APPLICATION OF BLUETOOTH LOW ENERGY (BLE) FOR OUTDOOR ASSET TRACKING

YEAP TUCK KEONG

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering (Hons) Electronic Engineering

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

January 2024

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	YEAP TUCK KEONG
ID No.	:	19AGB03935
Date	:	06/09/2024

APPROVAL FOR SUBMISSION

I certify that this project report entitled **“APPLICATION OF BLUETOOTH LOW ENERGY FOR OUTDOOR ASSET TRACKING”** was prepared by **YEAP TUCK KEONG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Ir Dr. Teh Peh Chiong
Date	:	7.9.2024

APPLICATION OF BLUETOOTH LOW ENERGY (BLE) FOR OUTDOOR ASSET TRACKING

ABSTRACT

Outdoor asset tracking enables organisations to efficiently monitor and manage valuable assets, which is beneficial in industries such as logistics, construction, and agriculture. The benefits of using Bluetooth Low Energy (BLE) technology for outdoor asset tracking include low cost, compatibility with a wide range of devices, and ease of use. However, the implementation of standard Bluetooth technology has been found to have limitations in outdoor environments due to signal attenuation and interference, as well as being restricted by a limited range, which poses challenges in establishing reliable outdoor asset tracking. The lowest layer of the BLE stack is found to be the Physical Layer (PHY); the solution presented to mitigate or overcome these challenges is the implementation of Coded PHY. Several improvements can be identified using Coded PHY technology, such as increased reliability and range through the use of forward error correction techniques that allow the transmitted signals to tolerate interference and reach greater distances.

This research aims to utilize the STEVAL-IDB012V1 development board equipped with Bluetooth version 5.3 for outdoor asset tracking using Coded PHY in BLE. By comparing Coded PHY with 1M PHY in terms of distance through RSSI readings, the study highlights the advantages of employing Coded PHY. Despite its higher power consumption, Coded PHY is preferred for distance-critical applications like outdoor asset tracking. Additionally, a log distance path loss model can be employed to estimate the distance between the receiving devices such as the smartphone and the evaluation board based on RSSI values, facilitating easy location tracking of belongings.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

GHz	Gigahertz
MHz	Megahertz
Mbps	Megabits per second
Kbps	Kilobits per second
m	Metre
dBm	Decibel milliwatts
dB	Decibel
$RSSI_{d0}$	Reference Received Signal Strength Indication, dBm
RSSI	Received Signal Strength Indication, dBm
d	Distance, m
d0	Reference distance, m
n	Path loss exponent
X_σ	Zero-mean normal random variable with standard deviation
h	Height, m
GPS	Global Positioning System
EM	Electromagnetic
WLAN	Wireless Local Area Network
Wi-Fi	Wireless Fidelity
GSM	Global System for Mobiles
CDMA	Code-Division Multiple Access
2G	Second-Generation Cellular Network
3G	Third-Generation Cellular Network
4G	Forth-Generation Cellular Network
LTE	Long-term Evolution
5G	Fifth-Generation Cellular Network
IoT	Internet of Things

PAN	Personal Area Network
LE	Low Energy
BLE	Bluetooth Low Energy
EDR	Enhanced Data Rate
BR	Basic Rate
ISM	Industrial, Scientific, and Medical
FHSS	Frequency Hopping Spread Spectrum
RFID	Radio Frequency Identification
PHY	Physical Layer
GFSK	Gaussian Frequency Shift Keying
DQPSK	Differential Quadrature Phase Shift Keying
HSP	Headset Profile
HFP	Hands-Free Profile
A2DP	Advanced Audio Distribution Profile
SPP	Serial Port Profile
GATT	Generic Attribute Profile
GAP	Generic Access Profile
L2CAP	Logical Link Control and Adaptation Protocol
ATT	Attribute Protocol
1M PHY	1 Megabit Physical Layer
2M PHY	2 Megabit Physical Layer
CRC	Cyclic Redundancy Check
NR	Signal-to-Noise Ratio
FEC	Forward Error Correction
RF	Radio Frequency
RSSI	Received Signal Strength Indication
SoC	System-on-Chip
AoA	Angle of Arrival
AoD	Angle of Departure
AES	Advanced Encryption Standard
MEMS	Micro-Electro-Mechanical Systems
PC	Personal Computer
3D	Three-dimensional
GCC	CNU Compiler Collection

GDB	GNU Debugger
SWD	Serial Wire Debug
CMSIS	Common Microcontroller Software Interface Standard
DAP	Debug Access Port
IDE	Integrated Development Environment
CPU	Central Processing Unit
SIG	Special Interest Group
DFU	Device Firmware Update
USB	Universal Serial Bus
LED	Ligh-Emitting Diode
MA	Moving Average
Tx	Transmit
EMA	Exponential Moving Average
DMS	Degrees Minutes Seconds
DD	Decimal Degrees
MTU	Maximum Transmission Unit

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Wireless communication differs from wired communication, in which data transmission is conducted by connecting cables or wires physically. Wireless communication refers to the transfer of information or signals between two or more ends that are not connected by electrical conductors (Moozakis, 2023). It uses electromagnetic waves to transfer data across short or large distances. This technology has completely transformed the way we communicate, providing seamless connectivity without the limitations of physical lines. Wireless communication relies on the electromagnetic spectrum, which encompasses frequencies ranging from low-frequency radio waves to high-frequency gamma rays. Various wireless technologies utilize different portions of this spectrum. A variety of wireless communication technologies are available, such as satellite communication, which provides global coverage regardless of the density of the population; it offers telecommunication (Satellite Phones) and positioning and navigation (GPS) in transmitting signals over long distances to achieve global connectivity (Teja, 2024). Furthermore, infrared communication is available to use infrared waves of the EM spectrum for short-range communication, such as remote control, cars, and audio equipment. Bluetooth is a low-range wireless communication system, that allows users to connect various devices and provides signal transmission among the devices. Wireless Local Area Network (WLAN) or Wi-Fi permits users to access the internet wirelessly by connecting devices to an access point such as a Wi-Fi router. Cellular communication is supported by two standards such as the Global System for Mobiles (GSM) and Code-Division Multiple Access (CDMA) in the early days of cellular networking (Moozakis, 2023); it encompasses 2G, 3G, 4G/LTE, and 5G, performs voice and data transmission between devices over long distances via cellular networks.

Some key components needed in a wireless communication system are the transmitter and receiver which can be illustrated as shown in Figure 1.1. A transmitter converts data into electromagnetic signals suitable for wireless transmission, while a receiver records and decodes electromagnetic signals back into the original data (muRata, 2023). An antenna facilitates electromagnetic wave transmission and reception. In addition, the processes of modulation and demodulation are considered in the wireless communication system as shown in Figure 1.2. The modulation involves encoding information onto a carrier signal and allows the transmission of data over long distances, while demodulation involves extracting the original information from the modulated signal in reverse.

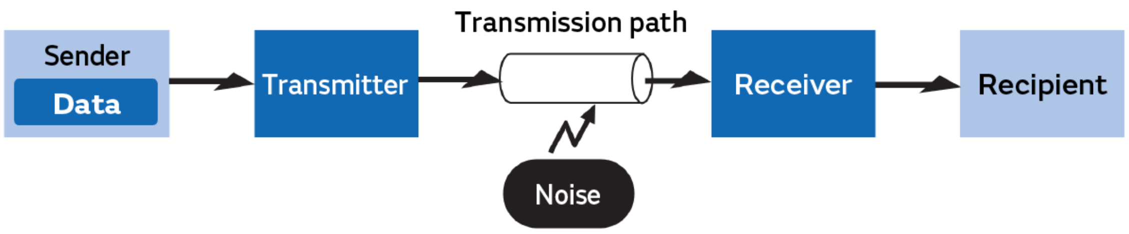


Figure 1.1: Transmitter and receiver in a wireless communication system (muRata, 2023).

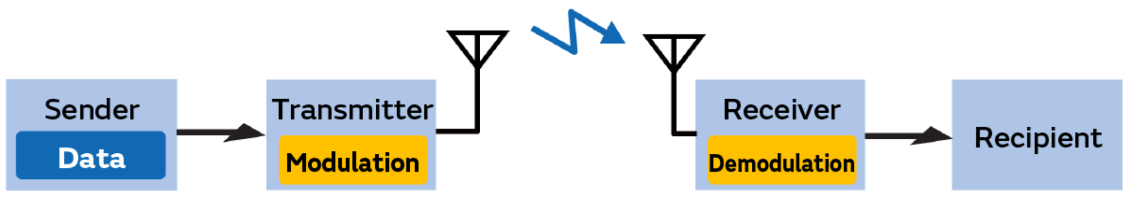
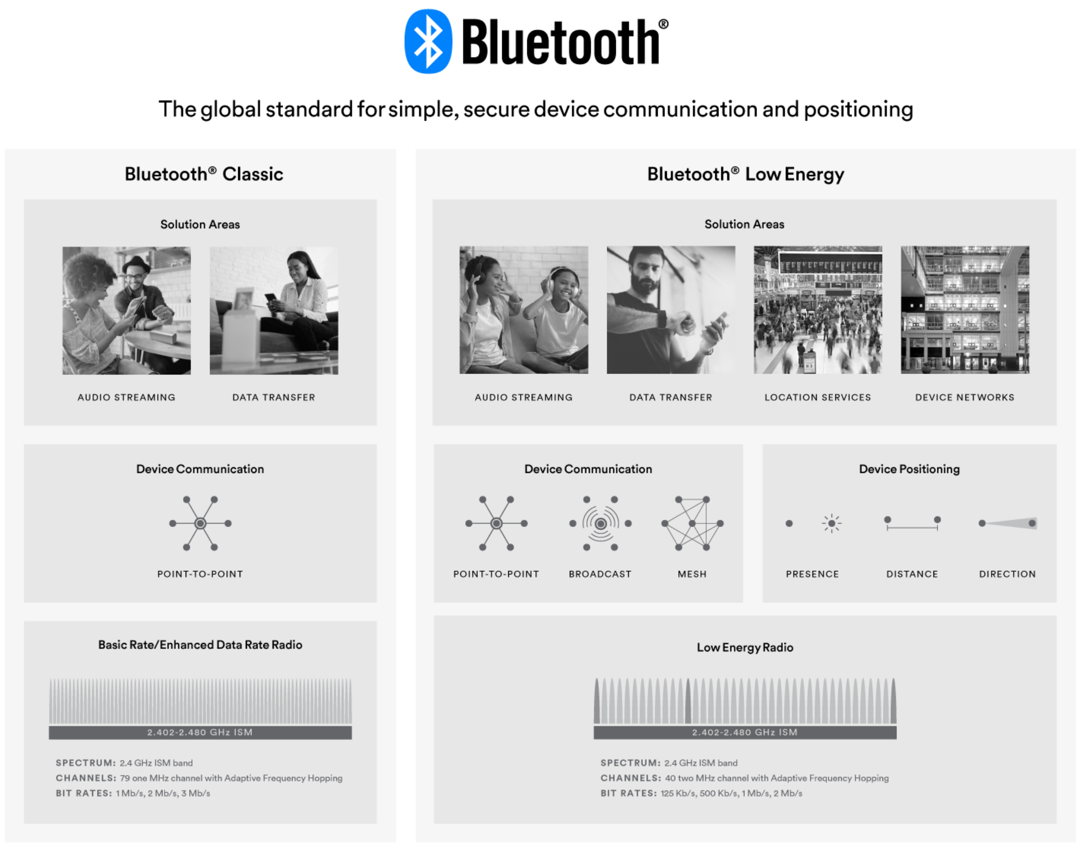


Figure 1.2: Modulation and demodulation applied for wireless communication (muRata, 2023).

Wireless communication is widely used in various fields, including automotive (vehicle-to-vehicle communication), satellite communication (GPS), Internet of Things (IoT) devices, wireless network communication (Wi-Fi and Bluetooth), mobile communication, and healthcare (wireless monitoring devices) (muRata, 2023). However, wireless communication encounters several barriers or challenges, including interference from other wireless devices, signal attenuation over distance, environmental factors such as weather conditions, and security issues such as data interception and illegal access (Agarwal, 2020).

As mentioned earlier, Bluetooth technology is found to be one of the wireless communications, that enables communication between devices without cables or wires. Bluetooth uses a short-range radio frequency and any devices incorporating the technology will be able to communicate with it as it is within the required distance. Its purpose is to facilitate data transfer over short distances, enabling a range of devices such as computers, tablets, smartphones, headphones, and IoT devices to communicate directly without the need wireless router or access point (Intel, n.d.). Bluetooth utilises low-power radio waves to create personal area networks (PANs) and operates within the 2.4 GHz frequency band (GeeksforGeeks, 2024). This makes it ideal for applications such as file sharing, hands-free telephony, wireless audio streaming, and Internet of Things connectivity. Bluetooth Classic and Bluetooth Low Energy (LE) are two Bluetooth standards in use today as hsown in Figure 1.3 (Bluetooth®, n.d.). Enhanced Data Rate (EDR) and Bluetooth Basic Rate (BR) are referred to as Bluetooth Classic and are primarily used to support wireless audio streaming. With its low power consumption, Bluetooth Low Energy is a popular choice for locating devices. Operating in the 2.4 GHz ISM (Industrial, Scientific, and Medical) band and using Frequency Hopping Spread Spectrum (FHSS) for channel utilisation are the similarities between the two standards. The two technologies differ significantly in terms of power consumption; BLE uses less energy than Bluetooth Classic, extending the battery life of devices. In addition, Bluetooth Classic offers higher data rates than BLE which is up to 3 Mbps, while BLE can only offer data rates of around 1 Mbps. Bluetooth technology is a widely used and versatile wireless communication standard due to several key features. Firstly, it operates in the 2.4 GHz ISM frequency band as mentioned previously, making it compatible with a wide range of devices worldwide.

Additionally, its low power consumption is ideal for battery-powered devices, which helps to extend the life of portable electronics. Bluetooth uses short-range radio waves that can typically reach up to 10 metres. However, newer versions of Bluetooth, such as Bluetooth 5.0, significantly increase this range, making it easier to connect over larger areas (Jones, 2020). Another distinguishing feature is its ability to accommodate multiple connections simultaneously, enabling efficient data flow between different devices within a Bluetooth network. Bluetooth technology includes strong security features such as encryption and authentication methods to ensure safe data transfer and prevent unwanted access (Agarwal, 2021).



SPECTRUM: 2.4 GHz ISM band
CHANNELS: 79 one MHz channel with Adaptive Frequency Hopping
BIT RATES: 1 Mb/s, 2 Mb/s, 3 Mb/s

SPECTRUM: 2.4 GHz ISM band
CHANNELS: 40 two MHz channel with Adaptive Frequency Hopping
BIT RATES: 125 Kb/s, 500 Kb/s, 1 Mb/s, 2 Mb/s

Figure 1.3: Comparison between Bluetooth Classic and Bluetooth Low Energy (Bluetooth®, n.d.).

Asset monitoring is a critical procedure for companies and organizations to effectively manage and maximize the use of their physical assets throughout their lifetime. These assets can take many forms, including equipment, vehicles, tools, and more, all of which are essential resources for business operations (Rittenberg & Bottorff, 2022). Asset tracking systems gather real-time data on the location, status, and condition of assets using various technologies such as Bluetooth, RFID (Radio Frequency Identification), GPS (Global Positioning System), and barcodes (Jonker, 2023). This enables enterprises to obtain valuable information on asset utilization, movement patterns, and maintenance requirements. It can lead to improvements in operational efficiency, decreased expenses, improved risk management, and reduced risk of theft or loss (Fatima, 2023). Asset tracking is essential for enhancing security measures, attaining regulatory compliance, and optimizing resource allocation and inventory management. In simple terms, asset monitoring is crucial for the efficient administration and care of tangible assets, enabling businesses to maximize their value and streamline their processes.

Bluetooth technology offers several advantages for tracking assets in indoor or short-range outdoor locations. Bluetooth Low Energy (BLE) technology enables long-term asset tracking without the need for regular battery replacement, thus extending the battery life of tracking devices. Additionally, Bluetooth-enabled tracking devices are a cost-effective alternative for tracking assets in restricted spaces, as they are typically less expensive than other asset tracking solutions such as RFID and GPS-based systems (CAMPORESOFT, 2019). Indoor asset tracking is well-suited to Bluetooth technology because GPS signals may be unreliable or unavailable in such environments (Sokolova, 2023). By using Bluetooth beacons or tags, enterprises can accurately track the location of their assets in real-time. Bluetooth is particularly useful in settings such as warehouses, healthcare, or manufacturing facilities where precise asset location within a specific area is necessary due to its excellent accuracy in short-range tracking (Link Labs, 2021). Bluetooth technology also provides built-in security features such as authentication and encryption, ensuring that asset-tracking data is kept safe and shielded from unwanted access (ToolSense, 2023).

1.2 Problem Statements

Asset tracking enables companies and organisations in a wide range of industries to successfully manage their assets. From inventory management and loss prevention to resource optimisation and regulatory compliance, asset tracking offers many benefits that are critical to ensuring operational efficiency, reducing risk, and driving business success. Organisations can access data instantly and reduce the risk of asset theft or loss through effective asset tracking. It also helps to analyse the condition of assets, eliminate costly ghost assets, and create efficient maintenance plans.

BLE, GPS, and RFID are the common technologies used to track assets. RFID systems use radio frequency signals to locate and monitor tagged assets. RFID is often used in environments such as manufacturing plants, warehouses, and retail outlets to track high volume of assets. Satellites are used by GPS to provide precise location data for tracked assets outdoors. When high-value assets, equipment, and vehicles need to be monitored over a wide area, GPS-based asset tracking systems are often used. Low-power, short-range communication is the focus of BLE. Hospitals, airports, and industrial facilities are among the environments where BLE-based asset-tracking technologies are commonly used.

Bluetooth technology is generally more cost-effective than GPS and RFID, especially for indoor and close-range outdoor applications. In general, the costs of Bluetooth solutions are lower in terms of infrastructure and hardware than GPS and RFID. BLE is also designed to use minimal power, making it suitable for battery-powered devices that need to operate for long periods of time without battery replacement or recharging. This makes Bluetooth a practical option for asset-tracking applications where battery life is critical. Smartphones, tablets, and other mobile devices support Bluetooth technology, making it easy for users to interact with and manage tracked assets using their current devices. As a result, Bluetooth-based asset tracking systems are easier to use and more accessible. Hence, BLE was discussed and implemented for outdoor asset tracking applications in this project.

1.3 Aims and Objectives

The objectives of this project are shown as follows:

i) To understand the BLE (Bluetooth 5) technology for asset tracking.

ii) To understand the important parameters for BLE asset tracking.

iii) To demonstrate the application of BLE for outdoor asset tracking.

1.4 Scope

This project focuses on implementing outdoor asset tracking using Bluetooth Low Energy (BLE). Previous discussions have highlighted the suitability of Bluetooth technology for short to medium-range tracking, making it ideal for monitoring assets in confined spaces such as buildings, campuses, or industrial facilities. However, long-range monitoring for outdoor asset tracking typically requires the use of technologies such as GPS.

To overcome the limitations of Bluetooth technology in distance detection compared to other technologies, this project will explore and consider the use of a feature offered in BLE known as Coded PHY. Coded PHY has the potential to extend the effective range of asset tracking, thereby making the project's goal of outdoor asset tracking more practical and feasible. By leveraging Coded PHY, we aim to enhance the range and reliability of Bluetooth-based asset tracking, enabling effective monitoring of assets even in outdoor environments with greater distances between the tracker and the asset.

1.5 Outline

This report consists of five chapters, including Introduction, Literature Review, Methodology, Results and Discussions, and Conclusions and Recommendations. The background, problem statement, and objectives of this project are mentioned in Chapter 1 (Introduction).

Furthermore, Chapter 2 (Literature Review) reviews the contextual framework of the study concerning Bluetooth technology, including both Bluetooth Classic and BLE. A comparative analysis between Bluetooth Classic and BLE will be conducted to discuss their distinctive characteristics. Additionally, parameters related to range in Bluetooth devices will be examined to understand the fundamental elements for extending range. Studies concerning BLE will be reviewed to enhance the understanding of BLE concepts and align them with the objective of this project. Moreover, the hardware and software used in this project will be introduced as well.

Additionally, Chapter 3 (Methodology) mentions the approaches to be used to achieve the objectives of this project. The detailed execution steps in conducting the experiments will be included. The key parameters and data analysis methods identified in the research reviewed from the previous chapter will be studied and implemented in this project as well.

Chapter 4 (Results and Discussions) will focus on the presentation of the results and the discussion around them. This will involve documenting the results of the project and thoroughly explaining the results collected. The comparison between the physical layers which are Coded PHY and 1M PHY is examined. Additionally, the accuracy of the distance estimation using the path loss model is discussed. Furthermore, the direction prediction using techniques such as trilateration and sensor integration is performed. By examining these findings in detail, the aim is to ensure and verify the accuracy and reliability, thereby strengthening the credibility of the project's conclusions.

In Chapter 5 (Conclusions and Recommendations), the focus shifts to summarising the project's findings and validating the original objectives. Recommendations for future action are also provided and discussed in detail. These recommendations are intended to improve the conduct of similar projects in the future, to achieve more accurate and insightful results.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter examines the existing literature to gain insight into the research topic, highlighting its significance and relevance within the field of study. The literature review plays a crucial role in providing the basic concepts necessary to carry out this project.

The background study reviews Bluetooth technology, including discussions of its two standards: Bluetooth Classic and Bluetooth Low Energy. It also identifies key range parameters when using Bluetooth technology for outdoor asset tracking. It also summarises relevant studies, including literature reviews and theoretical analyses. The studies contribute to a better understanding of BLE-related concepts, and the experiments conducted in these studies validate the theories discussed.

2.2 Background of study – Bluetooth

A study is carried out on Bluetooth technology, looking at two standards: Bluetooth Classic and BLE, focusing on their respective characteristics. Furthermore, the physical layers within BLE are analysed in terms of data transfer rates and distance capabilities. A comparison is also made between Bluetooth Classic and BLE, taking into account their unique characteristics. Additionally, the factors influencing the devices connected via Bluetooth technology are examined.

