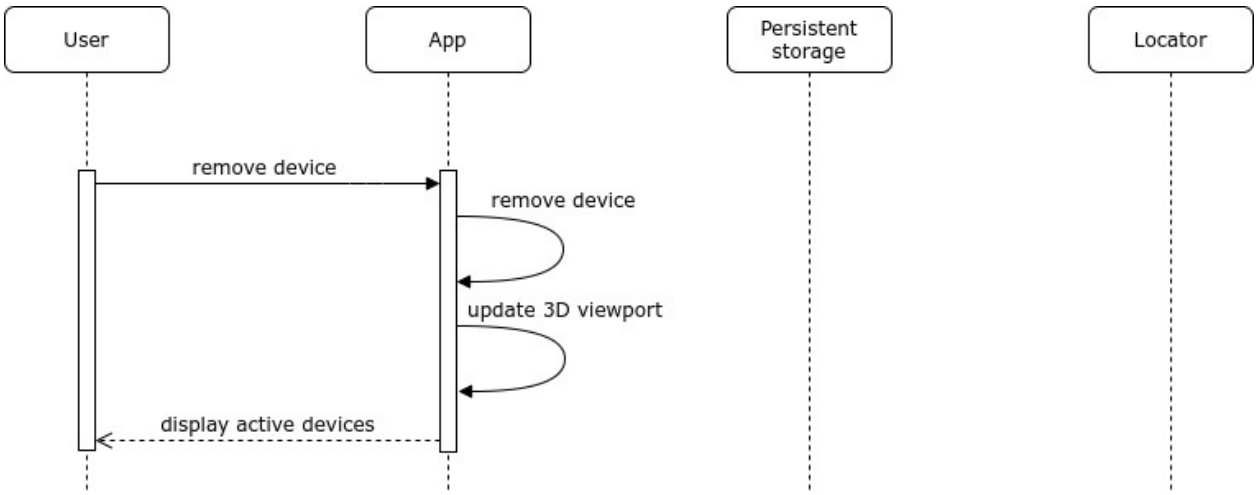
3. Domain Model

3.1 Sequence diagram

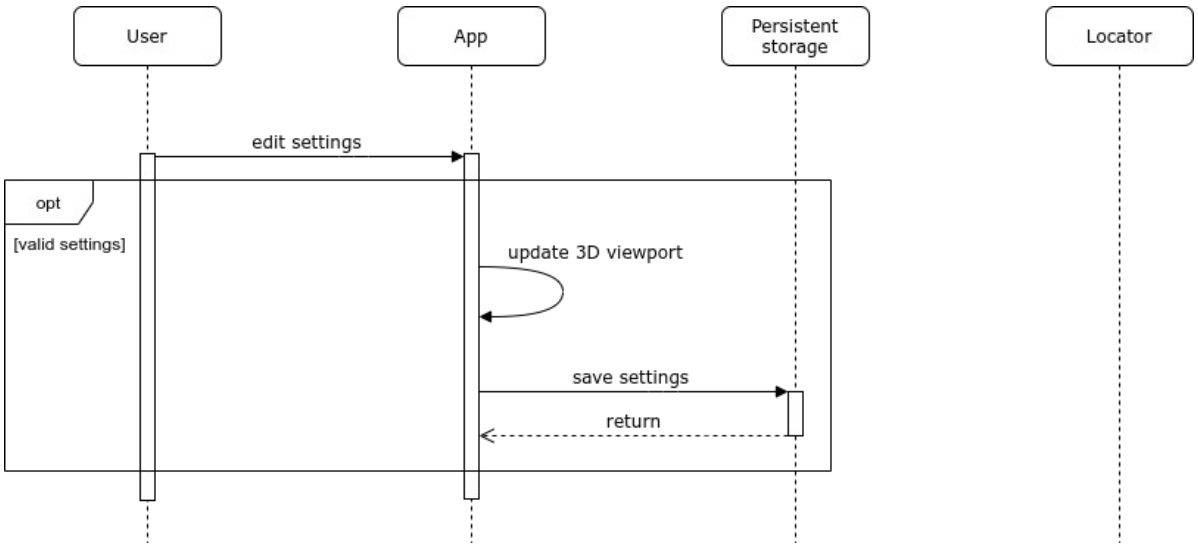
3.1.1 User adds devices