2.2.1 Introduction to Bluetooth Classic

Bluetooth Classic refers to one of the Bluetooth standards in wireless communication technology that operates in the physical layer of the Bluetooth protocol stack. Bluetooth Classic connects devices within a short range using radio signals in the 2.4 GHz ISM band. The transmission power levels, frequency hopping patterns, and modulation strategies that enable reliable communication are all included in the physical layer. The modulation techniques available for Bluetooth Classic's two modes (BR and EDR) are 1 Mbps GFSK for BR, up to 2 Mbps for π/4-DQPSK and up to 3 Mbps for 8DPSK in the case of EDR as shown in Figure 2.1 (electronicsnotes, n.d.). Bluetooth BR/EDR can transmit high-definition video and audio at higher data rates due to its higher data rates. High data rates are required for high-quality audio, which BLE typically cannot deliver (Argenox, 2020).

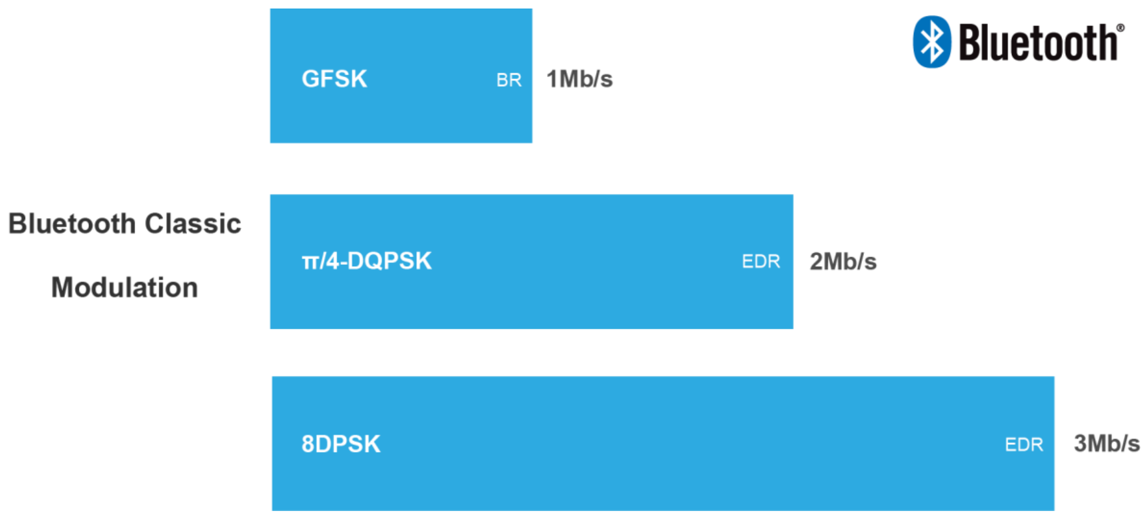
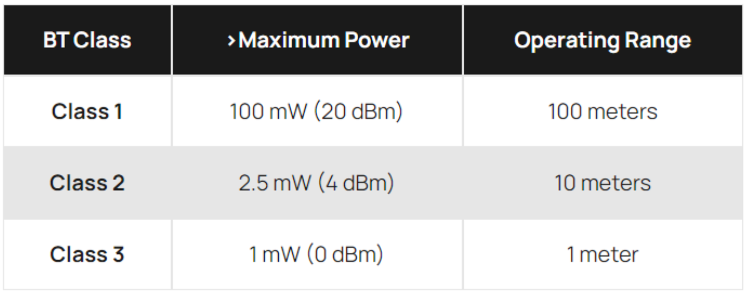


Figure 2.1: Modulation techniques in Bluetooth Classic (Argenox, 2020).

Bluetooth Classic devices are categorised into three groups according to the transmission power of the devices such as Class 1, Class 2, and Class 3 as shown in Figure 2.2 (Ezurio, n.d.). With a range of up to 100 metres, Class 1 devices have the highest transmitting power, up to about +20 dBm. These devices are typically used in long-range communication applications. Class 2 devices, which have a range of up to 10 metres and can transmit up to +4 dBm. With a range of up to one metre and the lowest output power (+0 dBm), Class 3 devices are often used in applications that require close-range communication.



BT Class	›Maximum Power	Operating Range
Class 1	100 mW (20 dBm)	100 meters
Class 2	2.5 mW (4 dBm)	10 meters
Class 3	1 mW (0 dBm)	1 meter

Figure 2.2: Three classes in Bluetooth Classic (Ezurio, n.d.).

Channels and bandwidth allocation are concepts defined by the Bluetooth Classic physical layer. Using frequency hopping spread spectrum technology, Bluetooth Classic transmits data over 79 channels in the 2.4 GHz ISM band. By hopping in different directions, this pattern reduces interference from other wireless technologies using the same frequency band. A total of 79 MHz of bandwidth is available for Bluetooth Classic transmission, with a bandwidth of 1 MHz for each channel from 2402 MHz to 2480 MHz as shown in Figure 2.3 (Argenox, 2020).

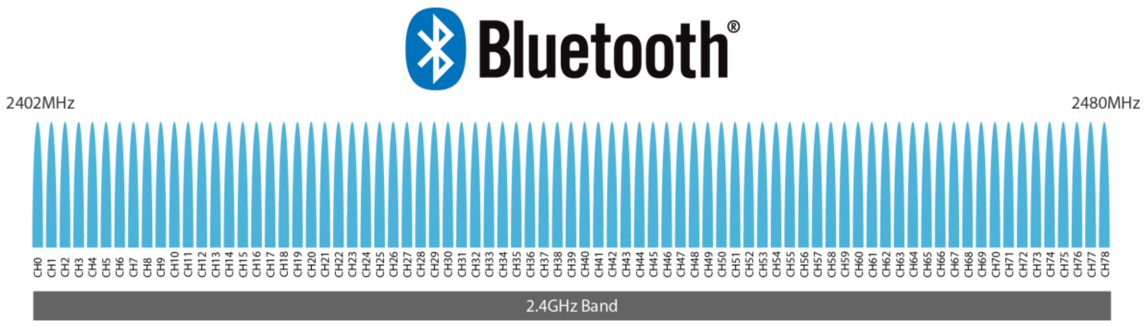

Bluetooth®
2402MHz
2480MHz
CH0 CH1 CH2 CH3 CH4 CH5 CH6 CH7 CH8 CH9 CH10 CH11 CH12 CH13 CH14 CH15 CH16 CH17 CH18 CH19 CH20 CH21 CH22 CH23 CH24 CH25 CH26 CH27 CH28 CH29 CH30 CH31 CH32 CH33 CH34 CH35 CH36 CH37 CH38 CH39 CH40 CH41 CH42 CH43 CH44 CH45 CH46 CH47 CH48 CH49 CH50 CH51 CH52 CH53 CH54 CH55 CH56 CH57 CH58 CH59 CH60 CH61 CH62 CH63 CH64 CH65 CH66 CH67 CH68 CH69 CH70 CH71 CH72 CH73 CH74 CH75 CH76 CH77 CH78
2.4GHz Band

Figure 2.3: Bandwidth divided into 79 channels in Bluetooth Classic (Agarwal, 2021).

The features and functionality of Bluetooth Classic devices are largely determined by their profiles. The way Bluetooth devices connect and communicate with each other to perform specific functions or services is defined by a profile. For instance, the Headset Profile (HSP) for hands-free calling, the Hands-Free Profile (HFP) for hands-free audio, the Advanced Audio Distribution Profile (A2DP) for high-quality audio streaming, and the Serial Port Profile (SPP) for creating virtual serial connections between devices are examples of common Bluetooth Classic profiles (JIMBLOM, n.d.). Each profile describes in detail the protocols, data formats and communication techniques that are required to achieve its intended functionality.

2.2.2 Introduction to Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a low-power, short-range wireless communication protocol for devices (BasuMallick, 2022). BLE was incorporated into the Bluetooth 4.0 protocol in 2010 to address the growing need for low-power wireless connectivity in many applications.

The physical radio data rate, or the speed at which the radio transmits data, defines the data throughput of BLE. For Bluetooth versions below 5.0, the rate is fixed at 1Mbps, but for Bluetooth versions 5.0 and above, the rate varies depending on the mode and PHY. The rate can be 1Mbps which is similar to previous iterations, or 2Mbps when using the high-speed feature. The rate drops to 500 or 125 Kbps if the long-range option is applied (AFANEH, 2022).

BLE uses 40 channels and frequency hopping spread spectrum to operate in the 2.4 GHz ISM band in terms of channels and bandwidth as shown in Figure 2.4 (MathWorks, n.d.). With 2 MHz bandwidth per channel, a total of 80 MHz bandwidth is available for BLE communications. Its architecture not only reduces interference from other wireless technologies but also makes communication durable and resilient.

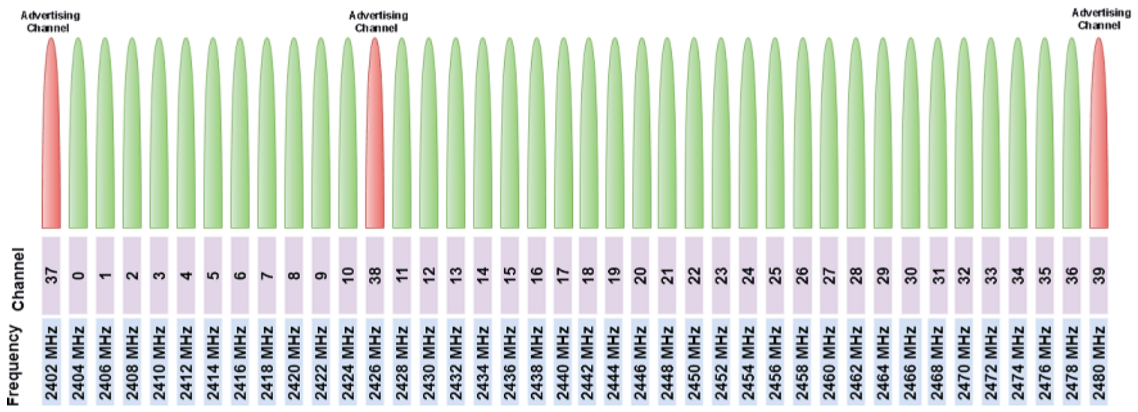

Advertising Channel
Advertising Channel
Advertising Channel
Channel
37 0 1 2 3 4 5 6 7 8 9 10 38 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 39
Frequency
2402 MHz 2404 MHz 2406 MHz 2408 MHz 2410 MHz 2412 MHz 2414 MHz 2416 MHz 2418 MHz 2420 MHz 2422 MHz 2424 MHz 2426 MHz 2428 MHz 2430 MHz 2432 MHz 2434 MHz 2436 MHz 2438 MHz 2440 MHz 2442 MHz 2444 MHz 2446 MHz 2448 MHz 2450 MHz 2452 MHz 2454 MHz 2456 MHz 2458 MHz 2460 MHz 2462 MHz 2464 MHz 2466 MHz 2468 MHz 2470 MHz 2472 MHz 2474 MHz 2476 MHz 2478 MHz 2480 MHz

Figure 2.4: Bandwidth divided into 40 channels in BLE (MathWorks, n.d.)

The Bluetooth LE protocol stack is also known as Bluetooth Low Energy architecture. The application layer, host layer and controller layer are the three key components that make up the BLE protocol stack architecture as shown in Figure 2.5 (Woolley, 2017). The hardware component of the Bluetooth Low Energy architecture is the controller. The physical layer, which is the lowest layer, is defined as the actual radio used for modulation and demodulation of the data and communication. The Link Layer sits on top of this and is responsible for packet transport, device discovery and connection establishment. Generic Attribute Profile (GATT), Generic Access Profile (GAP), Logical Link Control and Adaptation Protocol (L2CAP) and Attribute Protocol (ATT) are components of the host layer (BasuMallick, 2022). The L2CAP acts as a protocol multiplexer, combining multiple protocols into a single BLE packet, while the ATT and GATT govern data exchange between BLE devices. Key components of device behaviour, such as connection establishment, device discovery and security management, are managed by the GAP. The part of the Bluetooth Low Energy stack architecture that communicates directly with users is called the application layer. It includes the general application architecture, application logic and user interface.

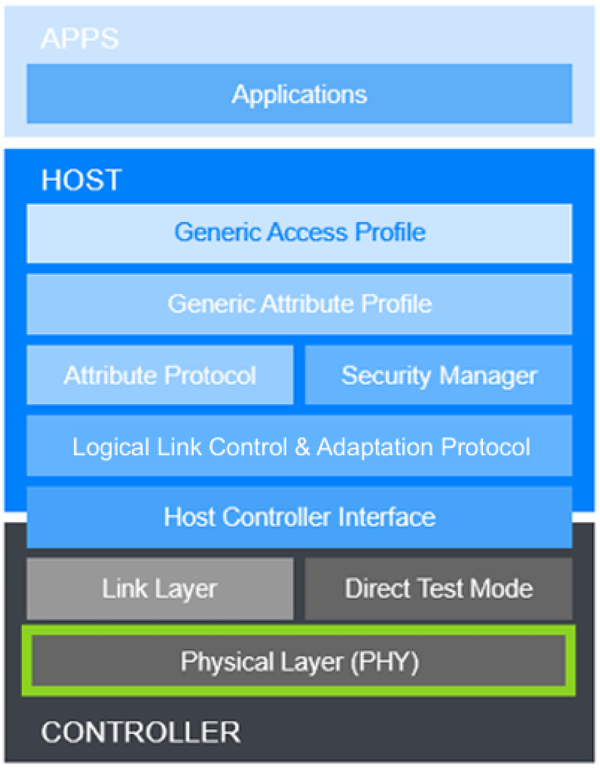


Figure 2.5: BLE stack (Woolley, 2017).

2.2.2.1 Introduction to 1 Mbps PHY in BLE

LE 1M PHY, the Physical Layer (PHY) utilized in Bluetooth 4, employs Gaussian Frequency Shift Keying modulation with a symbol rate of 1 mega symbol per second (Ms/s) (Woolley, 2017). In practical terms, this symbol rate directly translates to a bit rate of 1 Mb/s, as each symbol corresponds to one data bit. Notably, LE 1M remains a mandatory and available option for use in Bluetooth 5, ensuring compatibility and continuity across BLE implementations.

2.2.2.2 Introduction to 2 Mbps PHY in BLE

The introduction of the new LE 2M PHY in Bluetooth 5 enables higher data rates, operating at 2 mega symbols per second (Ms/s), surpassing the capabilities of the LE

1M PHY used in Bluetooth 4. Both the LE 2M and LE 1M PHYs are classified as Uncoded PHYs as they use a 1-symbol representation per data bit.

It is important to note that the feature of doubling the data speed of Bluetooth 5 may require updates to hardware or software stacks in older devices, as not all chipsets may support this enhancement. In addition, to achieve the higher throughput of the LE 2M PHY, both communicating Bluetooth devices must support this new PHY (Afaneh, 2023). However, there are limitations to using the LE 2M PHY, such as the inability to transmit primary advertisements on the primary channels, unlike the mandatory LE 1M PHY. Therefore, not all chipsets claiming Bluetooth 5 compatibility may support the higher data rates of the LE 2M PHY. Advertising and discovery can still take place on the LE 1M PHY, with connections established on the secondary advertising channel using the LE 2M PHY.

2.2.2.3 Introduction to Coded PHY in BLE

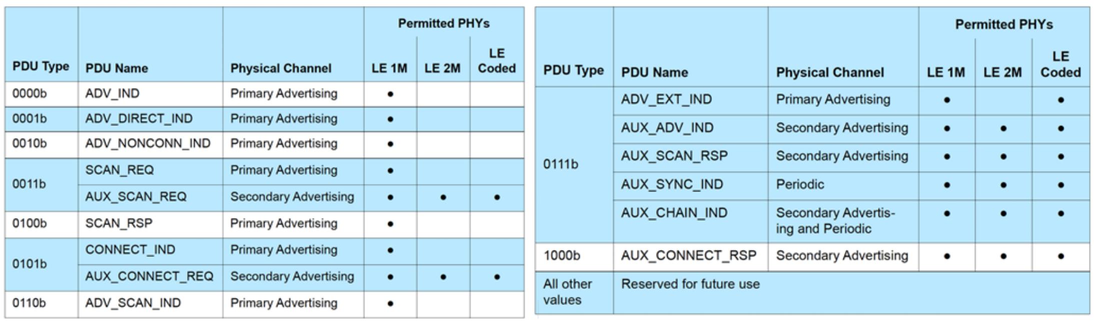
The Coded PHY, introduced in Bluetooth Low Energy (BLE) with Bluetooth 5, aims to improve communication range and robustness in complex wireless environments, extending the applicability of BLE. Unlike traditional uncoded PHYs such as LE 1M and LE 2M, which use a one-to-one bit-to-symbol mapping, the Coded PHY uses a 2-symbol (S=2) or 8-symbol (S=8) representation per bit.

While Bluetooth 4 BLE only detects errors using a method known as Cyclic Redundancy Check (CRC), Bluetooth 5 introduces error correction, allowing data to be accurately decoded even at a lower Signal-to-Noise Ratio (SNR), thereby extending the transmission range. Using Forward Error Correction (FEC), the Coded PHY encodes data redundantly, enabling receivers to reconstruct lost or corrupted data, increasing communication reliability.

The choice between S=2 and S=8 encoding schemes affects range and data rate. S=2 doubles the range, while S=8 quadruples it. However, this increases the number of symbols transmitted and reduces the overall data rate (Woolley, 2017). At lower data rates (125 Kbps to 2 Mbps), the coded PHY prioritises range over throughput. This trade-off improves signal penetration and resilience to obstructions and interference. Particularly beneficial for applications requiring extended range, such as outdoor asset tracking and industrial monitoring, the Coded PHY ensures reliable BLE communication in challenging RF environments.

2.2.2.3.1 Implementation of Coded PHY

There are two approaches to using Coded PHY in Bluetooth Low Energy (BLE) communication: advertising-only state and connection state (Afaneh, 2023). In advertising-only mode, Coded PHY is used to advertise, allowing other devices that support Coded PHY to discover the advertisements. When a connection is established between two BLE devices, one device advertises using Coded PHY while the other device searches or scans for these advertisements on the same PHY. Extended Advertisement is applied in this scenario because it allows Coded PHY to be the primary channel, unlike other types of advertising as shown in Figure 2.6. Choosing the suitable advertisement type is essential in enabling BLE applications under Coded PHY.



			Permitted PHYs		
PDU Type	PDU Name	Physical Channel	LE 1M	LE 2M	LE Coded
0000b	ADV_IND	Primary Advertising	•		
0001b	ADV_DIRECT_IND	Primary Advertising	•		
0010b	ADV_NONCONN_IND	Primary Advertising	•		
0011b	SCAN_REQ	Primary Advertising	•		
	AUX_SCAN_REQ	Secondary Advertising	•	•	•
0100b	SCAN_RSP	Primary Advertising	•		
0101b	CONNECT_IND	Primary Advertising	•		
	AUX_CONNECT_REQ	Secondary Advertising	•	•	•
0110b	ADV_SCAN_IND	Primary Advertising	•		

			Permitted PHYs		
PDU Type	PDU Name	Physical Channel	LE 1M	LE 2M	LE Coded
0111b	ADV_EXT_IND	Primary Advertising	•		•
	AUX_ADV_IND	Secondary Advertising	•	•	•
	AUX_SCAN_RSP	Secondary Advertising	•	•	•
	AUX_SYNC_IND	Periodic	•	•	•
	AUX_CHAIN_IND	Secondary Advertis-ing and Periodic	•	•	•
1000b	AUX_CONNECT_RSP	Secondary Advertising	•	•	•
All other values	Reserved for future use				

Figure 2.6: Advertising types available (Afaneh, 2023).

In addition, the PHY Update Procedure facilitates the transition to Coded PHY, allowing it to be switched from the previous connections such as 1M PHY. This PHY Update Procedure can be conducted by either the central or peripheral device; the PHY

Update Procedure initiated by the central can be illustrated as shown in Figure 2.7 while the operation in Figure 2.8 shows the peripheral starts the update procedure.

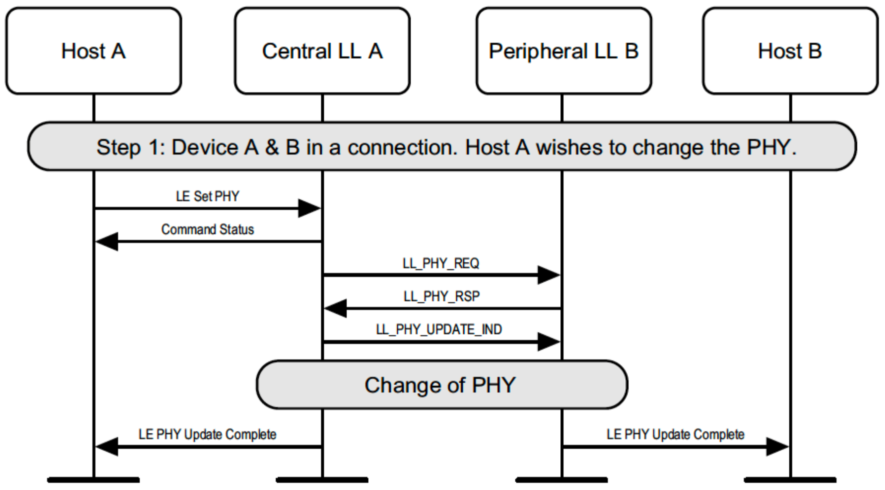
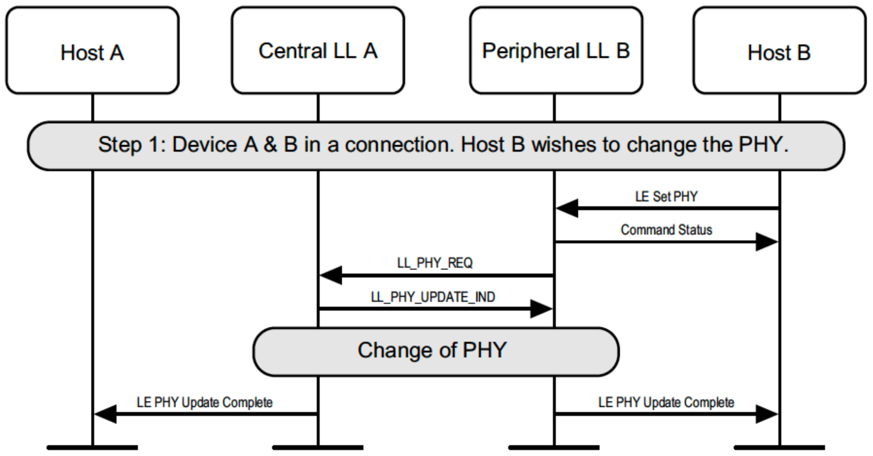


Figure 2.7: Operation of central initiates PHY Update Procedure (Afaneh, 2023).

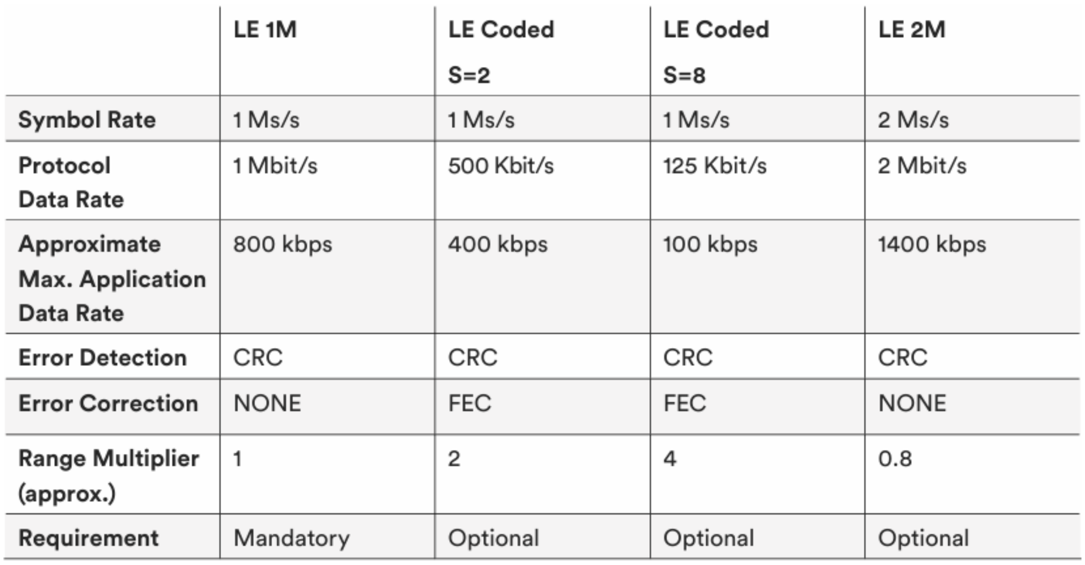


Type text her

Figure 2.8: Operation of peripheral initiates PHY Update Procedure (Afaneh, 2023).

2.2.2.4 Comparison among physical layers in BLE

The differences between the various physical layers in BLE are outlined based on their data rate and range characteristics as shown in Figure 2.9. Each PHY offers unique trade-offs in terms of data rate, symbol representation, range, and application suitability. While the LE 1M and LE 2M PHYs are suitable for typical applications with moderate data throughput requirements, the LE Coded PHY offers extended range capabilities at a reduced data rate, making it particularly suitable for scenarios where communication range is a priority.



	LE 1M	LE Coded S=2	LE Coded S=8	LE 2M
Symbol Rate	1 Ms/s	1 Ms/s	1 Ms/s	2 Ms/s
Protocol Data Rate	1 Mbit/s	500 Kbit/s	125 Kbit/s	2 Mbit/s
Approximate Max. Application Data Rate	800 kbps	400 kbps	100 kbps	1400 kbps
Error Detection	CRC	CRC	CRC	CRC
Error Correction	NONE	FEC	FEC	NONE
Range Multiplier (approx.)	1	2	4	0.8
Requirement	Mandatory	Optional	Optional	Optional

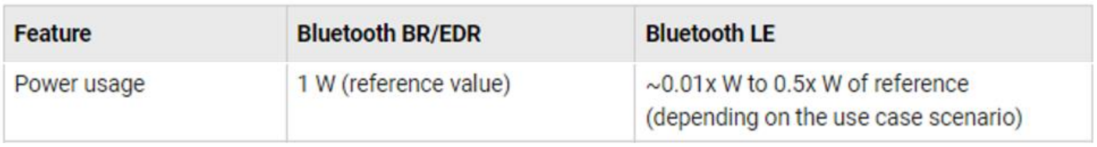
Figure 2.9: Comparison between LE physical layers (Woolley, 2017).

2.2.3 Comparisons between Bluetooth Classic and BLE

The study examines and contrasts Bluetooth Classic and BLE across several key metrics, including power consumption, range, data throughput, communication mode, and application suitability. From the comparison results, a summary is provided, and scenarios or operations are identified where each technology is best suited.

2.2.3.1 Power consumption

As mentioned before, Bluetooth Low Energy (BLE) is designed to operate at significantly reduced power levels compared to traditional Bluetooth, resulting in lower power consumption and longer battery life. BLE achieves this by effectively managing its power consumption, particularly through its ability to enter a sleep mode when not actively transmitting data. On the other hand, Bluetooth Classic lacks this ability and remains either fully powered on or fully powered off, with no in-between states (PROCTOR, 2023). Furthermore, BLE transmits data in smaller increments even during active transmission, further reducing power consumption compared to Bluetooth Classic, which typically requires more power for similar data transfers which can be illustrated as shown in Figure 2.10. Hence, BLE is a compelling choice for battery-powered devices and applications that require low-power wireless communication because of its optimised power management and efficient data transfer, which helps to extend battery life.



Feature	Bluetooth BR/EDR	Bluetooth LE
Power usage	1 W (reference value)	~0.01x W to 0.5x W of reference (depending on the use case scenario)

Figure 2.10: Comparison between power consumption in both Bluetooth Classic and BLE (MathWorks, n.d.).

2.2.3.2 Range

Bluetooth Classic and BLE differ in the range they can communicate with. Bluetooth Classic typically has a greater range than BLE, making it suitable for applications that require wider coverage (How To Electronics, 2023). Conversely, BLE's shorter range makes it more suitable for applications that require more localised coverage, such as indoor navigation or location-based services.

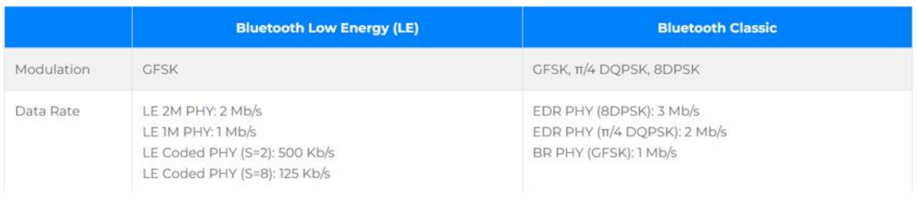
For Bluetooth Classic devices, Class 1 devices can achieve ranges of up to 100 metres, Class 2 devices up to 10 metres, and Class 3 devices up to 1 metre in optimal conditions. In contrast, BLE devices typically have a range of 10-100 metres in standard mode, which is influenced by factors such as transmission power, environmental conditions, and antenna design.

Bluetooth Classic generally has a longer range than BLE. However, the long-range mode of BLE, also known as Coded PHY introduced with Bluetooth 5.0, significantly extends its range. This enhancement allows BLE devices to achieve distances of up to several hundred metres in optimal conditions.

2.2.3.3 Data Throughput

Both Bluetooth Classic and BLE utilize Gaussian Frequency Shift Keying (GFSK) modulation for data transmission. However, Bluetooth Classic has additional modulation techniques such as π/4-Differential Quadrature Phase Shift Keying (π/4-DQPSK) and 8-Differential Phase Shift Keying (8DPSK). As mentioned previously, the modulation techniques available in Bluetooth Classic depend on the mode: Basic Rate (BR) mode supports 1 Mbps GFSK, while Enhanced Data Rate (EDR) mode supports up to 2 Mbps for π/4-DQPSK and up to 3 Mbps for 8DPSK. On the other hand, BLE supports multiple data rates using different physical layers (PHY). It offers 1 Mbps in LE 1M PHY and 2 Mbps in LE 2M PHY. Additionally, BLE supports lower data rates of 125 Kbps and 500 Kbps using LE Coded PHY. These lower data rates are achieved by applying coding schemes to optimise power consumption for various applications.

The difference between Bluetooth Classic and BLE in modulation techniques and data rate can be illustrated as shown in Figure 2.11. Hence, Bluetooth Classic's data rates depend on the mode (BR or EDR), while BLE supports multiple data rates through different PHYs, including lower data rates achieved through coding schemes for power optimisation.

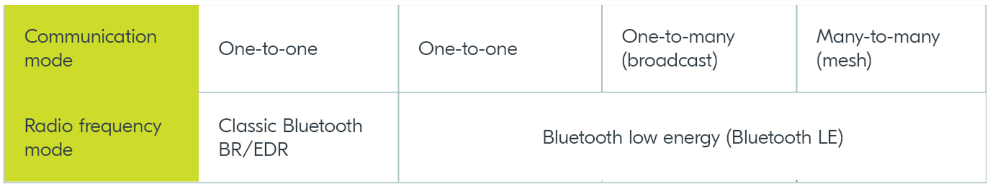


	Bluetooth Low Energy (LE)	Bluetooth Classic
Modulation	GFSK	GFSK, π/4 DQPSK, 8DPSK
Data Rate	LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE Coded PHY (S=2): 500 Kb/s LE Coded PHY (S=8): 125 Kb/s	EDR PHY (8DPSK): 3 Mb/s EDR PHY (π/4 DQPSK): 2 Mb/s BR PHY (GFSK): 1 Mb/s

Figure 2.11: Comparison between modulation and data rate in both Bluetooth Classic and BLE (Bluetooth®, n.d.).

2.2.3.4 Communication mode

Bluetooth Classic operates in a point-to-point communication mode, enabling direct connections between two devices for tasks such as streaming audio, file transfer, and hands-free calling. On the other hand, Bluetooth Low Energy (BLE) supports multiple communication topologies, including point-to-point, broadcast, and mesh networking. Like Bluetooth Classic, BLE enables point-to-point communication for tasks such as connecting smartphones to wearable fitness trackers or smart home devices (NORDIC, 2021). In addition, BLE supports broadcast communication, which allows a single device to send data to multiple recipients simultaneously, commonly used for proximity sensing and location-based services. Notably, a key advantage of BLE is its native support for mesh networking, which enables decentralised communication between multiple devices, ideal for applications such as smart lighting, home automation, and large-scale sensor networks. In essence, the difference in communication mode between Bluetooth Classic and BLE is summarized as shown in Figure 2.12, in which both Bluetooth Classic and BLE offer point-to-point communication, BLE's additional support for mesh networking enhances its suitability for distributed and scalable applications.

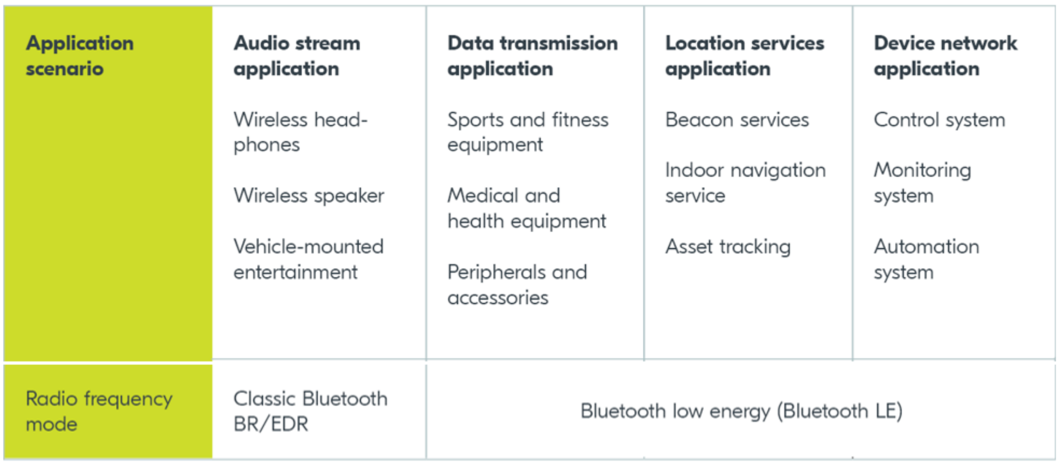


Communication mode	One-to-one	One-to-one	One-to-many (broadcast)	Many-to-many (mesh)
Radio frequency mode	Classic Bluetooth BR/EDR	Bluetooth low energy (Bluetooth LE)		

Figure 2.12: Comparison in communication mode between Bluetooth Classic and BLE (NORDIC, 2021).

2.2.3.5 Application

Bluetooth Classic is typically used in scenarios that require high data throughput, and continuous communication, such as audio streaming, file transfer, and hands-free calling in a variety of devices including automotive systems, headphones, speakers, and smartphones. In contrast, Bluetooth Low Energy (BLE) is well suited to applications where low power consumption, intermittent communication, and extended battery life are important. Applications related to data transfer, location services, and device networking are well suited to BLE. It is widely used in IoT devices, wearables, health and fitness trackers, smart home devices, proximity sensors, and location-based services. BLE's energy-efficient communication mode makes it particularly suitable for devices that rely on battery power for extended periods of time. The application scenarios for both Bluetooth Classic and BLE can be illustrated as shown in Figure 2.13.



Application scenario	**Audio stream application**	**Data transmission application**	**Location services application**	**Device network application**
	Wireless head-phones	Sports and fitness equipment	Beacon services	Control system
	Wireless speaker	Medical and health equipment	Indoor navigation service	Monitoring system
	Vehicle-mounted entertainment	Peripherals and accessories	Asset tracking	Automation system
Radio frequency mode	Classic Bluetooth BR/EDR	Bluetooth low energy (Bluetooth LE)		

Figure 2.13: Applications that are suitable for Bluetooth Classic and BLE (NORDIC, 2021).

2.2.3.6 Summary

Both Bluetooth Low Energy (BLE) and Bluetooth Classic are Bluetooth-based wireless communication technologies, but they have different functions and features. Where power sources are readily available and high data rates are required, Bluetooth Classic will be the better option for this case. On the other hand, BLE will be best suited for low-power, low-data-rate applications, especially Internet of Things devices that rely on small batteries.

2.2.4 Measurement of signal strength

RSSI, or Received Signal Strength Indication, is a key parameter in wireless communication systems such as Bluetooth that measures the strength of the signal a device receives from a transmitter. Usually expressed in decibels (dB), and RSSI does not represent the speed of a Bluetooth connection, but rather the strength of the signal (Li, 2022). The relationship between the signal strength and RSSI value is summarized as shown in Table 2.1. In fact, RSSI values can vary significantly between chipset manufacturers such as the same RSSI reading on two different smartphones with different chipsets may indicate different signal strengths (Gao, 2015).

Table 2.1: Relationship between RSSI and signal strength (Li, 2022).

RSSI	Signal Strength
Below -50 dBm	Good
-70 dBm to -80 dBm	Fair
-100 dBm	No Signal

For Bluetooth, RSSI is an important indicator of signal strength and connection quality. Higher RSSI values generally indicate stronger signals and better connection quality. In addition, RSSI helps to estimate the proximity between Bluetooth devices. By tracking changes in RSSI over time, applications can measure the distance between devices, facilitating features such as location tracking and asset management. Moreover, monitoring RSSI values can detect potential problems with the stability of the connection. Abrupt fluctuations or drops in RSSI can indicate interference, obstructions, or environmental factors affecting the wireless link.

2.2.5 Factors affect the range of Bluetooth devices

The range of connected devices is influenced by a number of factors, including the radio spectrum used by Bluetooth technology, which is advantageous for wireless communication. The physical layer (PHY) defines essential aspects of radio usage such as data rate, error detection and correction methods, interference protection, and signal clarity techniques over different ranges (Franklin & Pollette, n.d.). Furthermore, receiver sensitivity is critical since it measures the minimum signal strength for correct data decoding. In addition, transmit power plays an important role in this case, higher transmit signal strength correlates with longer achievable ranges. However, the cost of increased battery consumption needs to be considered (Li, 2022). Moreover, antenna design and placement have a significant impact on range and signal strength, with antenna gain playing a critical role in converting electrical signals to radio waves and vice versa. Pathloss, which is influenced by factors such as distance, humidity, and transmission medium, must be considered as it weakens signals as they propagate. Hence, these factors affect the range and reliability of Bluetooth communications and require careful consideration during device design and deployment in order to achieve a long distance between connected devices.

2.3 Journals review

Journals reviewed are related to Bluetooth technology, such as testing the communication range using beacons, comparison between Bluetooth version 5 and Bluetooth version 4.2, and distance estimation based on the log-distance path loss model. They emphasize the RSSI value differs between different devices, mentioning the advantage provided by the new physical layer introduced in Bluetooth 5 compared to the physical layer in Bluetooth 4, and highlighting the importance of path loss exponent in getting an accurate distance estimation.

2.3.1 Testing the Communication Range of Ibeacon Technology by Boros, Kuffa, and Skýpalová

This study investigated the utilization of iBeacon technology and evaluated signal strength between various devices using RSSI values. The same beacon chip was employed to establish connections with four different receiving devices featuring different Bluetooth versions. These devices included two minicomputers (Raspberry Pi® 4B and Raspberry Pi® Zero) and two mobile phones (Samsung Galaxy A12 and iPhone 13 Pro) (Boros, Kuffa, and Skýpalová, 2022). An application was installed on each receiving device to scan for the beacon and collect real-time RSSI values.

During the tests, the receiving devices were stationed at fixed points (the starting point of measured distance), while the beacon was positioned at specific distances ranging from 1 to 40 metres. Each distance was tested multiple times, and the average of 1000 RSSI values was calculated for analysis. The results recorded as shown in Figure 2.14, indicated a minimal difference in RSSI values between the minicomputers and similarly between the mobile phones. The signal quality between minicomputers and mobile phones varied from -15 dBm to -20 dBm.

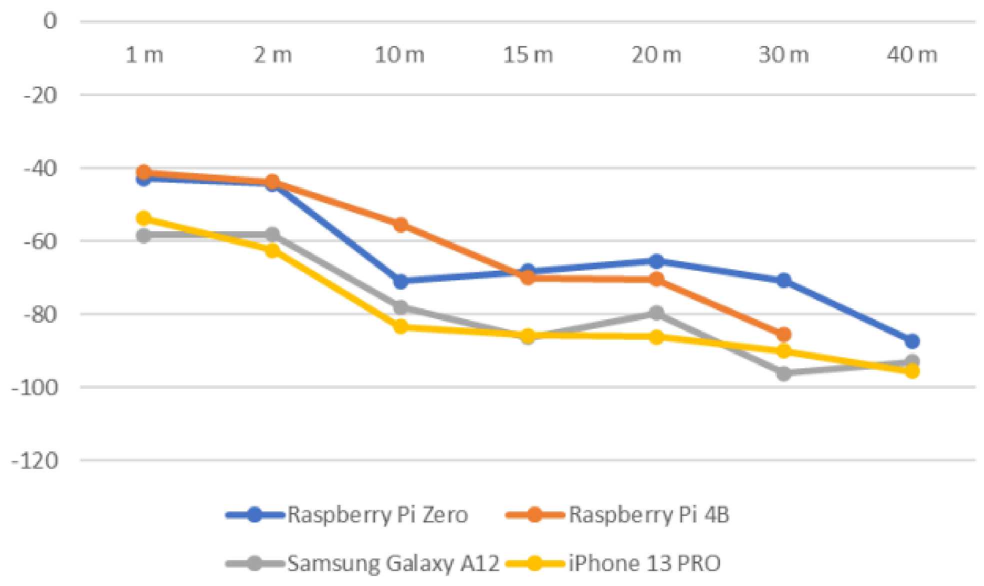


Figure 2.14: Average RSSI value recorded for multiple devices at different distances (Boros, Kuffa, and Skýpalová, 2022).

The study's analysis highlighted that the type of receiving device significantly influenced the RSSI values in iBeacon technology. Furthermore, it emphasized the necessity of involving various devices to determine the maximum range between the receiving device and beacon accurately, as RSSI fluctuations impact result precision significantly. For instance, the study observed a difference of -25 dBm at a distance of 10 meters for the case of the Raspberry Pi Zero, indicating the importance of multiple receiving devices in the test of range measurement.

2.3.2 An analysis of Bluetooth 5 in comparison to Bluetooth 4.2 by Lyatuu

This study compared the performance of Bluetooth 5 and Bluetooth 4.2 in terms of power consumption, range capabilities, and data throughput. It utilised two identical system-on-chip (SoC) devices supporting Bluetooth 5, such as the nRF52840 from Nordic Semiconductor (Lyatuu, 2022). One kit served as the master connected to a PC, while the other acted as the slave. Both kits were loaded with throughput-related firmware from Nordic Semiconductor for testing.

The study examined three physical layers available in Bluetooth 5 which are 1M PHY (supported in Bluetooth 4.2), 2M PHY, and Coded PHY. Tests were conducted over distances ranging from 2 to 450 meters between the two kits. The graph plotted from the results as shown in Figure 2.15, illustrated an inverse relationship between distance and throughput across all Bluetooth versions, indicating that longer distances result in lower throughput. Coded PHY in Bluetooth 5 demonstrated better range capabilities compared to other PHYs, as expected due to its design focusing on longer distances for the connection.