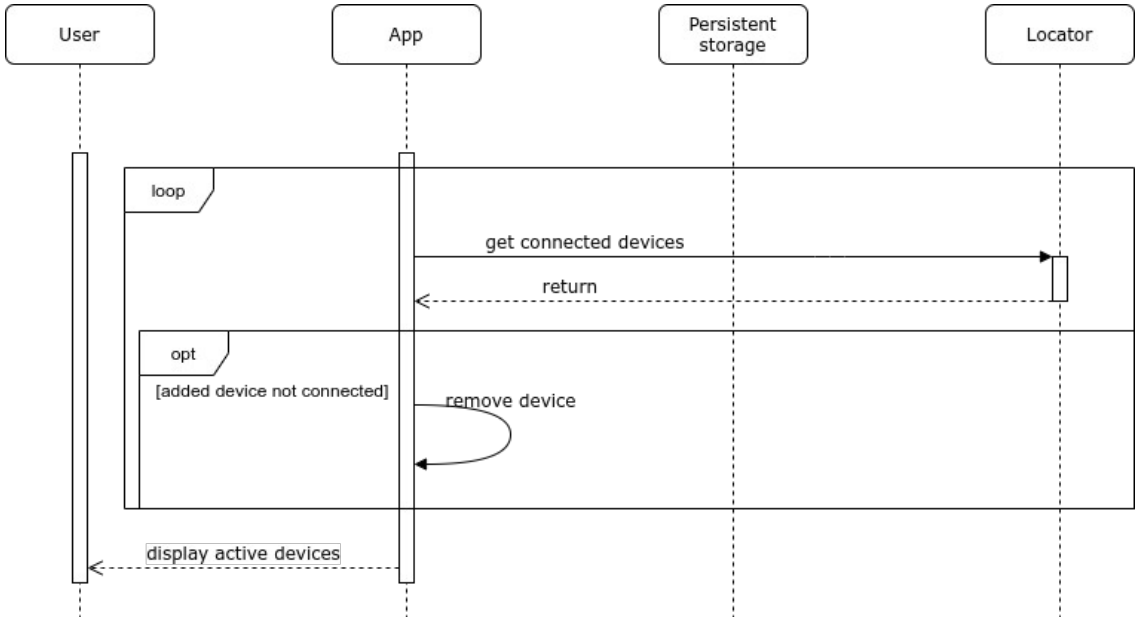
3.1.2 User removed devices



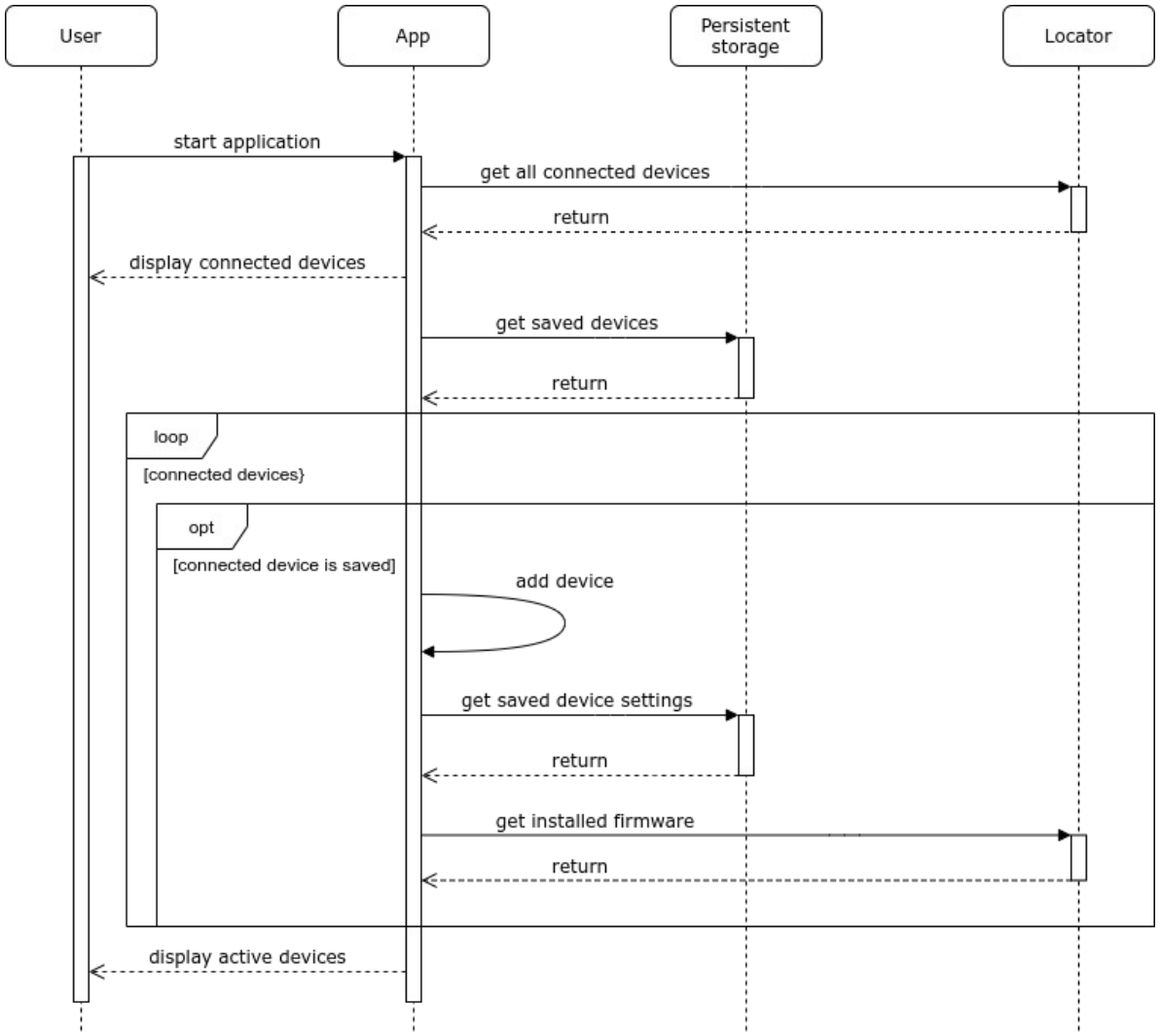
3.1.3 User edits settings



3.1.4 App detects and removes disconnected devices



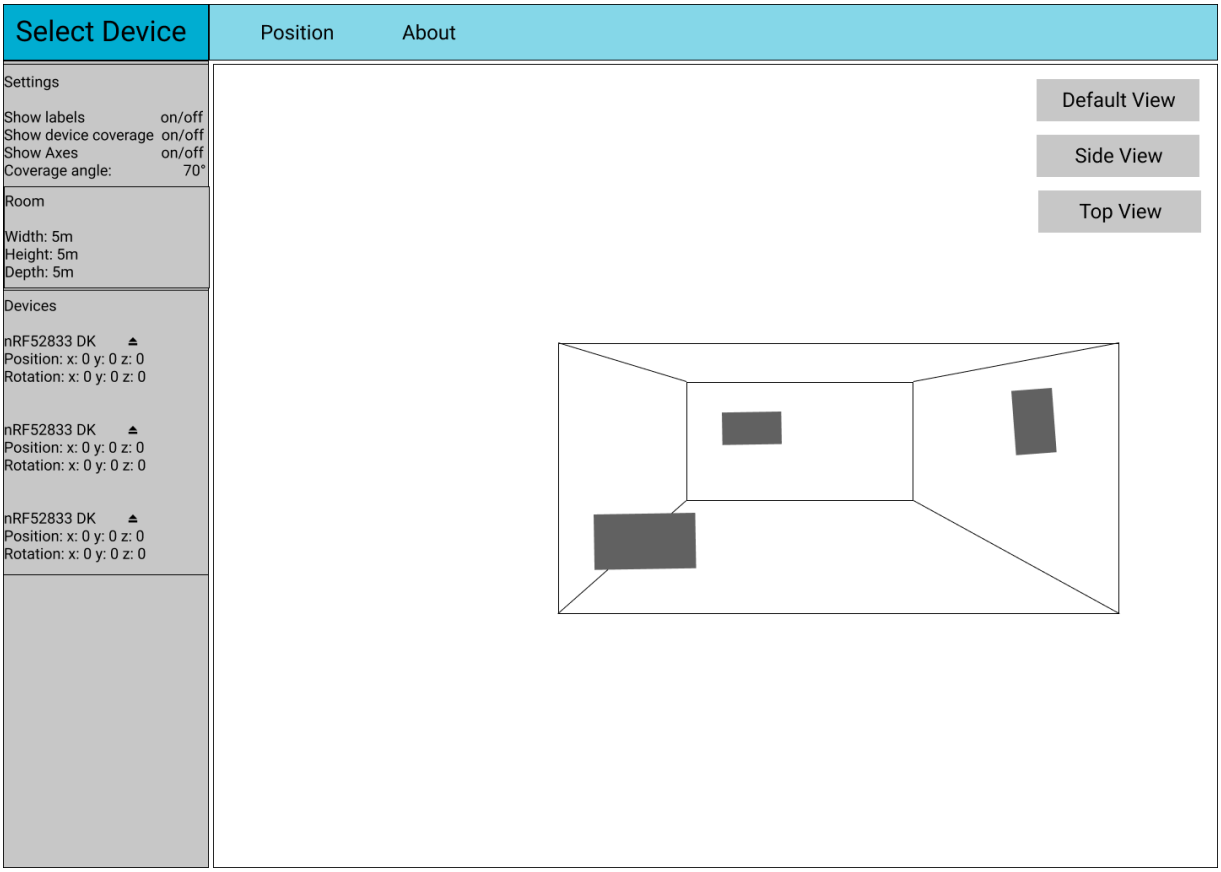
3.1.5 App detects and adds saved devices on start