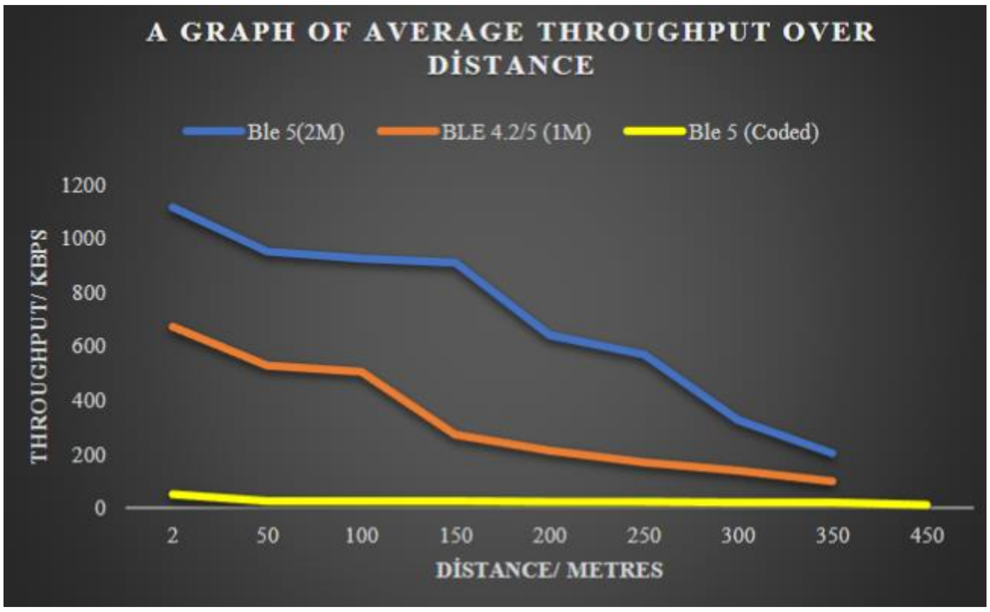


Figure 2.15: Average throughput collected for different PHYs at different distances (Lyatuu, 2022).

Furthermore, power consumption tests were conducted across different PHYs, indicating that 2M PHY consumes less power than others. This was explained due to its higher data rate, allowing for shorter transmission times and increased energy efficiency. The study concluded that Bluetooth 5 was improved compared to Bluetooth 4.2 in terms of range and data rate. The lower data rate of 1M PHY in Bluetooth 4.2 resulted in higher power consumption compared to 2M PHY. Although Coded PHY offered an extended range, its lower data rate required longer data transfer times, and higher energy consumption compared to others.

2.3.3 Calibration of BLE beacons and its impact on distance estimation using the log-distance path loss model by Vanzin and Oyamada

This study examined the use of BLE beacons for object localization, with a focus on estimating distances between beacons and smartphones using RSSI measurements. The beacons were designed using HM-10 BLE boards, and transmit signals to smartphones acting as receiving devices (Vanzin and Oyamada, 2021). Log-distance path loss model was introduced to determine the RSSI values based on calculation as shown in Equation (2.1).

$$RSSI = RSS_{d0} - 10n \times \log_{10}\left(\frac{d}{d0}\right) + X_{\sigma} \tag{2.1}$$

The reference RSSI (RSS_{d0}), representing the RSSI value at a reference distance (d_0). In this study, the reference distance was set to 1m and the reference RSSI was detected using the BLE scanner application installed. The zero-mean normal random variable with standard deviation (X_{σ}) was set to zero as the tests were performed in an open environment without shadows. The path loss exponent (n), indicating the rate of signal loss with distance, is dependent on the environment and obstacles. The n-value was initially set to 2, but it was not optimal due to several factors such as hardware and environment. It was later optimized to obtain more accurate distance estimation. To determine the optimal n-value, the values of d_0 and X_{σ} were set to zero, the equation was rewritten as shown in Equation (2.2). Once the optimal n-value was obtained, the corresponding distance (d) to the specific RSSI can be calculated as shown in Equation (2.3).

$$n = \frac{RSS_{d0} - RSSI}{10 \times \log_{10}(d)} \tag{2.2}$$

$$d = 10^{\left(\frac{RSS_{d0} - RSSI}{10 \times n}\right)} \tag{2.3}$$

The beacons were defined with an individual Minor value and grouped under the same Major value. The experiments were conducted with distances ranging from 0.5 to 5 metres, with variations of 0.5 metres between each distance. The RSSI reading collected from the BLE scanner reflected the actual RSSI for each beacon at different distances. The comparison of calculated RSSI between the n=2 and a custom n-value to the actual RSSI value in each beacon was presented based on the graph plotted as shown in Figure 2.16, Figure 2.17, and Figure 2.18.

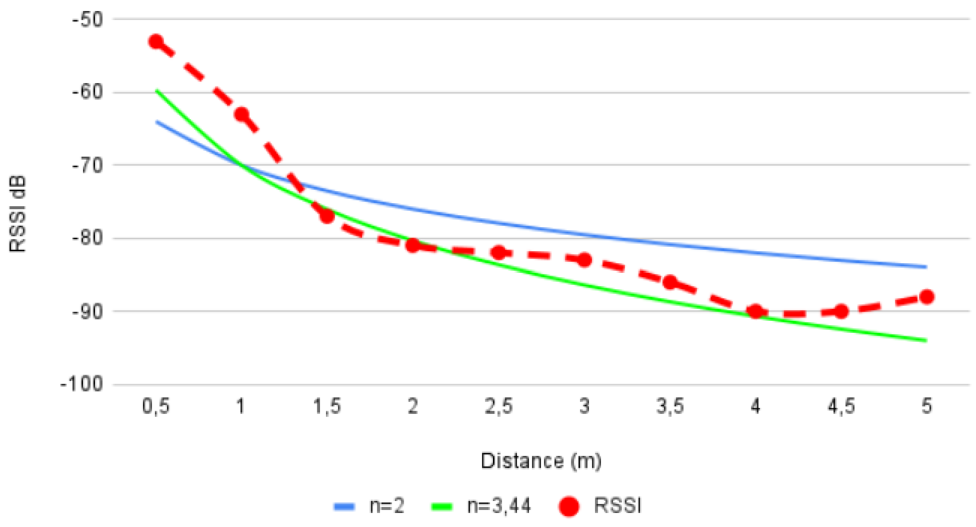


Figure 2.16: Results collected for Beacon A (Vanzin and Oyamada, 2021).

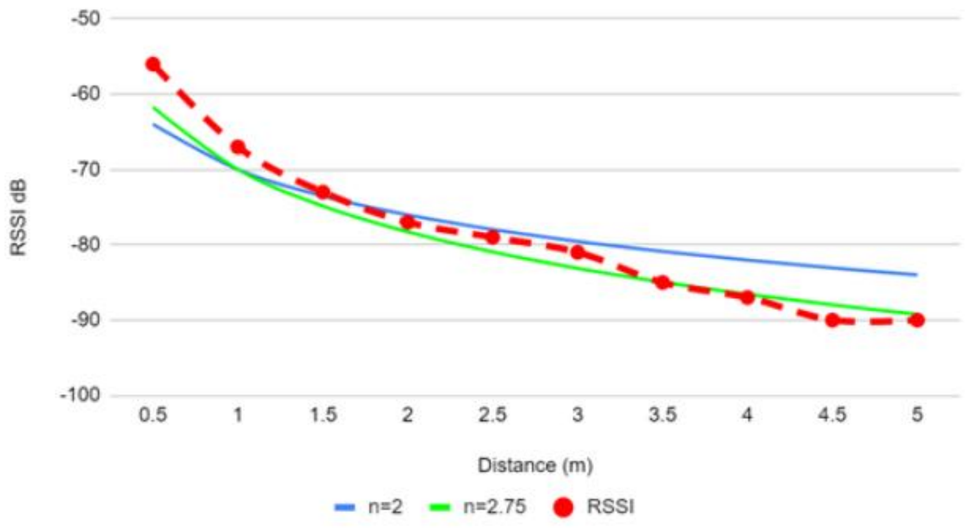


Figure 2.17: Results collected for Beacon B (Vanzin and Oyamada, 2021).

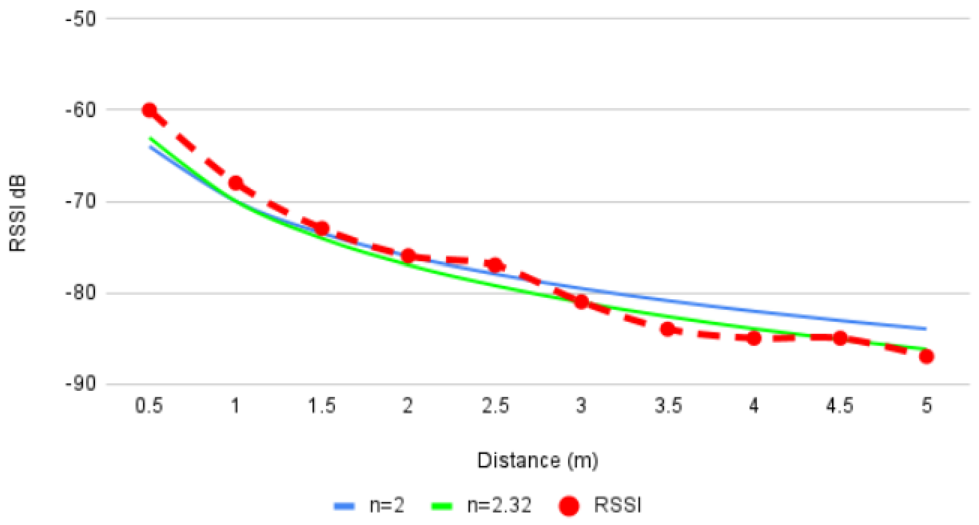


Figure 2.18: Results collected for Beacon C (Vanzin and Oyamada, 2021).

Based on the results recorded, the study identified a custom n-value that yielded closer matches to actual RSSI values compared to the default n=2. Moreover, the study demonstrated the importance of tuning the n-value in the log-distance path loss model, as the percentage error between actual RSSI values and the RSSI calculated using the custom n-values were lower compared to using n=2. The findings suggest that a common n-value is not optimal across different devices, highlighting the need to determine an optimal n-value for each scenario to improve distance estimation accuracy.

2.3.4 Development of Localization Technique using Trilateration Algorithm for E-Puck2 Robot by Harmanda, Priandana, and Hardhienata

This paper explores indoor localization techniques for E-Puck2 robots, specifically focusing on helping the robots determine their positions within a predefined environment. The study implements the Trilateration algorithm to locate the E-Puck2 robot based on Received Signal Strength Indication (RSSI) values obtained from Bluetooth Low Energy (BLE) beacons (Harmanda, Priandana, and Hardhienata, 2020). These beacons are placed in different fixed points, and the RSSI values from the BLE signals are recorded. The RSSI data and the beacons' position are transmitted via a serial port to a server, the Trilateration algorithm is then used to process them to determine the robot's location.

The Trilateration algorithm determines an object's position by using range measurements from three or more reference points with known locations. The object's position can be obtained by finding the intersection based on the distances between the object and the three or more reference points. The Trilateration algorithm applied can be illustrated as shown in Figure 2.19.

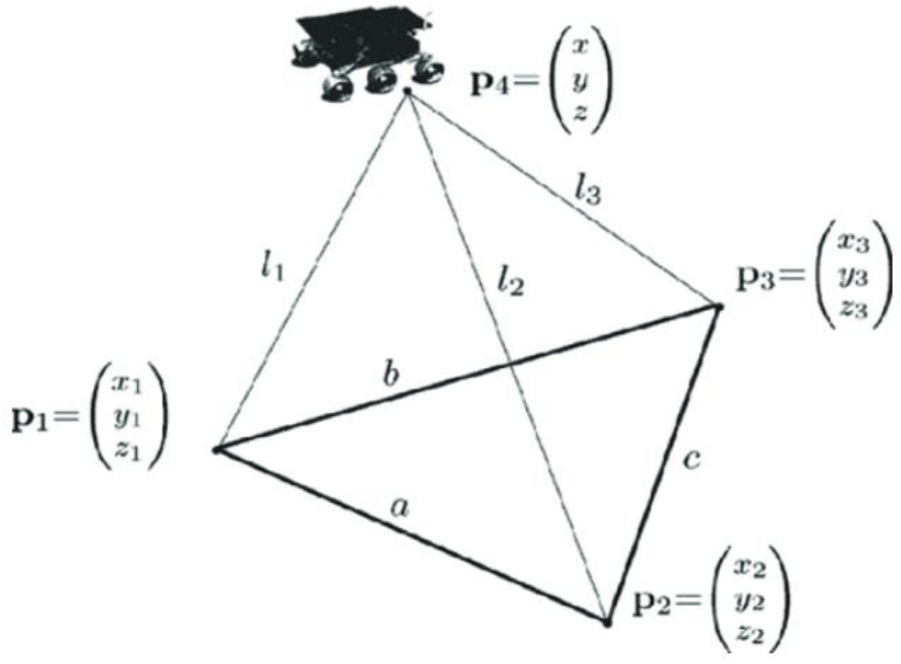

$\mathbf{p}_4 = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$
l_3
l_1
l_2
$\mathbf{p}_3 = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}$
b
$\mathbf{p}_1 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$
c
a
$\mathbf{p}_2 = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$

Figure 2.19: Trilateration algorithm (Harmanda, Priandana, and Hardhienata, 2020).

In this study, BLE signals emitted by the E-Puck2 robot are captured by BLE beacons from different positions for localization purposes. These signals are RSSI values, which are then used to compute the distance between each beacon and the robot. The equation used to determine the position (x, y) of an object in the Trilateration algorithm can be found in Equation (2.4). Based on this equation, the process of calculation for three stations or beacons can be defined as shown in Equation (2.5), Equation (2.6), and Equation (2.7).

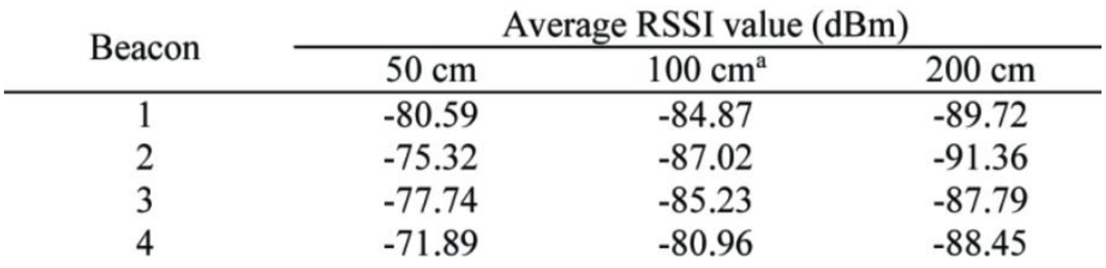
$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = d_i^2 \tag{2.4}$$

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = d_1^2 \tag{2.5}$$

$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = d_2^2 \tag{2.6}$$

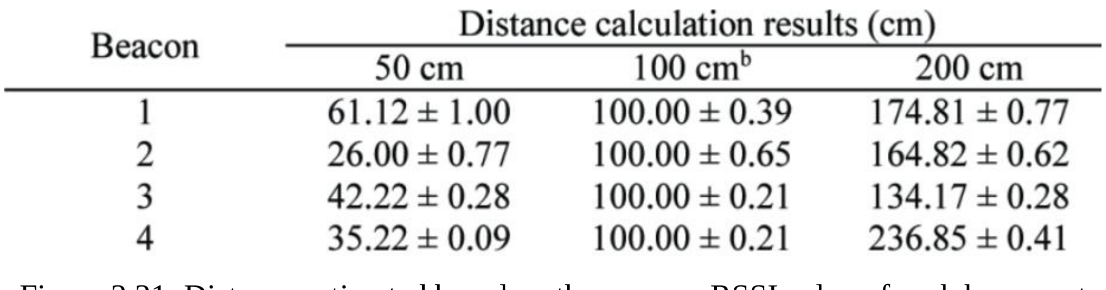
$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = d_3^2 \tag{2.7}$$

Using the distances derived from the RSSI values, the robot's position is determined and compared to its actual position. Additionally, tests were conducted to examine how the distance between the robot and the beacons affects the RSSI values captured. For each distance (50 cm, 100 cm, and 200 cm), 47 RSSI values were recorded, and the average RSSI value for each beacon was calculated as shown in Figure 2.20. This average was then used to estimate the distance and was compared with the actual distance as shown in Figure 2.21.



Beacon	Average RSSI value (dBm)		
	50 cm	100 cm[a]	200 cm
1	-80.59	-84.87	-89.72
2	-75.32	-87.02	-91.36
3	-77.74	-85.23	-87.79
4	-71.89	-80.96	-88.45

Figure 2.20: Average RSSI value of each beacon at different distances (Harmanda, Priandana, and Hardhienata, 2020).



Beacon	Distance calculation results (cm)		
	50 cm	100 cm[b]	200 cm
1	61.12 ± 1.00	100.00 ± 0.39	174.81 ± 0.77
2	26.00 ± 0.77	100.00 ± 0.65	164.82 ± 0.62
3	42.22 ± 0.28	100.00 ± 0.21	134.17 ± 0.28
4	35.22 ± 0.09	100.00 ± 0.21	236.85 ± 0.41

Figure 2.21: Distance estimated based on the average RSSI value of each beacon at different distances (Harmanda, Priandana, and Hardhienata, 2020).

Based on the results, it showed that the estimated distances from the RSSI values did not accurately represent the real distances. Therefore, it was concluded that the RSSI values at specific distances between the robot and the beacons were inconsistent, and the distances calculated from these values were not reliable indicators of the actual distances.

2.3.5 Indoor Localization in BLE using Mean and Median Filtered RSSI Values by Venkatesh, Mittal, and Tammana

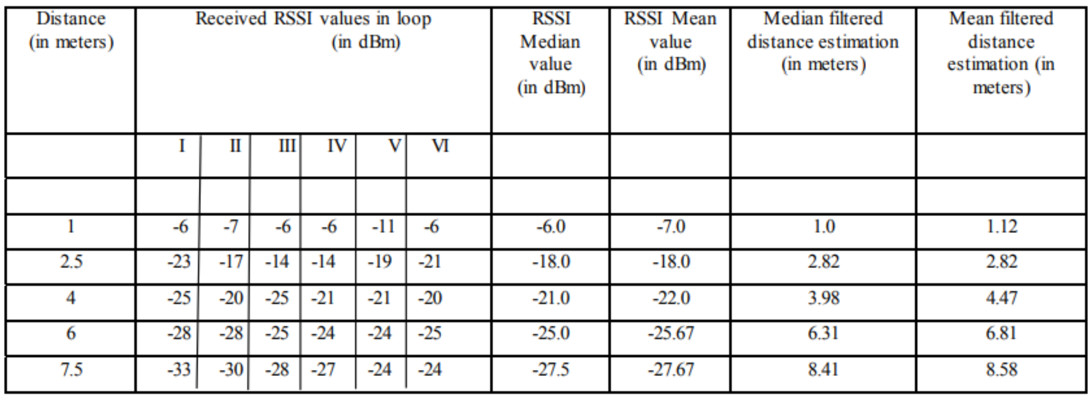
The use of RSSI values is applied for indoor localization applications using BLE technology, but these values are affected by variations due to environmental factors. This paper suggests overcoming this issue by stabilizing the received power using mean and median filters.

When a Bluetooth device starts searching for nearby Bluetooth devices, it measures the RSSI for each detected device. A more negative RSSI value indicates that the peer device is farther away. Various factors such as multipath fading, a high density of obstacles, and low transmitter power can cause fluctuations in RSSI values. These variations can lead to inaccuracies in detecting which transmitter is closest to the scanning device.

To address these fluctuations, a Bluetooth device is placed at a fixed location to collect RSSI data from surrounding Bluetooth beacons and send it to a server in real time. By placing the server at fixed distances from the beacons, it can estimate the variations and provide feedback to the user (mobile terminal) to correct the RSSI in real time. Outliers can be removed using mean and median filters. If there is an odd number of RSSI values, the median is the middle value of the sorted list. If there is an even number, the median is calculated by adding the two middle values and dividing by two.

The Bluetooth controller connects to reference devices using connection handles (Connection handle 1 to Reference-1 and Connection handle 2 to Reference-

2). Once these connections are established, the RSSI values for each connection handle are obtained. The mean and median filters are then applied to stabilize the RSSI values. Tests were conducted at different distances, such as 1 m, 2.5 m, 4 m, 6 m, and 7.5 m, using mean and median filtered values. Based on the results as shown in Figure 22, the distance estimated based on median or mean filtered RSSI is similar to the real distance. The results showed that both filters improved the accuracy of localization.



Distance (in meters)	Received RSSI values in loop (in dBm)						RSSI Median value (in dBm)	RSSI Mean value (in dBm)	Median filtered distance estimation (in meters)	Mean filtered distance estimation (in meters)
	I	II	III	IV	V	VI				
1	-6	-7	-6	-6	-11	-6	-6.0	-7.0	1.0	1.12
2.5	-23	-17	-14	-14	-19	-21	-18.0	-18.0	2.82	2.82
4	-25	-20	-25	-21	-21	-20	-21.0	-22.0	3.98	4.47
6	-28	-28	-25	-24	-24	-25	-25.0	-25.67	6.31	6.81
7.5	-33	-30	-28	-27	-24	-24	-27.5	-27.67	8.41	8.58

Figure 2.22: Distance estimated based on median or mean filtered RSSI value (Venkatesh, Mittal, and Tammana, 2021).

2.3.6 Moving averaging method of RSSI based distance estimation for wireless capsule localization by Hany, Akter, and Hossain

This study introduces a method for estimating the distance of a wireless capsule endoscope (WCE) using the received signal strength indicator (RSSI). Since RSSI weakens with increasing distance, it can be utilized for distance estimation. To address the fluctuation in RSSI measurements, this study proposes a moving average approach for smoothing path loss to estimate the capsule's distance more accurately.

A moving average filter works by replacing each data point with the average of its surrounding points within a defined range as shown in Equation (2.8), effectively acting as a lowpass filter with smoothing characteristics represented by a difference equation.

$$PL_s(d_i) = \frac{1}{2N+1} \sum_{k=i-N}^{i+N} PL(d_k) \tag{2.8}$$

A simulation model was created using MATLAB, featuring eight receiver sensors positioned at the corners of a 3D sensor array. These sensors are employed to localize the capsule as it moves through the small intestine by estimating the distance between the transmitter and receivers.

The path loss data appears randomly scattered due to the unpredictable deviations caused by the non-uniform medium of propagation, causing the distance estimates based on scattered path loss are inaccurate and can differ significantly from the actual distance. To enhance the accuracy of these estimations, the moving averaging method is implemented to address the random nature of path loss. The study also examined how the number of data points affects the accuracy of the proposed moving average method, the best accuracy found for this current application in this study is achieved with 201 data points. By applying a moving average (MA) filter to 201 neighbouring data points (N = 201), the random deviations are minimized, resulting in estimated distances that closely match the actual distances as shown in Figure 2.23.