4. Prototypes

4.1 Wireframes

Main screen in position pane with selected and set up devices:



The “Select Device” button expands down the entire height to show a list of valid, connected and unselected devices.
The “About” button switches the viewport to show some info about the app. It is a default pane autotimatically created by the shared assets for nRF Connect apps.
The “Default View”, “Side View” and “Top View” will switch the canvas perspective.

D System Document

22

Indoor 3D Positioning System
System Documentation

Version 1.0

Revision history

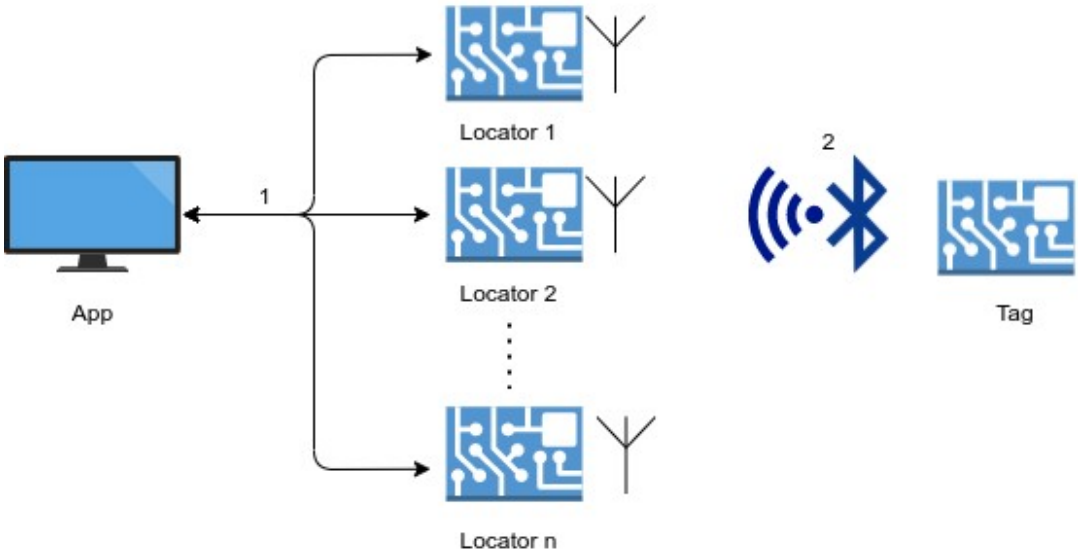
Date	Version	Description	Author
03.04.2021	0.1	Added chapter 1-6	Pascal Pickel
14.04.2021	0.1	Edited chapter 3 and added reference to source code in chapter 5	Pascal Pickel
15.05.2021	1.0	Finishing touches and added repository link	Pascal Pickel