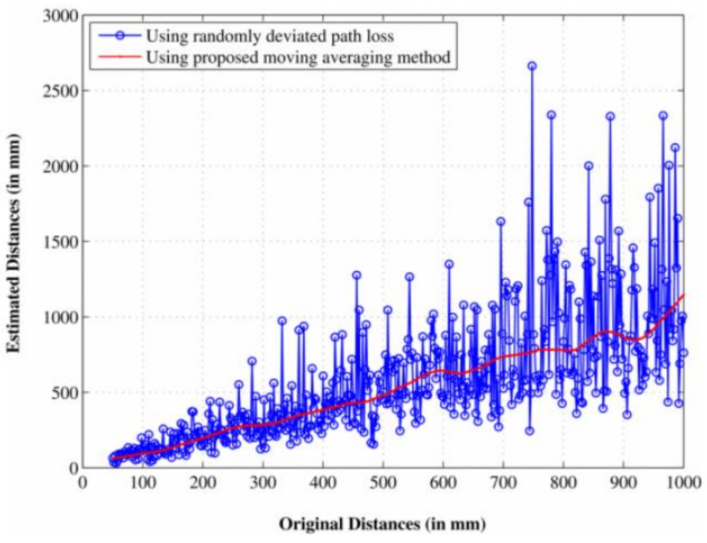


Figure 2.23: Comparison between unfiltered and moving average filtered in Distance estimation (Hany, Akter, and Hossain, 2016).

Based on the observation in Figure 2.24, clearly shows that the distance estimation error was found to be significantly reduced when using the moving average method compared to estimates without it.

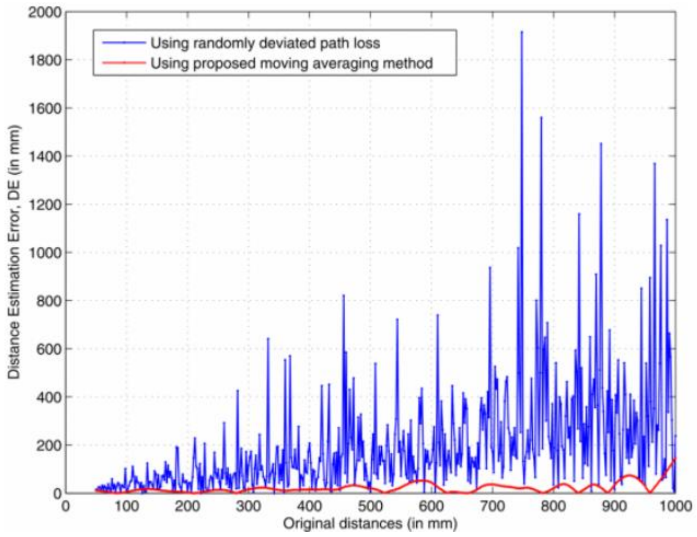


Figure 2.24: Comparison between unfiltered and moving average filtered in Distance estimation error (Hany, Akter, and Hossain, 2016).

2.3.7 Research of RSSI indoor ranging algorithm based on Gaussian-Kalman linear filtering by Zhang, Zhang, and Wan

Due to the complex propagation characteristics of the RSSI signal and the presence of various disturbances, achieving accurate indoor localization solely based on RSSI is challenging and unreliable, this study introduces a filtering algorithm to process the RSSI signal more effectively. The Gaussian-Kalman filtering algorithm is applied to the RSSI signals, demonstrating improved performance in terms of rejecting perturbations, enhancing accuracy, and maintaining stability.

When using the RSSI-based ranging formula to calculate the location of unknown points, one major challenge is determining the values of the environmental parameters such as reference RSSI (A) and path loss exponent (n). The ranging formula indicates that varying these parameters can significantly impact the accuracy of the estimated distance. To ensure the formula accurately reflects the transmission characteristics of the current indoor environment and maintains precise RSSI-based ranging, A and n are optimized through linear regression to find the values most suitable for the specific environment. The linear regression applied to estimate the environmental impact parameters A and n can be illustrated as shown in the following equations.

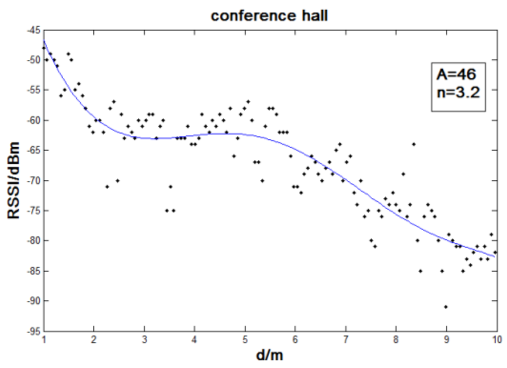
$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{2.9}$$

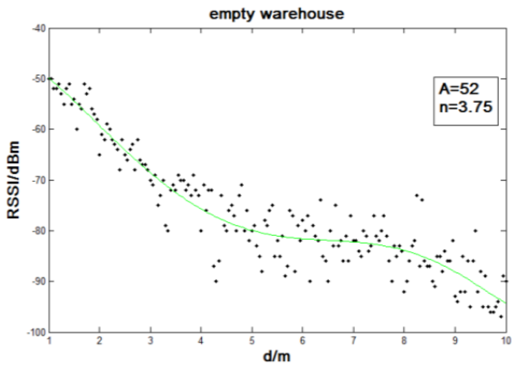
$$\overline{RSSI} = \frac{1}{n}\sum_{i=1}^{n} RSSI_i \tag{2.10}$$

$$\therefore n = \sum_{i=1}^{n}(d - \bar{d})\, RSSI_i / \sum_{i=1}^{n}\left(d - \bar{d}\right)^2 \tag{2.11}$$

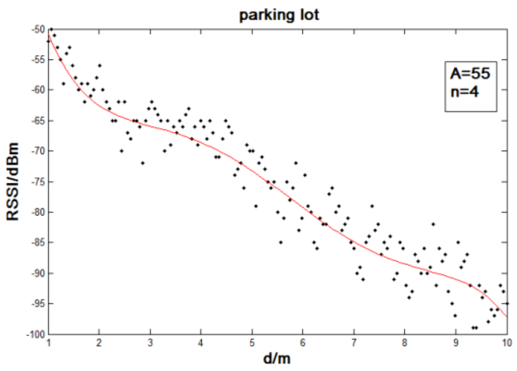
$$\therefore A = \overline{RSSI} - n\bar{d} \tag{2.12}$$

In this study, it utilizes the CC2540 chip from Texas Instruments as the transmitter, constructing a platform based on BLE low-power Bluetooth technology. Its transmission power is set at -4 dBm and is used as the base station to measure RSSI values in various indoor settings. The RSSI signals are then collected in different environments and locations such as conference halls, empty warehouses, and parking lots. The collected data is analyzed using linear regression to assess the RSSI-distance measurement under different conditions as illustrated in Figure 2.25, showing that stable and smooth changes in RSSI values can be obtained using suitable environmental impact parameters.





(a) (b)



(c)

Figure 2.25: Different parameters were recorded for RSSI readings at (a) the conference hall, (b) the empty warehouse, and (c) the parking lot (Zhang, Zhang, and Wan, 2016).

Other than just implementing the linear regression in smoothing the RSSI readings, this study also applied the Gausion filter, Kalman filter, and Gausion-Kalman filtering to further improve the accuracy of measuring results. The comparison between these filtering methods can be illustrated as shown in Figure 2.26. Based on the graph, shows that the relative error of improved filtering (Gaussian-Kalman filtering) gives the lowest relative error compared to others, ensuring the precision and stability of the ranging.

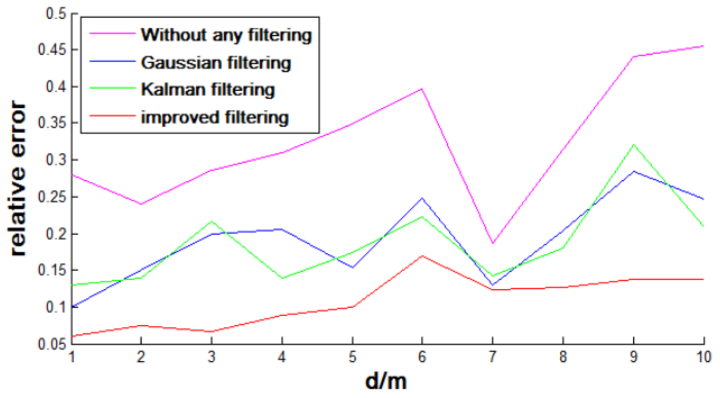


Figure 2.26: Relative error among unfiltered and filtered processes (Zhang, Zhang, and Wan, 2016).

2.3.8 RSSI based indoor localization for smartphone using fixed and mobile wireless node by Gani, OBrien, Ahamed, and Smith

This paper introduces a system for localization that functions effectively both indoors and outdoors. A mathematical model is developed to estimate the location (both distance and direction) of a mobile device using wireless technology. The RSSI values are collected in both indoor and outdoor environments. A low-pass filtering technique was then applied to reduce noise in the RSSI signals caused by various environmental factors. This filtering improves the reliability and usability of the RSSI values as a parameter for estimating the distance and direction of a mobile node from a smartphone. In this case, the Roving Networks WiFly RN-131GSX is used as a mobile Wi-Fi router.

Using both Android and iPhone devices, RSSI values are recorded at distances ranging from 10 feet to 80 feet between the smartphone and the mobile Wi-Fi node, in 2-foot intervals. For each distinct location, these distance-RSSI pairs are stored. The smartphone's accelerometer and magnetometer sensors are used to determine direction relative to true north. The mathematical model is applied to predict the distance and direction of the mobile Wi-Fi node.

The variation of RSSI value and the orientation of the mobile device and the Wi-Fi node is examined. To mitigate the effect of orientation, we collected RSSI values while rotating the smartphone 360 degrees on the horizontal plane. The mean of these values was used to calculate the distance. It was observed that rotating the smartphone minimized the impact of orientation on the RSSI value. In this case, the RSSI value was found to be the strongest when the smartphone directly faced the Wi-Fi node (line of sight). Using this observation, the direction is calculated as the angle from true north at which the strongest RSSI signal was recorded. To compute the heading, filtered accelerometer and magnetometer sensor data are used, while simultaneously gathering RSSI values for each degree of rotation. The mathematical model was then used to predict the distance and direction of the mobile node from the smartphone as shown in Figure 2.27.

- ***Direction***

$$rssi = \{rssi_0, rssi_1, rssi_2, \ldots, rssi_n\} \;,\; for\ 0 \le \theta \le 360$$

$$rssi_{\max} = \{rssi_j \mid rssi_j > \forall_{i,i \neq j} rssi_i\}$$

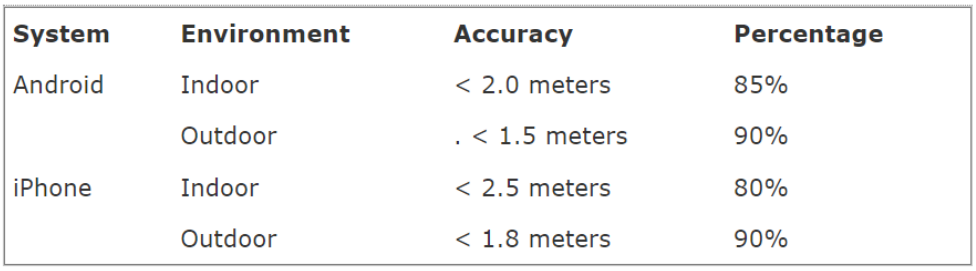
$$direction = \{\theta_i \mid rssi_{\max} = rssi_i,\ current\ direction is\ \theta_i from\ true\ north$$

- ***Distance***

$$(d, rssi_{mean})\ where\ \ rssi_{mean} = \frac{1}{n}\sum_{j=0}^{n} rssi_j \;,\; for\ 0 \le \theta \le 360$$

Figure 2.27: Mathematical model to obtain the direction and distance (Gani, OBrien, Ahamed, and Smith, 2013).

The exponential regression was employed using the Nelder-Mead Simplex Search method, and the resulting regression function was used to estimate the location. A working prototype of this model was developed for both Android and iPhone platforms. An evaluation of the accuracy of the systems for both indoor and outdoor environments was done. The experimental results on these smartphones demonstrated good accuracy, with an error margin of less than 2.5 meters as shown in Figure 2.28.



System	Environment	Accuracy	Percentage
Android	Indoor	< 2.0 meters	85%
	Outdoor	. < 1.5 meters	90%
iPhone	Indoor	< 2.5 meters	80%
	Outdoor	< 1.8 meters	90%

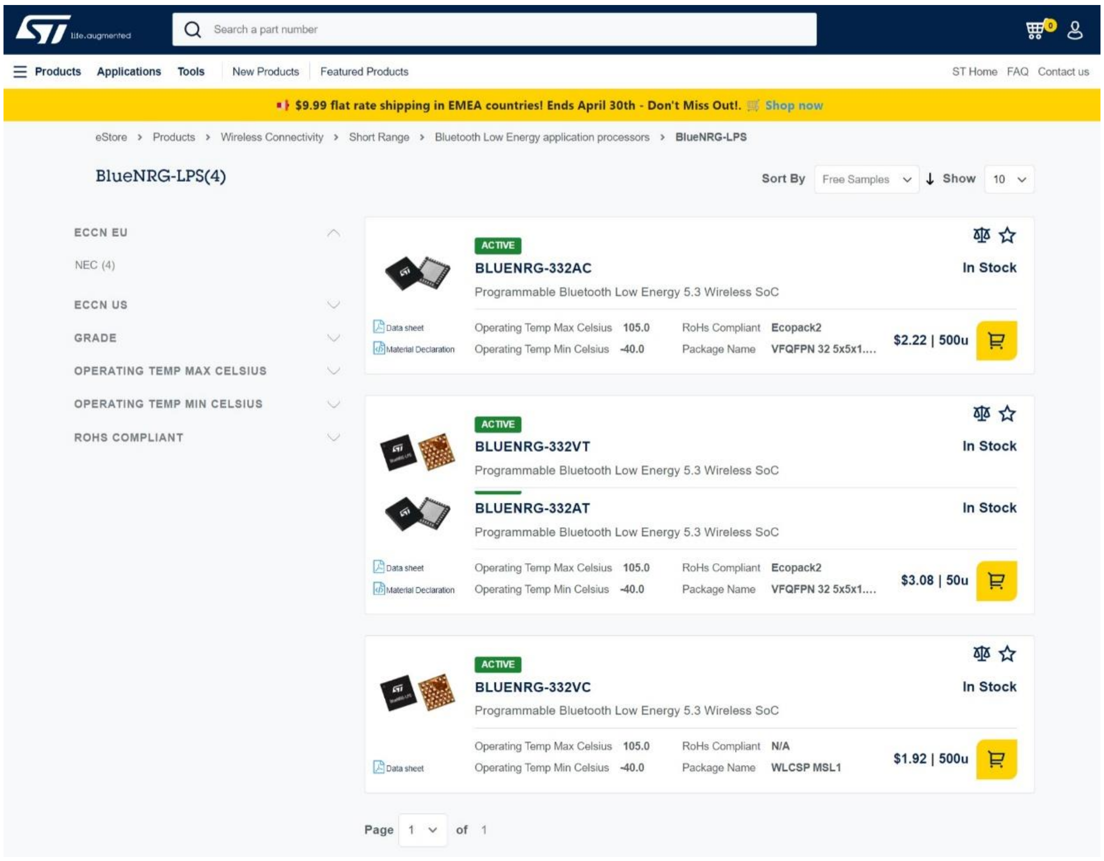
Figure 2.28: Accuracy of the developed system under indoor and outdoor environments (Gani, OBrien, Ahamed, and Smith, 2013).

2.4 Introduction to Hardware

The hardware used in this project is highlighted which is BlueNRG-LPS supported with Bluetooth version 5.3. Furthermore, the development board such as STEVAL-IDB012V1 which is equipped with BlueNRG-332AC SoC from BlueNRG-LPS, is used to perform BLE applications.

2.4.1 BlueNRG-LPS (SoC)

BlueNRG-LPS is a programmable Bluetooth® Low Energy wireless SoC solution that uses STMicroelectronics 2.4 GHz radio IPs for optimal performance and battery life. The order code of BlueNRG-LPS from the STMicroelectronics online website is BlueNRG-332xy and the relevant SoCs available to be ordered such as BLUENRG-332AC, BLUENRG-332VT, BLURNRG-332AT, and BLUENRG-332VC as shown in Figure 2.29 (STMicroelectronics, n.d.).


life.augmented
Search a part number
Products Applications Tools New Products Featured Products
ST Home FAQ Contact us
$9.99 flat rate shipping in EMEA countries! Ends April 30th - Don't Miss Out!. Shop now
Sort By Free Samples Show 10
ECCN EU
NEC (4)
ECCN US
GRADE
OPERATING TEMP MAX CELSIUS
OPERATING TEMP MIN CELSIUS
ROHS COMPLIANT
Page 1 of 1

eStore > Products > Wireless Connectivity > Short Range > Bluetooth Low Energy application processors > BlueNRG-LPS

BlueNRG-LPS(4)

ACTIVE
BLUENRG-332AC — In Stock
Programmable Bluetooth Low Energy 5.3 Wireless SoC
Data sheet | Material Declaration
Operating Temp Max Celsius 105.0 | RoHs Compliant Ecopack2
Operating Temp Min Celsius -40.0 | Package Name VFQFPN 32 5x5x1....
$2.22 | 500u

ACTIVE
BLUENRG-332VT — In Stock
Programmable Bluetooth Low Energy 5.3 Wireless SoC
BLUENRG-332AT — In Stock
Programmable Bluetooth Low Energy 5.3 Wireless SoC
Data sheet | Material Declaration
Operating Temp Max Celsius 105.0 | RoHs Compliant Ecopack2
Operating Temp Min Celsius -40.0 | Package Name VFQFPN 32 5x5x1....
$3.08 | 50u

ACTIVE
BLUENRG-332VC — In Stock
Programmable Bluetooth Low Energy 5.3 Wireless SoC
Data sheet
Operating Temp Max Celsius 105.0 | RoHs Compliant N/A
Operating Temp Min Celsius -40.0 | Package Name WLCSP MSL1
$1.92 | 500u

Figure 2.29: BLUENRG-LPS SoC available (STMicroelectronics, n.d.).

It complies with Bluetooth Low Energy SIG core specification version 5.3, which enables reliable point-to-point connectivity and Bluetooth Mesh networking. It supports features such as 2 Mbps data rate, long-range mode (Coded PHY), advertisement extensions, GATT cache, direction finding such as AoA and AoD, hardware support for simultaneous connection, and more. It also offers advanced security hardware support, including a true random number generator, AES encryption up to 128-bit security processor, public key accelerator, CRC calculation unit, 64-bit unique identifier, and flash read and write protection. BlueNRG-LPS can be configured to support standalone or network processor applications and supports standard and advanced communication interfaces. It operates in a temperature range of -40 °C to +105 °C and a power supply range of 1.7 V to 3.6 V, ensuring a power-saving mode exists to execute the low-power application. Furthermore, two package versions are offered to perform the functionality of BlueNRG-LPS such as the QFN32 and WLCSP36 packages.

2.4.2 STEVAL-IDB012V1 (Development board)

The STEVAL-IDB012V1 as shown in Figure 2.30, is an evaluation board launched by STMicroelectronics to perform the applications of BLE using BlueNRG-LPS SoC (BlueNRG-332AC) in a QFN32 package, which supports various BLE roles such as master, slave, and simultaneous master and slave, and provides long-range communication at a data rate of 2 Mbps, as well as direction finding using Angle of Arrival (AoA) and Angle of Departure (AoD) (STMicroelectronics, n.d.). This evaluation board integrates BLE features such as data length extension, extended advertisement and scanning, GATT caching, LE ping procedure, power control, and path loss monitoring.

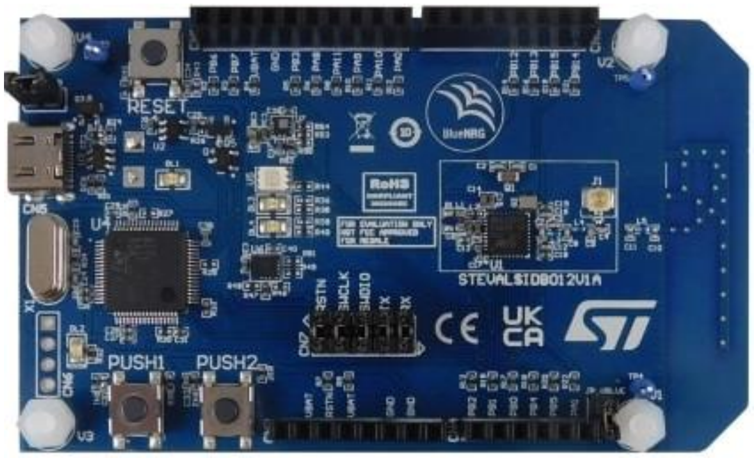


Figure 2.30: STEVAL-IDB012V1 evaluation board (STMicroelectronics, n.d.).

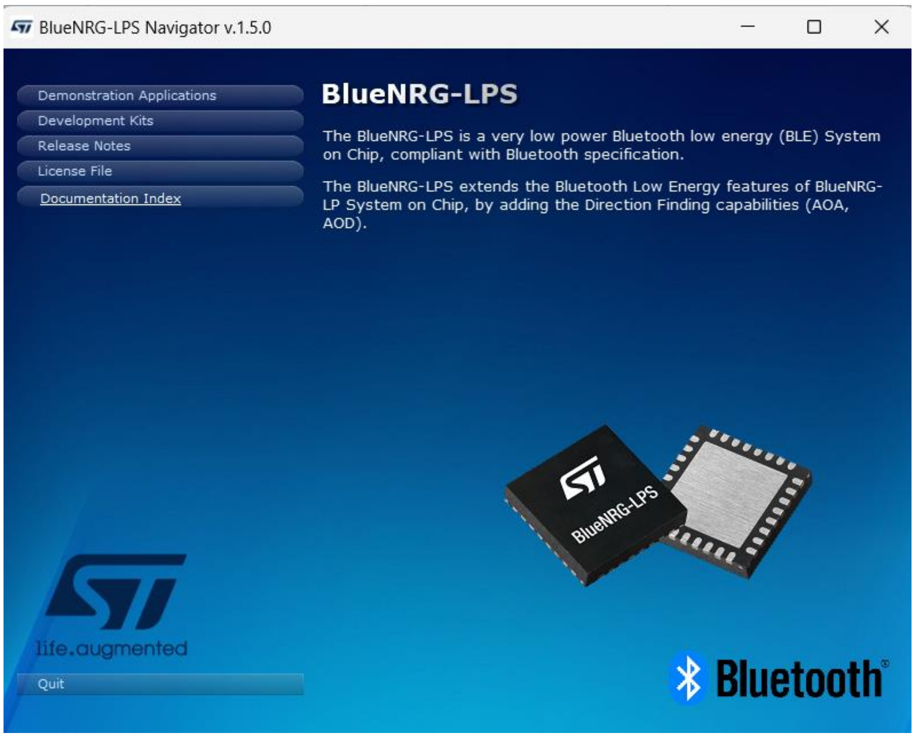
With programmable output power up to +8 dBm and great receiver sensitivity, it ensures efficient radio performance with minimal power consumption. In addition to user LEDs and buttons for interaction, it embeds MEMS sensors for environmental monitoring and provides Arduino R3 connectors for extended functionality. Furthermore, it supports multiple power options as the board can be powered on using either USB, battery, or external power supply.

2.5 Introduction to Software

The software used in this project is introduced which is the BlueNRG-LPS DK SW package used to program the development board that is equipped with BlueNRG-LPS SoC to implement the BLE application. Furthermore, WiSE STUDIO refers to IDE software that provides features to employ user applications such as compiling and debugging. Moreover, AIDA64 for Android is found to allow Android users to discover their phone's hardware and software information. Additionally, nRF Connect for Android refers to an application to be used for scanning the BLE devices, and the feature of generating RSSI values on the connected device is offered as well.

2.5.1 BlueNRG-LPS DK SW package

The STSW-BNRGLP-DK software package as shown in Figure 2.31, supports BlueNRG-LP and BlueNRG-LPS BLE SoC. It offers a comprehensive set of APIs and event callbacks for accessing Bluetooth LE functionality. Additionally, the package includes demo applications illustrating various BLE use cases, each accompanied by header and source files (STMicroelectronics, n.d.).



BlueNRG-LPS

The BlueNRG-LPS is a very low power Bluetooth low energy (BLE) System on Chip, compliant with Bluetooth specification.

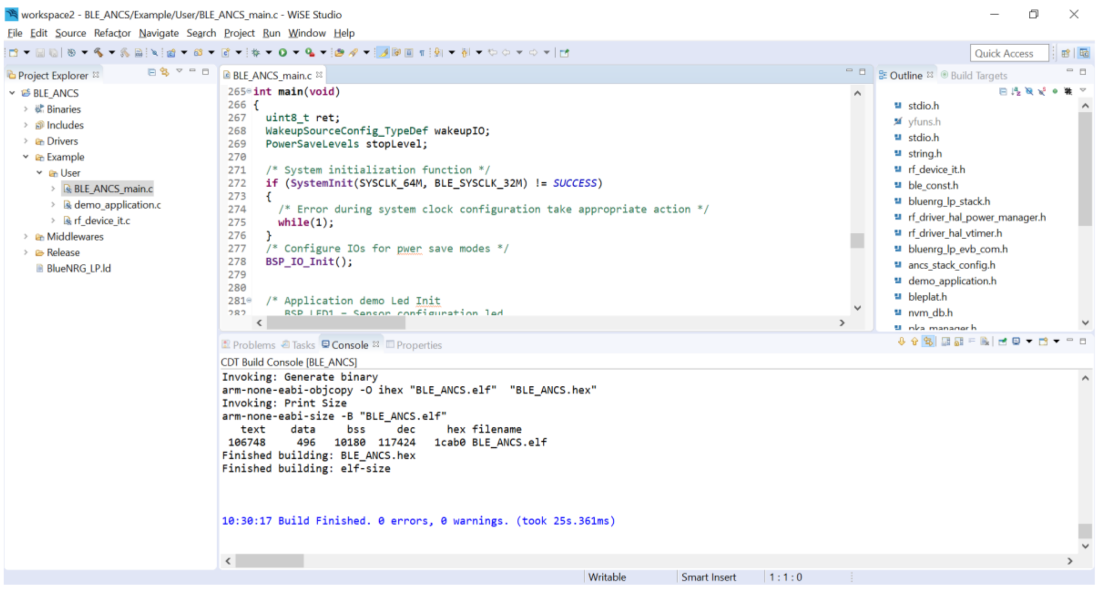
The BlueNRG-LPS extends the Bluetooth Low Energy features of BlueNRG-LP System on Chip, by adding the Direction Finding capabilities (AOA, AOD).

Figure 2.31: STSW-BNRGLP-DK for BlueNRG-LPS (STMicroelectronics, n.d.).

The package provides examples utilizing the low-level driver for the 2.4 GHz radio, serving as references for building other applications leveraging BlueNRG-LP and BlueNRG-LPS radio capabilities. Furthermore, it includes the BlueNRG-LP and BlueNRG-LPS navigators PC applications, offering an intuitive interface for selecting and running demo applications without additional hardware. It also offers a 3D view of available kits and information about associated hardware components.

2.5.2 WiSE STUDIO

The STSW-WISE-STUDIO package provides the WiSE-Studio Eclipse IDE compatible with the BlueNRG family of BLE system-on-chips such as BlueNRG-1, BlueNRG-2, BlueNRG-LPS, BlueNRG-LP and their respective evaluation platforms such as STEVAL-IDB012V1 (STMicroelectronics, n.d.). This IDE supports Eclipse, the GCC toolchain, and GDB-based debugging, providing a comprehensive development environment for creating and debugging user applications. Furthermore, the toolchain is freely available and uses the standard GCC C/C++ compiler. Users can compile, assemble, and link applications for their chosen device and debug via supported SWD channels such as CMSIS-DAP, J-Link or ST-Link/V2. The package also includes the IO Mapper tool, which allows the configuration of pin assignments, peripherals, and device settings for BlueNRG-LP and BlueNRG-LPS, and the generation of associated header and source files for IDE projects. Additionally, WiSE-Studio is compatible with multiple operating systems such as Windows, Linux, and MacOS, with separate software packages available for each platform.



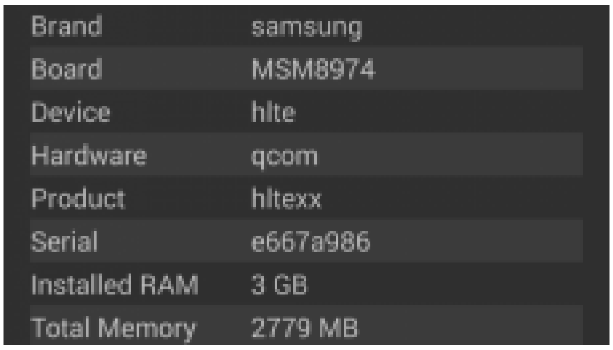
```
int main(void)
{
  uint8_t ret;
  WakeupSourceConfig_TypeDef wakeupIO;
  PowerSaveLevels stopLevel;

  /* System initialization function */
  if (SystemInit(SYSCLK_64M, BLE_SYSCLK_32M) != SUCCESS)
  {
    /* Error during system clock configuration take appropriate action */
    while(1);
  }
  /* Configure IOs for pwer save modes */
  BSP_IO_Init();


  /* Application demo Led Init
     BSP_LED1 - Sensor configuration led
```

```
Invoking: Generate binary
arm-none-eabi-objcopy -O ihex "BLE_ANCS.elf"  "BLE_ANCS.hex"
Invoking: Print Size
arm-none-eabi-size -B "BLE_ANCS.elf"
   text    data     bss     dec     hex filename
 106748     496   10180  117424   1cab0 BLE_ANCS.elf
Finished building: BLE_ANCS.hex
Finished building: elf-size



10:30:17 Build Finished. 0 errors, 0 warnings. (took 25s.361ms)
```

Figure 2.32: WiSE-Studio IDE (STMicroelectronics, n.d.).

2.5.3 AIDA64 for Android

AIDA64 for Android is used to gather and display the hardware and software information of Android devices (FinalWire, 2015). Installing the software into an Android-based phone, users are allowed to detect the details of the phone in terms of hardware and software easily. It is introduced to be a user-friendly application, a clean user interface displays the supported device categories, allowing users to get familiar with the options provided in the application within a short time. Several diagnostic information can be observed for devices such as phones, tablets, and smartwatches such as system information, including device mode, total memory and storage space, and Bluetooth version as shown in Figure 2.33. Additionally, CPU detection and real-time core measurement are available for users to discover. Furthermore, display-related information can be obtained from the application such as screen dimensions, pixel density, and refresh rate. Moreover, the battery level and temperature monitoring features are provided for users to analyse the battery status.



Brand	samsung
Board	MSM8974
Device	hlte
Hardware	qcom
Product	hltexx
Serial	e667a986
Installed RAM	3 GB
Total Memory	2779 MB

Figure 2.33: Showing system information of a device (FinalWire, 2015).

2.5.4 nRF Connect for Android

nRF Connect for Mobile is an application launched by Nordic Semiconductor, is a powerful tool to be installed in mobile devices; it allows users to scan and discover the BLE devices. Several Bluetooth SIG-adopted profiles and Device Firmware Update profiles (DFU) from Nordic Semiconductor or Eddystone from Google are supported by this application (Nordic Semiconductor, n.d.).

Figure 2.34: nRF Connect for Mobile (Nordic Semiconductor, n.d.).

Some features offered in this application are scanning for the BLE devices, showing RSSI graphs, connecting to BLE devices that are connectable, and interpreting the advertisement data. Android users will benefit from the additional features provided in this application, such as GATT server configuration, listing of paired devices, BLE advertising (peripheral role), support for Eddystone beacons and iBeacons, and the ability to simultaneously advertise and maintain multiple connections.

2.5.5 Putty

The terminal emulator program PuTTY is available for free and is open-source. PuTTY is a popular tool for securely connecting to remote systems and carrying out operations like system administration and remote file transfers. PuTTY is a vital tool for managing networks and servers as it supports terminal emulation, which enables users to interact with remote systems. In addition, PuTTY supports SSH, guaranteeing secure connections to remote servers, which is an important feature for safeguarding confidential information. Furthermore, PuTTY facilitates serial communication, allowing connections to hardware such as switches and serial consoles (Rushax, n.d.).

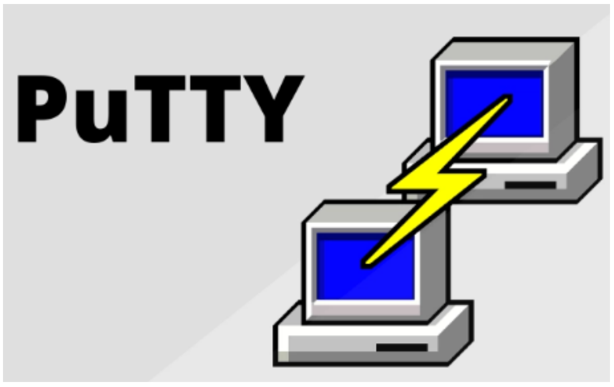


Figure 2.35: Putty application (Rushax, n.d.).

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter describes the system's idea to carry out the goal of the project which is outdoor asset tracking using BLE. The hardware used in the project is mentioned and the experiment continues after the Bluetooth specification supported by the hardware implemented in this study is discussed. Furthermore, the details of the experiment setup in different applications are included as well.

3.2 Bluetooth version of hardware applied

Four devices contribute to the hardware in this study: two mobile devices and two evaluation boards (both STEVAL-IDB012V1). The mobile devices used in this instance are the Mi 9T pro and the Samsung Tab S7. As previously stated, the evaluation board, STEVAL-IDB012V1, has BlueNRG-LPS (BlueNRG-332AC), making it take advantage of Bluetooth 5.3's features. The Android-based mobile devices that need to be tested are supported by AIDA64 for Android and nRF Connect for Android, which collect the necessary information. Determining the system information of devices is the primary goal of using AIDA 64 for Android. The Bluetooth versions of two mobile devices are focused after selecting the system choice offered by the application, as shown in Figure 3.1; both are compatible with Bluetooth 5.0, as shown in Figures 3.2 and 3.3.

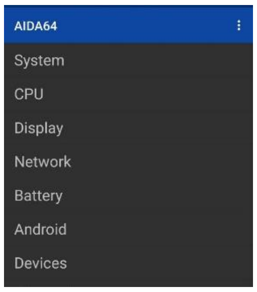
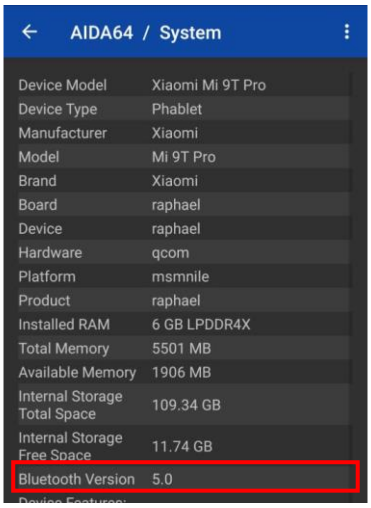


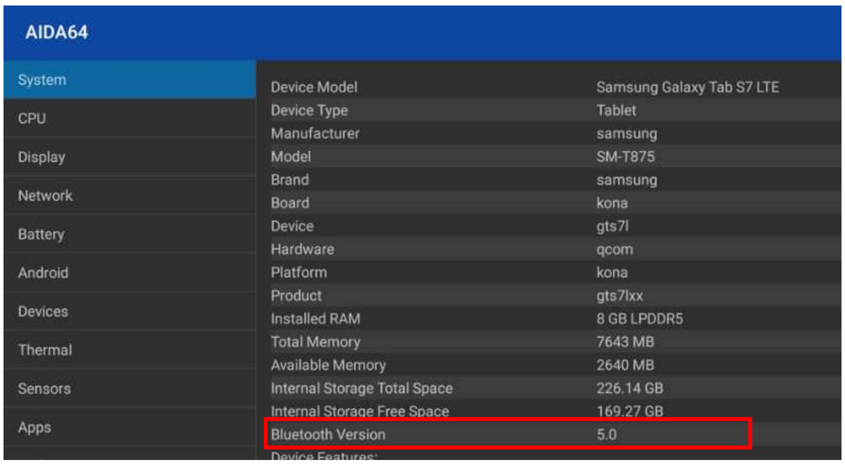
Figure 3.1: Options available in AIDA64 for Andriod.



AIDA64 / System

Device Model	Xiaomi Mi 9T Pro
Device Type	Phablet
Manufacturer	Xiaomi
Model	Mi 9T Pro
Brand	Xiaomi
Board	raphael
Device	raphael
Hardware	qcom
Platform	msmnile
Product	raphael
Installed RAM	6 GB LPDDR4X
Total Memory	5501 MB
Available Memory	1906 MB
Internal Storage Total Space	109.34 GB
Internal Storage Free Space	11.74 GB
Bluetooth Version	5.0

Figure 3.2: Bluetooth version in Mi 9T Pro is 5.0.



AIDA64

Device Model	Samsung Galaxy Tab S7 LTE
Device Type	Tablet
Manufacturer	samsung
Model	SM-T875
Brand	samsung
Board	kona
Device	gts7l
Hardware	qcom
Platform	kona
Product	gts7lxx
Installed RAM	8 GB LPDDR5
Total Memory	7643 MB
Available Memory	2640 MB
Internal Storage Total Space	226.14 GB
Internal Storage Free Space	169.27 GB
Bluetooth Version	5.0

Figure 3.3: Bluetooth version of SAMSUNG Tab S7 is 5.0.

Instead of using the AIDA64 for Android to obtain the Bluetooth version, the application of nRF Connect for Android is used as well to collect information such as hardware, Bluetooth-related features supported, and screen quality. Using the nRF Connect for Android, the Device information option is chosen as shown in Figure 3.4 to detect the Bluetooth capabilities of the devices. The BLE is focused, both mobile devices appear to support the physical layers of Bluetooth 5, including 2M PHY and Coded PHY, as shown in Figure 3.5.

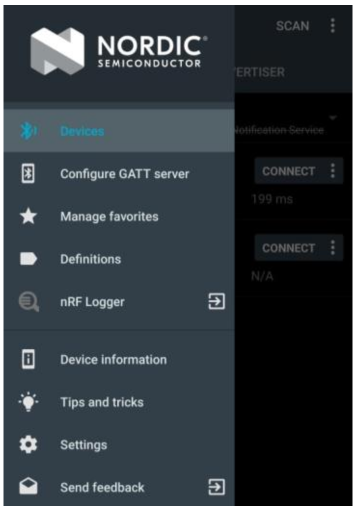
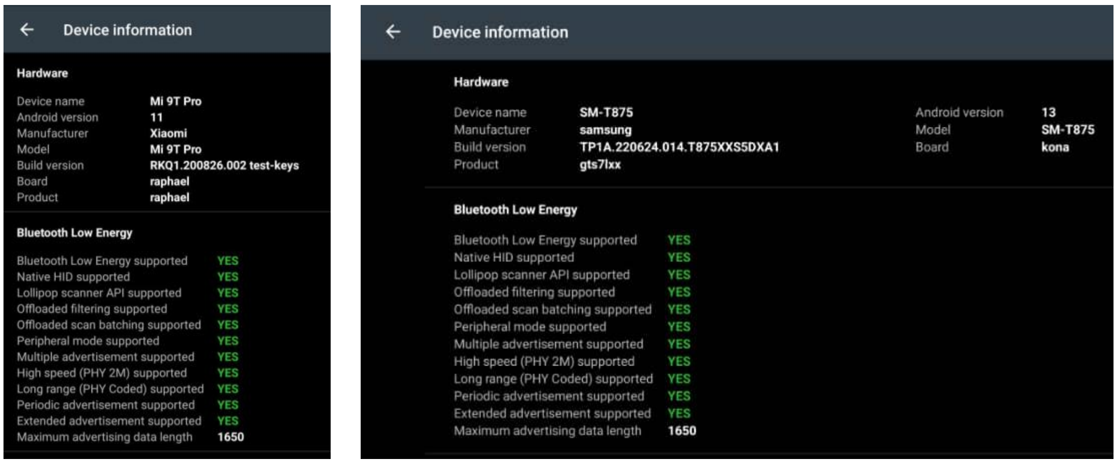


Figure 3.4: Options available in nRF Connect for Android.



Device information

Hardware

Device name	Mi 9T Pro
Android version	11
Manufacturer	Xiaomi
Model	Mi 9T Pro
Build version	RKQ1.200826.002 test-keys
Board	raphael
Product	raphael

Bluetooth Low Energy

Bluetooth Low Energy supported	YES
Native HID supported	YES
Lollipop scanner API supported	YES
Offloaded filtering supported	YES
Offloaded scan batching supported	YES
Peripheral mode supported	YES
Multiple advertisement supported	YES
High speed (PHY 2M) supported	YES
Long range (PHY Coded) supported	YES
Periodic advertisement supported	YES
Extended advertisement supported	YES
Maximum advertising data length	1650

Device information

Hardware

Device name	SM-T875	Android version	13
Manufacturer	samsung	Model	SM-T875
Build version	TP1A.220624.014.T875XXS5DXA1	Board	kona
Product	gts7lxx		

Bluetooth Low Energy

Bluetooth Low Energy supported	YES
Native HID supported	YES
Lollipop scanner API supported	YES
Offloaded filtering supported	YES
Offloaded scan batching supported	YES
Peripheral mode supported	YES
Multiple advertisement supported	YES
High speed (PHY 2M) supported	YES
Long range (PHY Coded) supported	YES
Periodic advertisement supported	YES
Extended advertisement supported	YES
Maximum advertising data length	1650

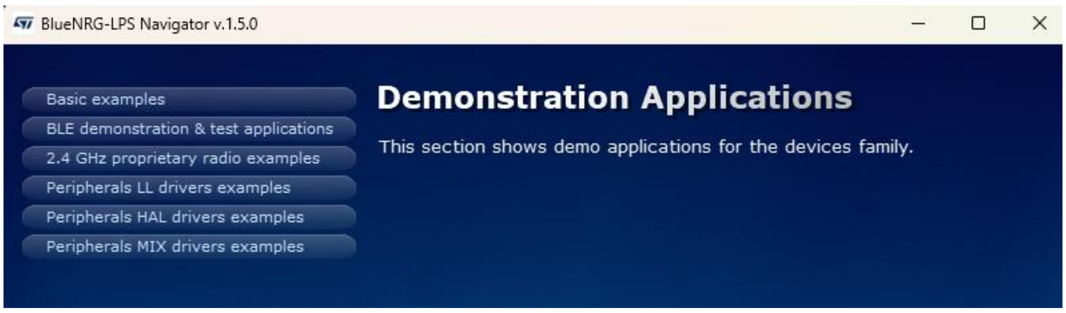
Figure 3.5: 2M PHY and Coded PHY are supported by both devices.

Hence, all of the devices used in this experiment have Bluetooth technology that supports at least Bluetooth version 5.0, allowing for the testing and completion of BLE applications like asset tracking.

3.3 Comparison between 1M PHY and Coded PHY

As mentioned previously, the differences among the physical layers can be classified into data throughput, power consumption, and distance. However, the distance between different physical layers will be the main focus to be compared and discussed in this study. The maximum range between the Bluetooth devices with the setting of 1M PHY and Coded PHY is determined based on the observation of RSSI values. The set of devices applied to determine the effect of different PHY in terms of maximum distance are two evaluation boards, mobile devices such as the Mi 9T pro, and the Samsung Tab S7 to one evaluation board. Hence, the experiment can tell the difference in RSSI readings among different devices under the same settings in physical layer and distance.