Table of Contents

1. Introduction

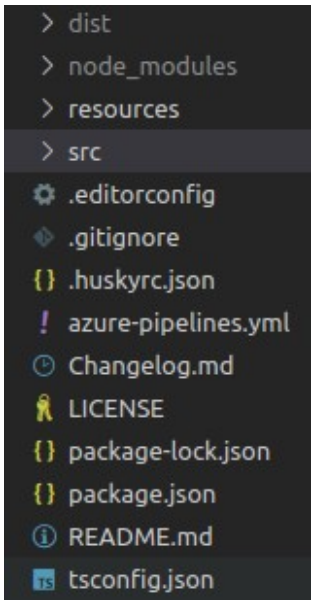
This document serves as a description of the Indoor 3D Positioning system by outlining everything the overarching architecture, project structre, source code documentation and installation manual.

2. Architecture



The system is best described from right to left. We have a tag sending Bluetooth packets (2) consistently. These packets are received by a number of locator devices which determine the relative direction of the packet. The relative direction is then sent to the Electron/React app through serial communication using USB cables (1) as a medium.

3. Project structure



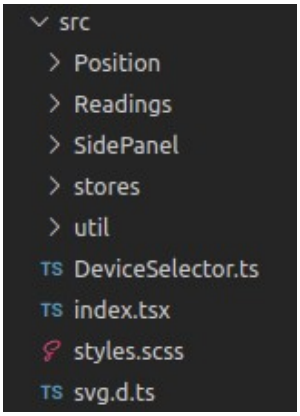
./dist: Contains the bundle.js created by npm (node package manager) used by Electron

./node_modules: Contains the files of dependencies imported by npm

./resources: Contains none-source code files such as images and the locator firmware hex file

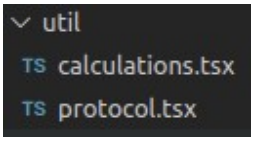
./src: Contains the source code files consisting of typescript and sass files

./: Various configuration files by git, code editors, npm and typescript

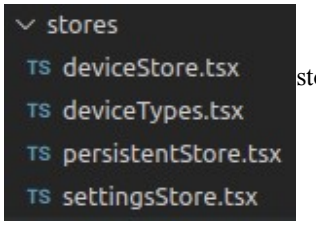


Within the src folder we have various groupings of React components which will be explored below.

Addtionally we can find the main react component within index.tsx, The DeviceSelector component, a sass styles file and a svg webpack declaration.

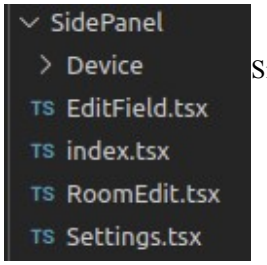


The util folder contains some common and isolated functions for the handling and manipulation of the locator device data stream



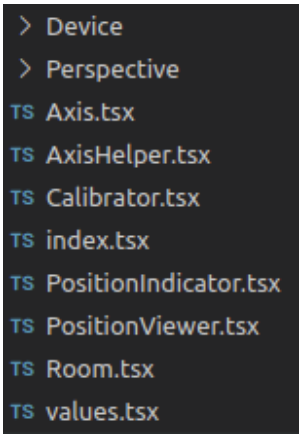
The stores folder houses the zustand stores in addition to a persistent storage and a small file for a type declaration to avoid a circle dependency issue

deviceStorage: State management code responsible for devices.
settingsStorage: State management code responsible for settings.
persistentStore: Code responsible for persisting settings and devicedata



The SidePanel folder holds all React Component Relevant to the SidePanel. This includes various edit fields as well as settings.

There is an additional grouping for various Device related files.



The Position folder holds all files related to the view port.

There are additional groupings for the Device and Perspective control related files.