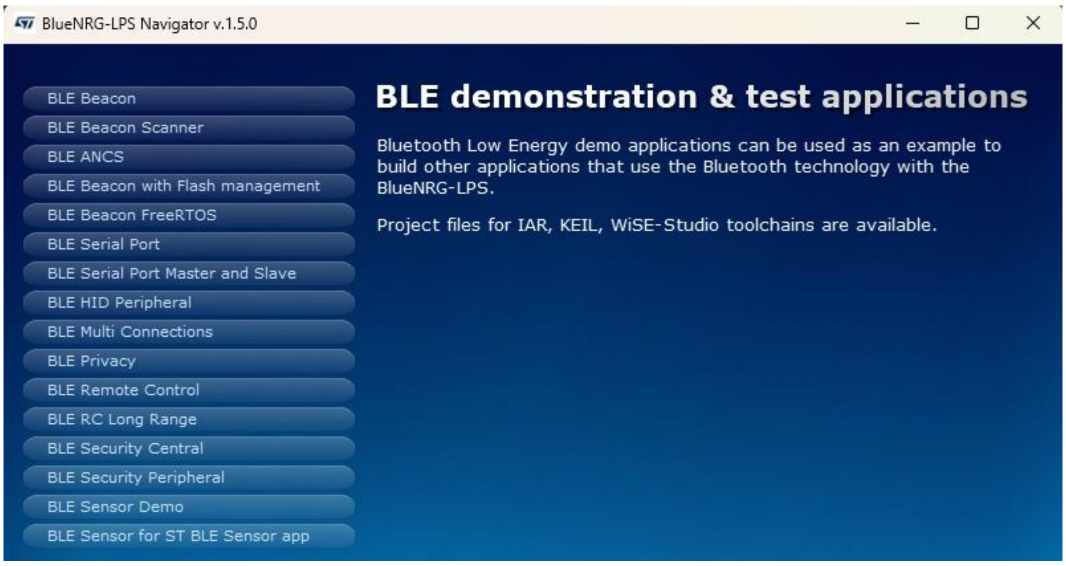
To access the difference in distance between Bluetooth devices under 1M PHY and Coded PHY by using two evaluation boards, the relevant code is needed for them to switch between different physical layers. The BlueNRG-LPS in the evaluation boards is supported by the software package (STSW-BNRGLP-DK), which serves to carry out the software programming to communicate the boards. By choosing the BLE RC Long Range from the BLE demonstration and test applications, as shown in Figures 3.6 and 3.7, the program allows users an example of a way to boot the application code into the boards directly and test the connection under 1M PHY and Coded PHY. The page that appears after selecting the long-range related option outlines how to test the PHYs. It also provides options for operating the board as a server or client, as shown in Figure 3.8. In this case, the idea of PHY testing falls under connection mode. To put it another way, the devices need initially be connected to a 1M PHY, then switching to a Coded PHY is allowable. In this long-range application, three LEDs identified as LED1, LED2, and LED3 are focused, Figure 3.9 illustrates the exact position of these LEDs. Further explanation of these LEDs' indications will be provided.


BlueNRG-LPS Navigator v.1.5.0
Basic examples
BLE demonstration & test applications
2.4 GHz proprietary radio examples
Peripherals LL drivers examples
Peripherals HAL drivers examples
Peripherals MIX drivers examples

Demonstration Applications

This section shows demo applications for the devices family.

Figure 3.6: Applications available to be tested.

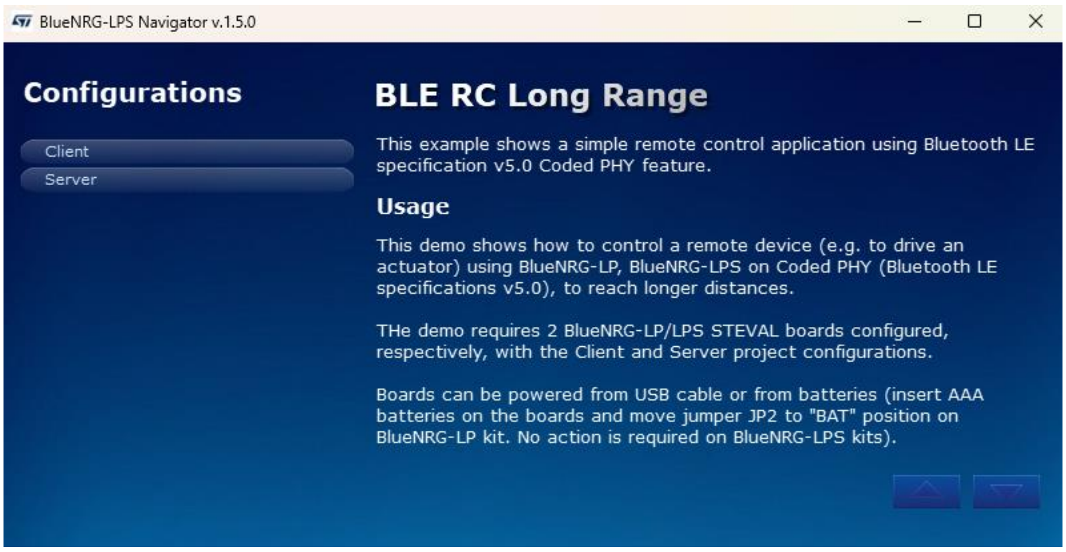


BLE demonstration & test applications

Bluetooth Low Energy demo applications can be used as an example to build other applications that use the Bluetooth technology with the BlueNRG-LPS.

Project files for IAR, KEIL, WiSE-Studio toolchains are available.

Figure 3.7: BLE-related applications available to be tested.



BLE RC Long Range

This example shows a simple remote control application using Bluetooth LE specification v5.0 Coded PHY feature.

Usage

This demo shows how to control a remote device (e.g. to drive an actuator) using BlueNRG-LP, BlueNRG-LPS on Coded PHY (Bluetooth LE specifications v5.0), to reach longer distances.

THe demo requires 2 BlueNRG-LP/LPS STEVAL boards configured, respectively, with the Client and Server project configurations.

Boards can be powered from USB cable or from batteries (insert AAA batteries on the boards and move jumper JP2 to "BAT" position on BlueNRG-LP kit. No action is required on BlueNRG-LPS kits).

Figure 3.8: Client and server selection for long-range applications.

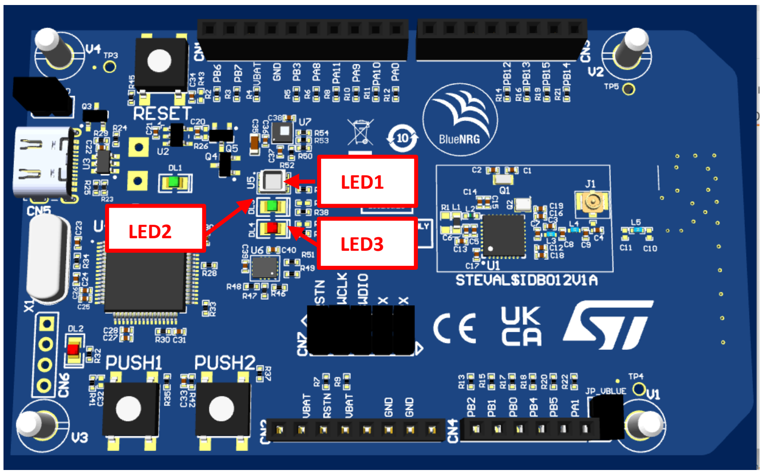


Figure 3.9: LED1, LED2, and LED3 on STEVAL-IDB012V1.

One board is booted as a client and the other performs the function of a server via a USB cable to enable communication between the two boards. Both boards can be powered on via a USB cable or a battery after they are successfully programmed. The LED2 of a board blinks once it is powered on, and searches for another board. As demonstrated in Figure 3.10, once both boards are powered on, LED2 will be on constantly which indicates that they are connected, and the blinking of LED3 signifies that they are connected or communicating with one another. When evaluating Bluetooth device range under various PHYs, the 1M PHY is initially applied to both boards. In this stage of testing the 1M PHY, the LEDs focused are only LED2 and LED3.

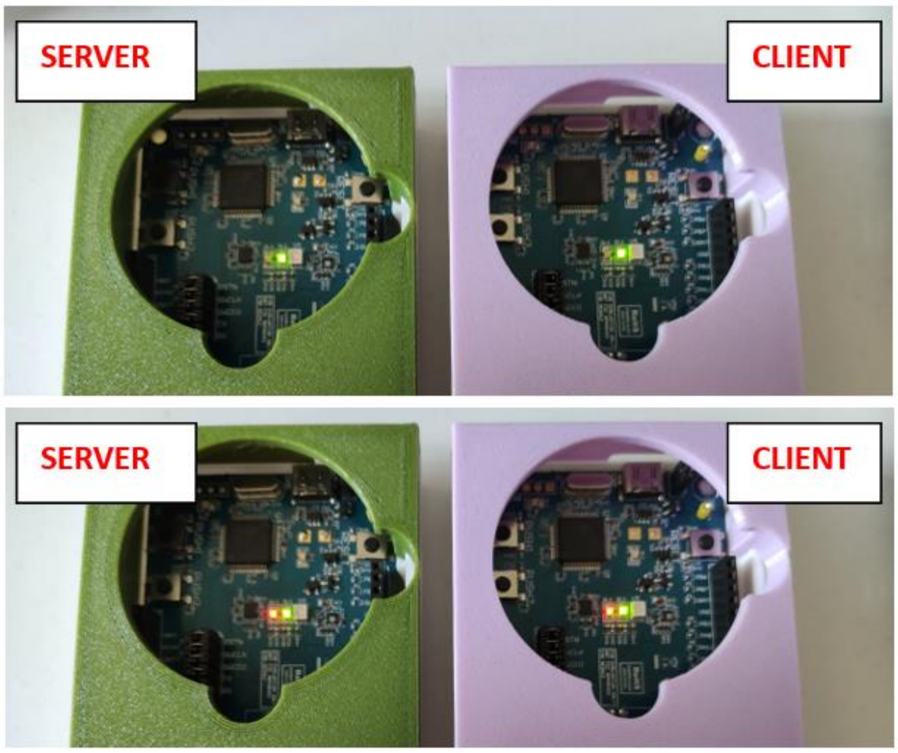


Figure 3.10: Binking LED3 indicates boards are communicating.

One of the boards is set up at the end as the starting point, while the other board is moved to a different location. Once LED3 is no longer powered on, it indicates the devices are disconnected, and the distance under a specific PHY is examined, as the state of LED3 indicates the state of communication. As observed in Figure 3.11, to effectively switch their physical layer to Coded PHY, one of the boards' PUSH1 buttons must be pressed and the LED1 turns on constantly, indicating the transition from 1M PHY and Coded PHY is successful. To get the distance result for Coded PHY, the procedure used before to get the distance of devices under 1M PHY is repeated between two boards after pressing the button.

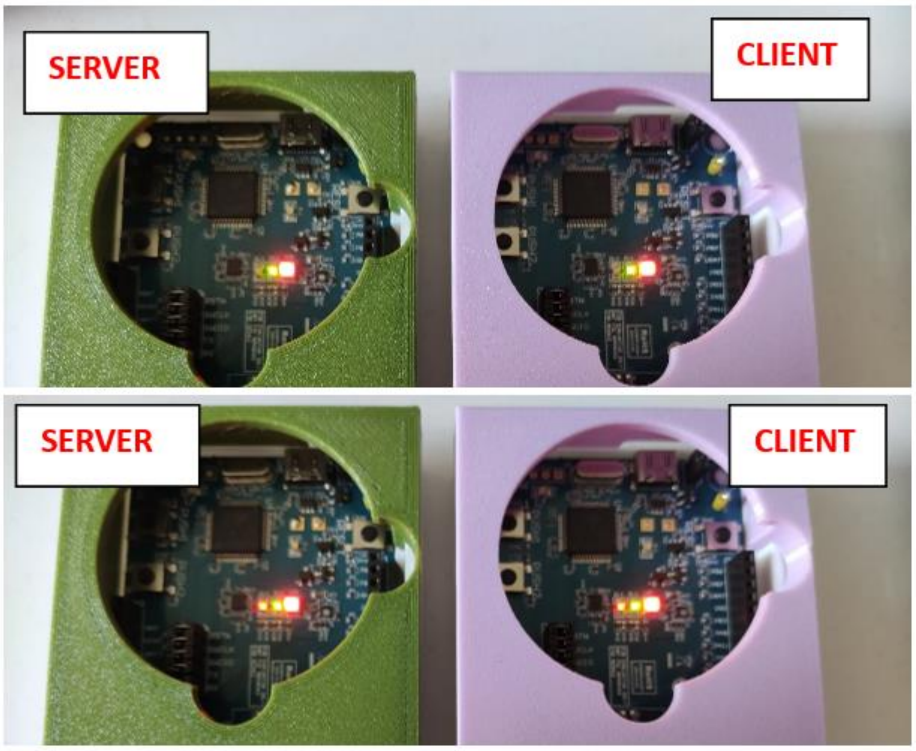


Figure 3.11: LED3 blinks and LED1 is on, indicating Coded PHY to be used.

To perform the test between different physical layers using a mobile device and an evaluation board, the software package (STSW-BNRGLP-DK) needs to be used to boot the code into the board. However, one evaluation board only is applied in this case and it acts as the server. Mobile devices such as the Mi 9T pro and the Samsung Tab S7 are treated to be client. To scan and discover the evaluation board, the application (nRF Connect for Mobile) is needed for mobile devices to communicate with the evaluation board. Once the server is booted into the evaluation board, the LED2 starts to blink. The application will search for the surrounding Bluetooth devices, and the board will be shown in the scanning list as shown in Figure 3.12.

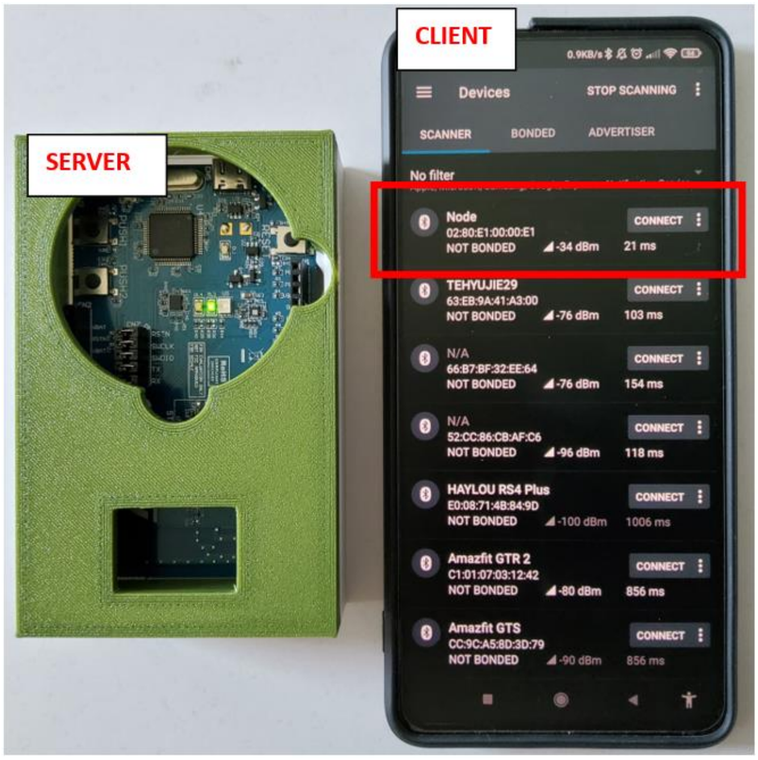


Figure 3.12: Evaluation board (Node) is discovered in the application.

By clicking the *connect* option provided in the application, they will communicate with each other under connection mode, and the services and characteristics of the board can be accessed. In this case, the LED3 is no longer to be the indicator to check the status of communication, instead, we observe it based on the application from the side of mobile devices. Once the mobile device is disconnected from the evaluation board, a disconnected message will be printed out in the application, and the maximum distance for the specific physical layer will be determined.

The 1M PHY is initially accessed, and the board and the mobile device can be switched from 1M PHY to Coded PHY by pressing the PUSH1 button. Alternatively, we can set the preferred phy from the application to modify the desired physical layer as shown in Figure 3.13. By selecting the Coded PHY to be the preferred physical layer as shown in Figure 3.14, the Coded PHY is updated from 1M PHY as shown in Figure 3.15. The status of the LED1 on the evaluation board is powered on, showing the Coded PHY is implemented successfully in the communication between two devices as shown in Figure 3.16.

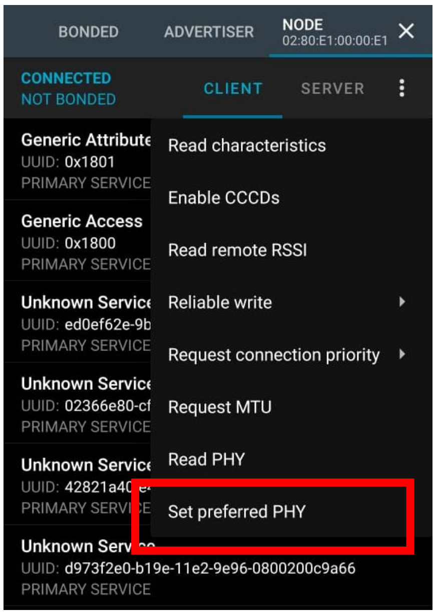
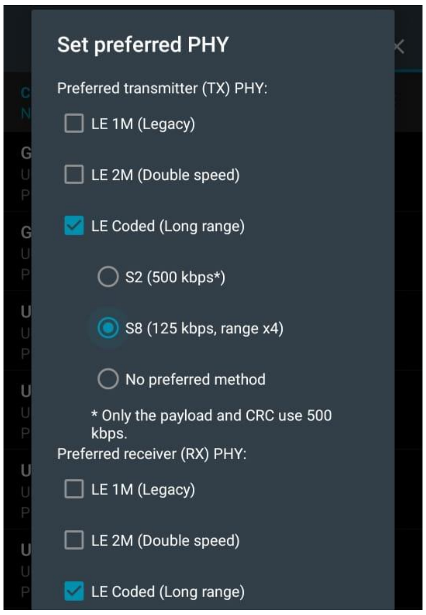


Figure 3.13: Set preferred PHY option is available in the application.



Set preferred PHY

Preferred transmitter (TX) PHY:

[ ] LE 1M (Legacy)

[ ] LE 2M (Double speed)

[✓] LE Coded (Long range)

( ) S2 (500 kbps*)

[✓ S8 (125 kbps, range x4)]

( ) No preferred method

* Only the payload and CRC use 500 kbps.

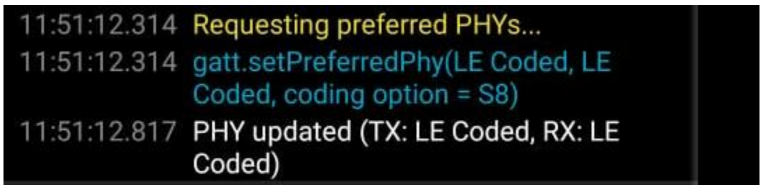
Preferred receiver (RX) PHY:

[ ] LE 1M (Legacy)

[ ] LE 2M (Double speed)

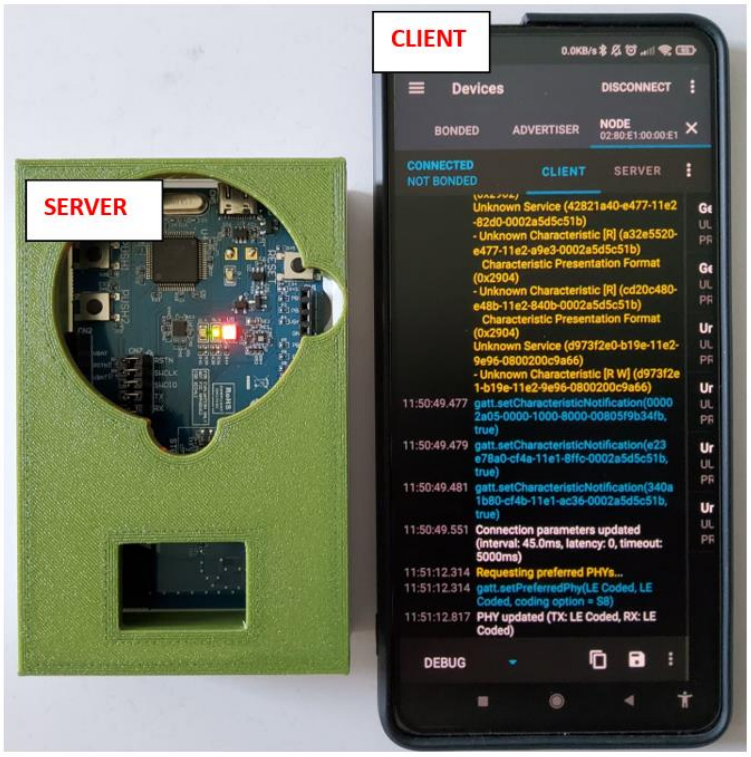
[✓] LE Coded (Long range)

Figure 3.14: Coded PHY can be switched from 1M PHY.



```
11:51:12.314  Requesting preferred PHYs...
11:51:12.314  gatt.setPreferredPhy(LE Coded, LE
              Coded, coding option = S8)
11:51:12.817  PHY updated (TX: LE Coded, RX: LE
              Coded)
```

Figure 3.15: Coded PHY is updated after setting Coded PHY as the preferred PHY.



```
                Unknown Service (42821a40-e477-11e2
                -82d0-0002a5d5c51b)
                - Unknown Characteristic [R] (a32e5520-
                e477-11e2-a9e3-0002a5d5c51b)
                  Characteristic Presentation Format
                (0x2904)
                - Unknown Characteristic [R] (cd20c480-
                e48b-11e2-840b-0002a5d5c51b)
                  Characteristic Presentation Format
                (0x2904)
                Unknown Service (d973f2e0-b19e-11e2-
                9e96-0800200c9a66)
                - Unknown Characteristic [R W] (d973f2e
                1-b19e-11e2-9e96-0800200c9a66)
11:50:49.477    gatt.setCharacteristicNotification(0000
                2a05-0000-1000-8000-00805f9b34fb,
                true)
11:50:49.479    gatt.setCharacteristicNotification(e23
                e78a0-cf4a-11e1-8ffc-0002a5d5c51b,
                true)
11:50:49.481    gatt.setCharacteristicNotification(340a
                1b80-cf4b-11e1-ac36-0002a5d5c51b,
                true)
11:50:49.551    Connection parameters updated
                (interval: 45.0ms, latency: 0, timeout:
                5000ms)
11:51:12.314    Requesting preferred PHYs...
11:51:12.314    gatt.setPreferredPhy(LE Coded, LE
                Coded, coding option = S8)
11:51:12.817    PHY updated (TX: LE Coded, RX: LE
                Coded)
```

Figure 3.16: LED1 of the board is on, indicating Coded PHY is applied.