4. Installation and execution manual

Install nRF Connect for Desktop launcher dependencies:
https://nordicsemiconductor.github.io/pc-nrfconnect-docs/getting_started#install-development-tools
https://nordicsemiconductor.github.io/pc-nrfconnect-docs/core_development#prerequisites

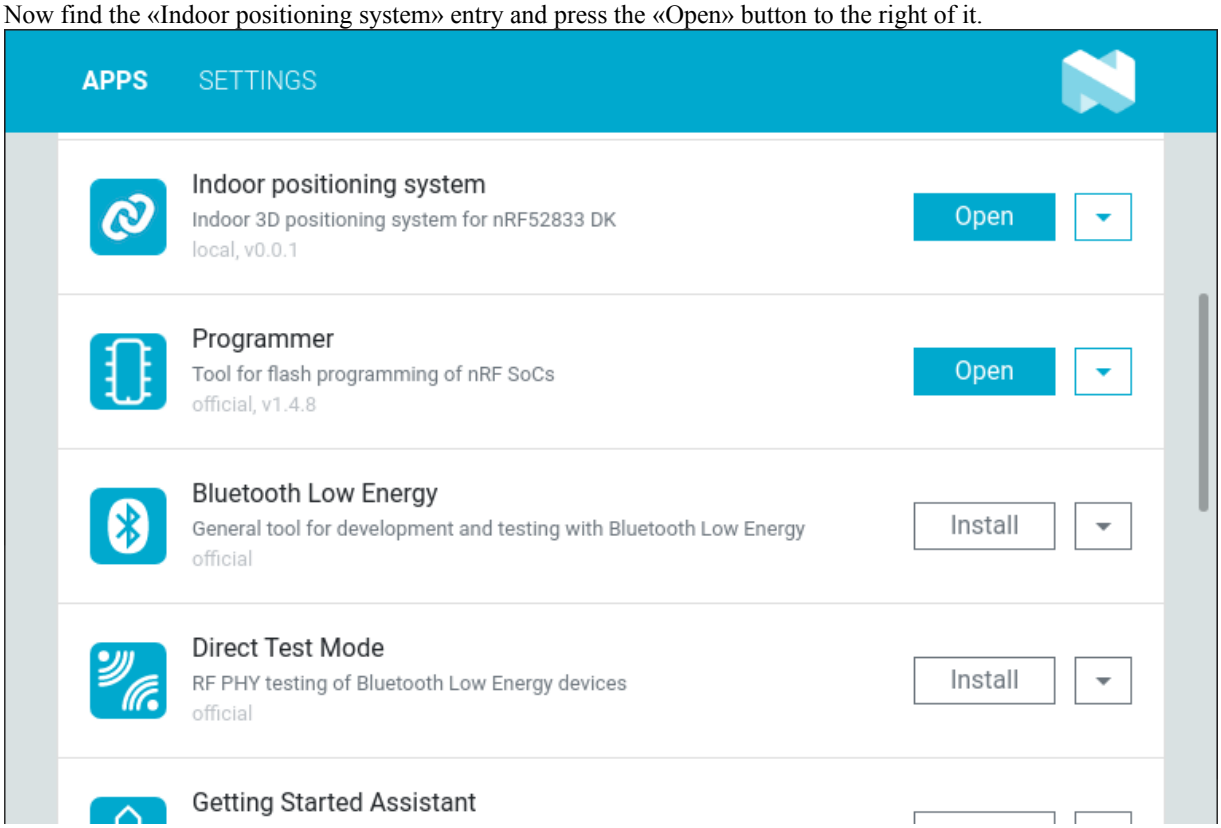
Clone the nRF Connect for Desktop repository by running the command «*git clone https://github.com/NordicSemiconductor/pc-nrfconnect-launcher.git*»

Go to the correct directory mentioned in
https://nordicsemiconductor.github.io/pc-nrfconnect-docs/get_an_existing_app_s_sources

If publicly available: Clone the Indoor 3D positioning system repository by running the command «*git clone https://github.com/PascalPickel/pc-nrfconnect-indoorpositioning.git*»
Else, unzip the code.zip into the above directory

The nRF Connect for Desktop launcher and Indoor 3D positioning application has to be compiled before it can be run (Described in detail here:
https://nordicsemiconductor.github.io/pc-nrfconnect-docs/app_development#compiling)
Go to Indoor 3D positioning application directory and execute the command «*npm run buid*»
Go to the nRF Connect for Desktop launcher directory and execute the command «*npm run build*» followed by «*npm run app*»

Now find the «Indoor positioning system» entry and press the «Open» button to the right of it.



5. Documentation of source code

The source code was written in typescript on the React framework. Typescript acts as the source code

documentation.
The source code can be found attached to the main report as a zip.file (code.zip)

6. Testing

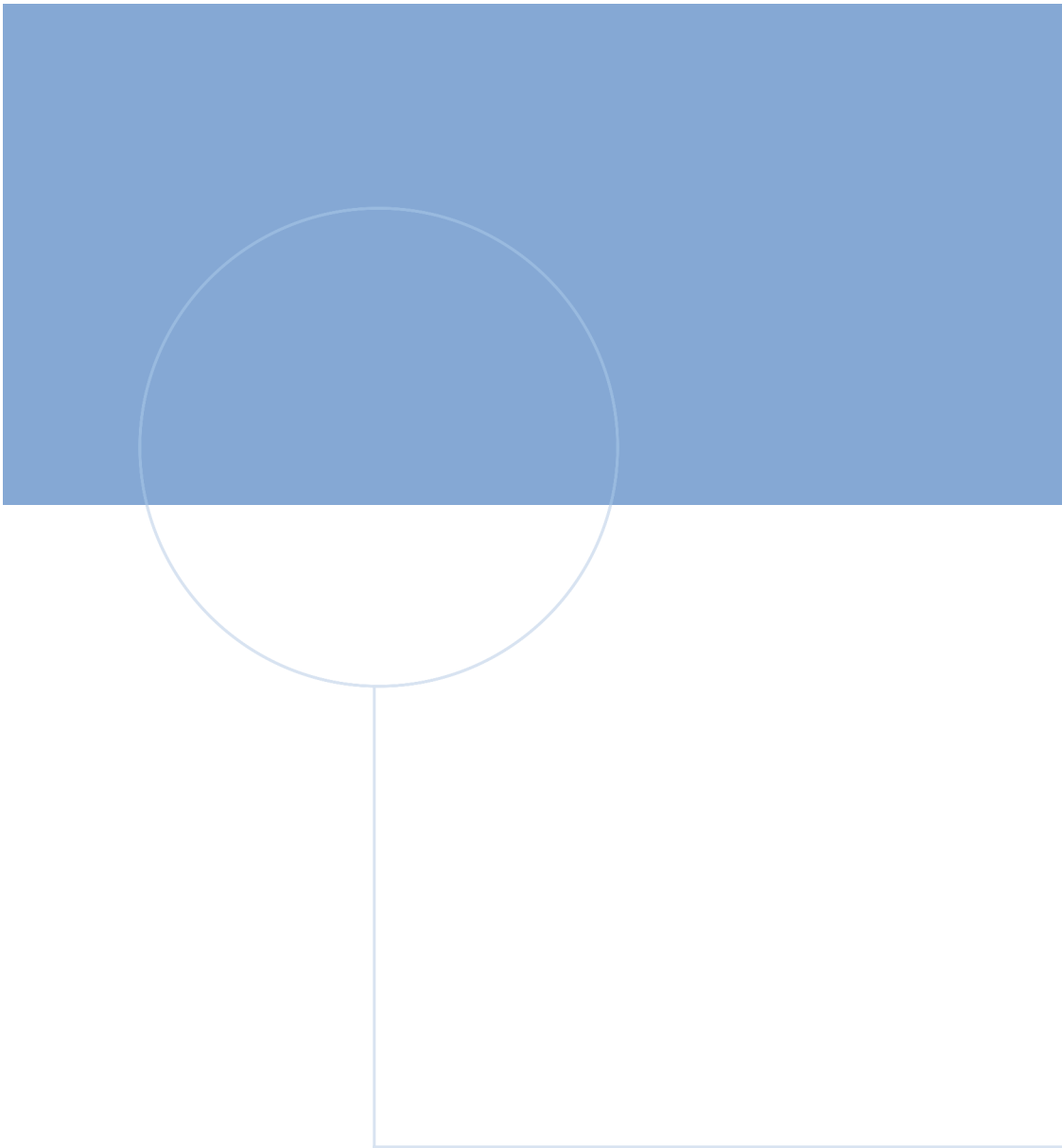
Testing was performed through lint. It identifies coding errors and incorrect coding style.
To manually run this test, execute the command: *npm run lint*
This test is also run automatically, and has to pass, when pushing changes to a git repository, unless one uses the *–no-verify* flag which removes this check.

E Project Manual

The project manual has a seperate submission and can be found there as a seperate pdf (Manual.pdf)

F Code

The code is attached as a zip file (code.zip)



NTNU
Kunnskap for en bedre verden

NORDIC SEMICONDUCTOR