3.3.1 Maximum distance

To perform the test using two evaluation boards, both of the boards are switched on and connected initially. One of the boards (Board A) is placed at the starting point (point A), and the other board (Board B) is then carried and moved away from Board A until the LED3 stops blinking as shown in Figure 3.17. At that moment, the current position (point B) of Board B is treated to be the ending point, and the distance between the point A and point B will be considered as the maximum distance as illustrated as shown in Figure 3.18.

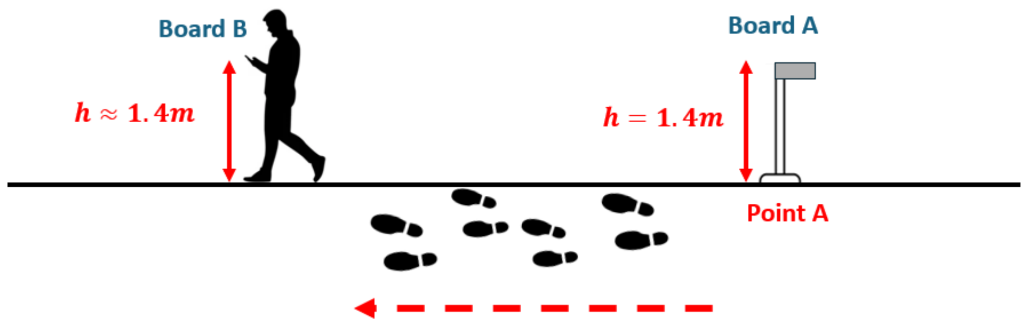


Figure 3.17: Two boards are separated to get the distance measurement.

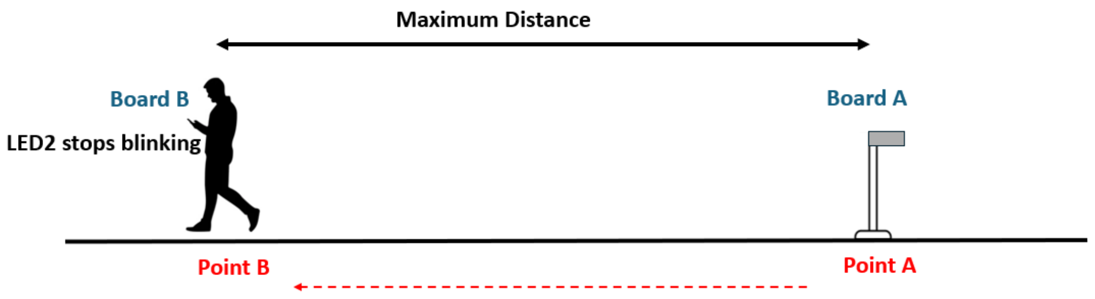


Figure 3.18: Maximum distance is obtained when LED2 stops blinking.

To ensure the accuracy of the results in getting the maximum distance, we will keep moving forward at 50m (point C) from point B to observe the status of LED2 as shown in Figure 3.19. If the LED2 is not powered on or blinking from point B to point C, then the distance between point A and point B is the maximum distance; otherwise, the maximum distance measurement will be continued. This test is to improve the results of distance measurement by considering the low data throughput in Coded PHY.

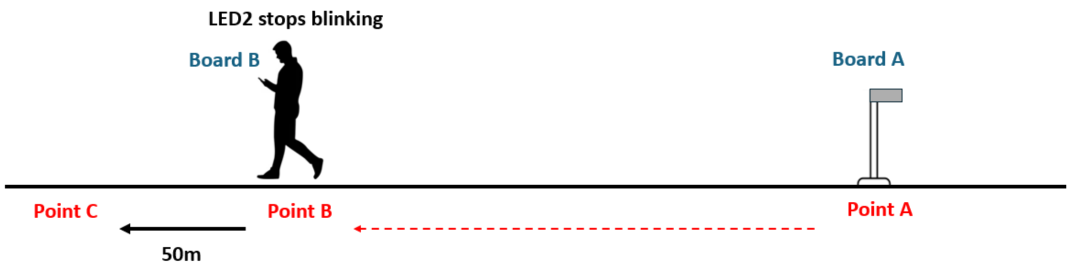


Figure 3.19: Additional 50m to be tested after LED2 stops blinking.

To conduct the distance test between the mobile device and one evaluation board, the previous process is repeated as shown in Figure 3.20, but the disconnected message is the signal to get the maximum distance between the two devices.

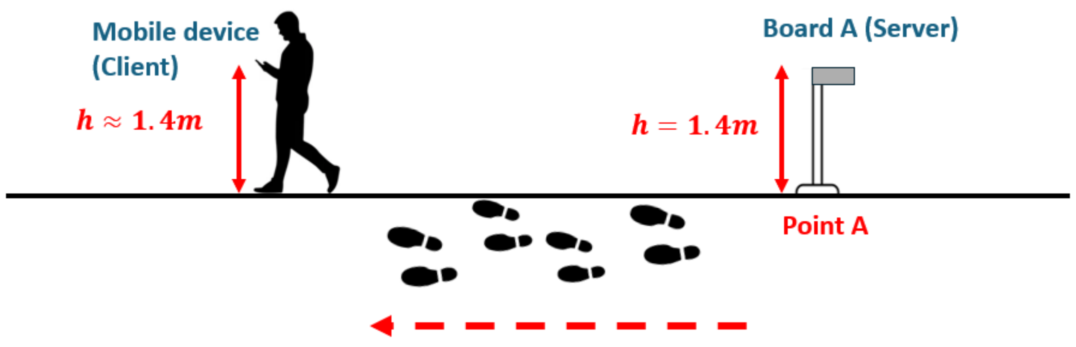


Figure 3.20: Distance measurement between a mobile device and one evaluation board.

A flowchart is used to illustrate the operation of conducting distance measurement as shown in Figure 3.21. After collecting the distance between devices for 1M PHY, the Coded PHY is then switched from 1M PHY and the distance measurement is repeated to get the maximum distance.

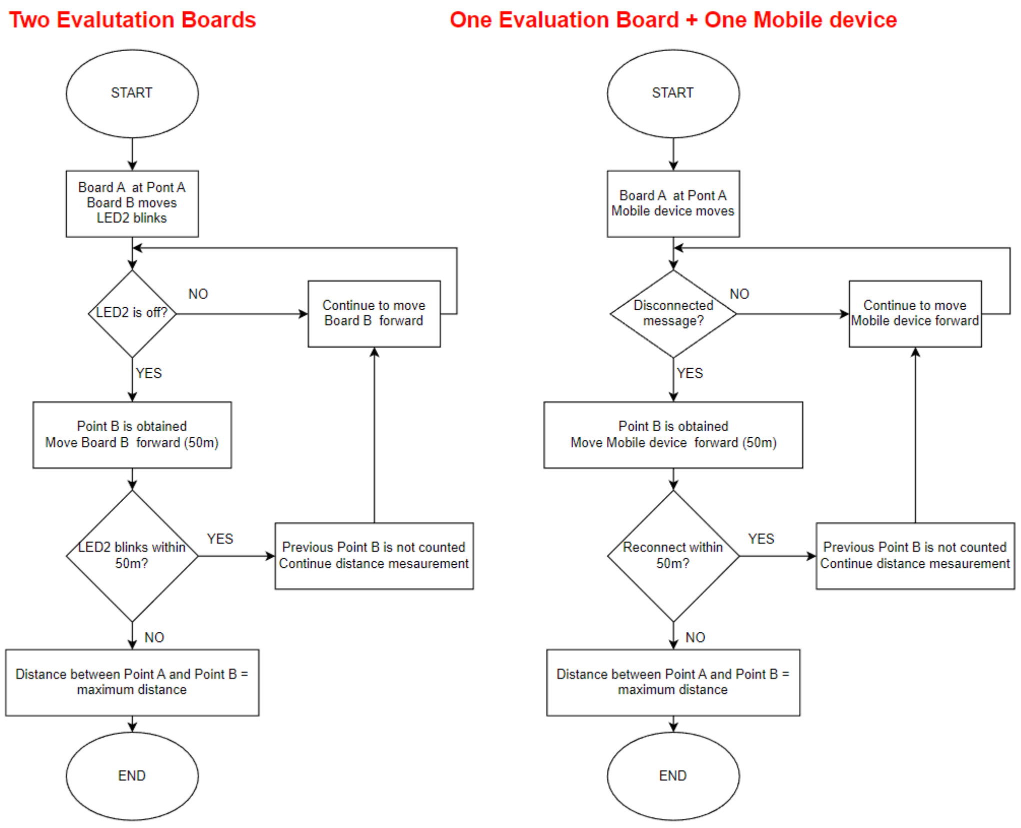

Two Evalutation Boards
START
Board A at Pont A
Board B moves
LED2 blinks
LED2 is off?
NO
Continue to move
Board B forward
YES
Point B is obtained
Move Board B forward (50m)
LED2 blinks within 50m?
YES
Previous Point B is not counted
Continue distance measurement
NO
Distance between Point A and Point B = maximum distance
END
One Evaluation Board + One Mobile device
START
Board A at Pont A
Mobile device moves
Disconnected message?
NO
Continue to move
Mobile device forward
YES
Point B is obtained
Move Mobile device forward (50m)
Reconnect within 50m?
YES
Previous Point B is not counted
Continue distance measurement
NO
Distance between Point A and Point B = maximum distance
END

Figure 3.21: Flowchart of distance measurement

3.3.1.1 Tested in different environments (Indoor and Outdoor)

To test the effect of both physical layers in distance, the environment is one of the factors to be considered. In this case, the environments to be examined are indoor and outdoor. The indoor environment test is conducted in building (Block E) in the campus area, while the outdoor environment test is performed in open area spaces. The measuring tape is used to measure the indoor distance to ensure accurate results are collected, while Google Maps is the software tool for us to use in measuring the distance between the two devices based on the measure distance option available as shown in Figure 3.22, for users to get the distance from point A to point B.

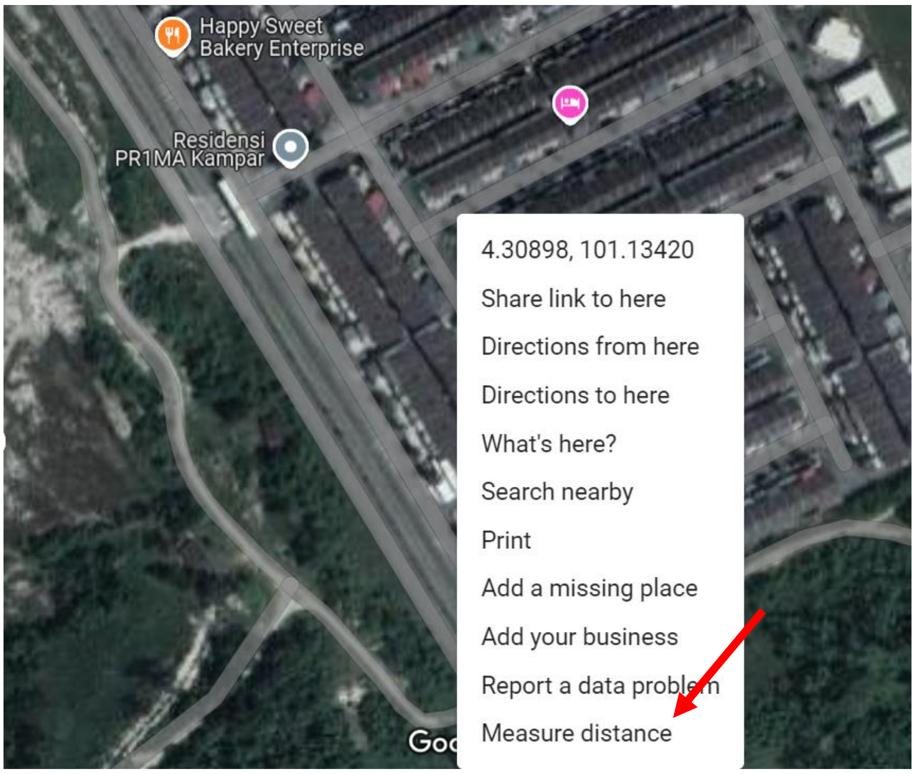


Figure 3.22: Measure distance option is available in Google Maps.

3.3.1.2 Tested in different heights

Other than examining the difference in environments, height is also considered to be one of the important factors in distance measurement. In this case, an extension pole is used and the boards or mobile devices are attached to the pole to get different height settings as shown in Figure 3.23. In this case, the height to be adjusted is 1.4 m, 2 m, and 3 m to perform the distance test as shown in Figure 3.24. Once the pole is set to the desired height and the device is attached to it, then the distance measurement is started.

Type text her

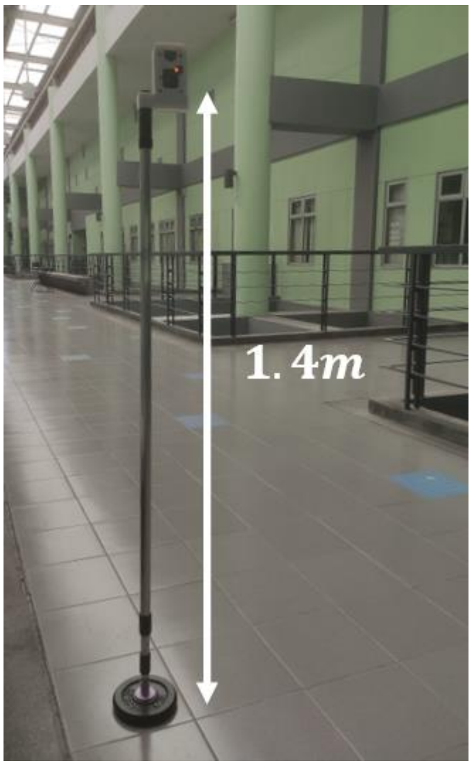


Figure 3.23: A board is attached to an extension pole.

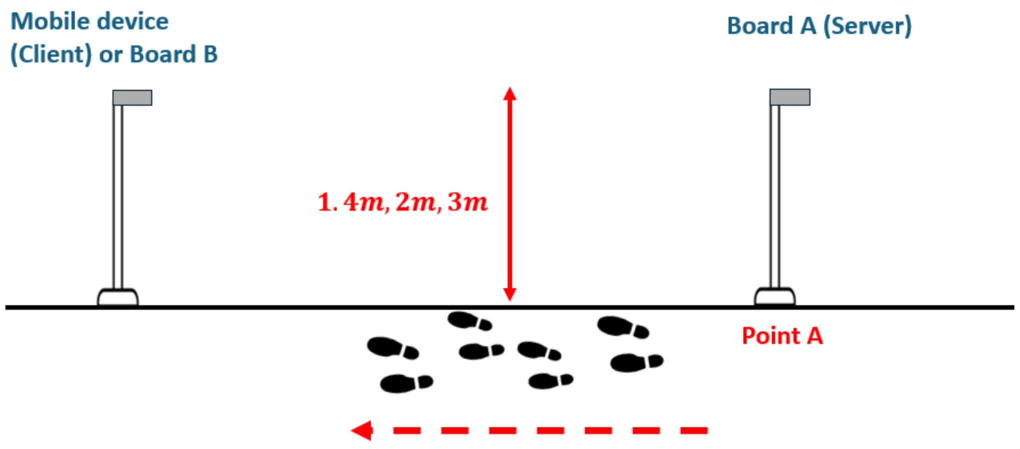


Figure 3.24: Different heights are applied using extension poles.

3.3.1.3 Tested in different transmit (Tx) powers

Furthermore, the transmit power is taken into account as it will be affecting the distance of the signal to be transmitted based on its value. In this case, the value of transmit power such as 0 dBm and 8 dBm are considered. For the boards that are coded based on the long-range application from the software package, the transmit power is initially set to 0 dBm. To increase the transmit power to 8 dBm, the code needs to be modified and programmed into the board using IDE software such as WiSE STUDIO. to meet the settings.

The evaluation board is recognised as a CMSIS-DAP device on Windows once it is powered on through a USB cable as shown in Figure 3.25. BLE long-range application needs to be imported to WiSE_STUDIO before updating the code to modify the transmit power as shown in Figure 3.26. A service and characteristic are created for modification on transmit power value, the characteristic value to be identified and set into transmit power value as shown in Figure 3.27.

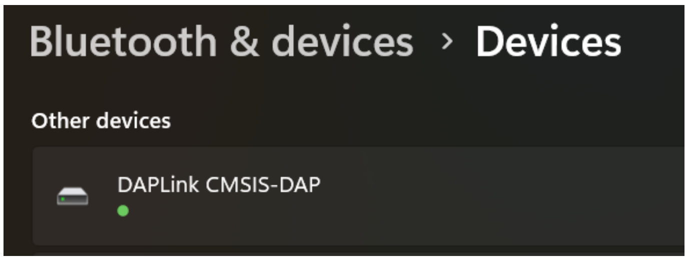


Figure 3.25: CMSIS-DAP is recognised after connecting the board to a laptop.

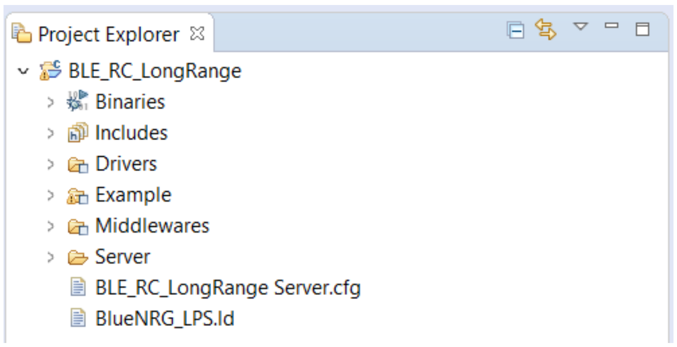


Figure 3.26: Project imported into WiSE-STUDIO.

```
else if (handle == txCharHandle + 1)
{
    if (data_length == 1)
    {
        // Extract the TX power level from the input data
        int8_t new_power_level = (int8_t)att_data[0];

        // Update the OUTPUT_POWER_LEVEL variable
        OUTPUT_POWER_LEVEL = new_power_level;

        // Set the TX power level using your BLE stack's API
        TX_Update((uint8_t)OUTPUT_POWER_LEVEL);

        PRINTF("Updated TX Power Level: %d dBm\n", OUTPUT_POWER_LEVEL);
    }
    else
    {
        // Handle error: Invalid data length
        PRINTF("Error: Invalid data length for TX power level update.\n");
    }
}
```

Figure 3.27: Code fragment to set the value of transmit power based on the characteristic value.

To programme the modified code into the board, WiSE-STUDIO is used to build the project after the modification is made on the code as shown in Figure 3.28, the CMSIS-DAP is confirmed to be selected before debugging as shown in Figure 3.29, then debug or run the simulation to boot the modified code into the board and Figure 3.30. Note that the board requires to be reset before booting the code into it. To do the reset action, the PUSH1 button needs to be pressed and held, the RESET button is then pressed as demonstrated as shown in Figure 3.31. By doing so, it activates the internal UART bootloader, and the simulation can be performed and the code can be sent to the board properly.

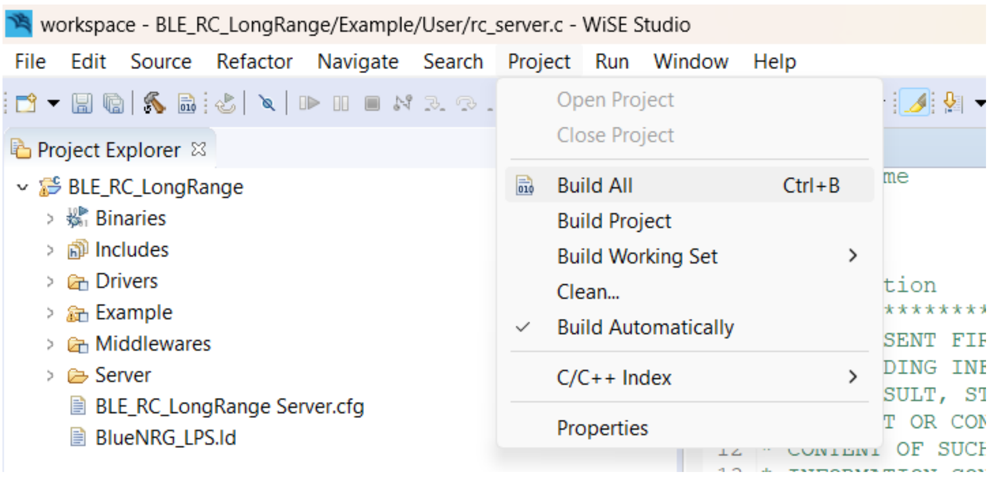
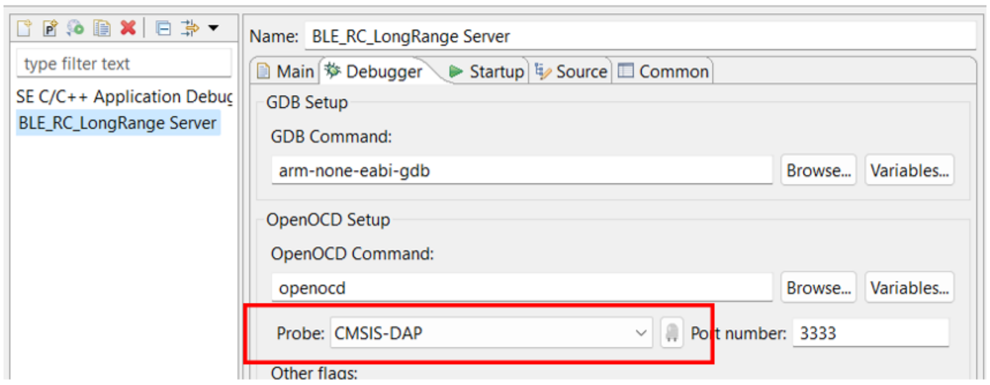


Figure 3.28: Build options are provided in WiSE-Studio after editing the code.



Name: BLE_RC_LongRange Server

GDB Setup

GDB Command: arm-none-eabi-gdb

OpenOCD Setup

OpenOCD Command: openocd

Probe: CMSIS-DAP

Port number: 3333

Other flags:

Figure 3.29: CMSIS-DAP is selected under Run/ Debug Configurations.

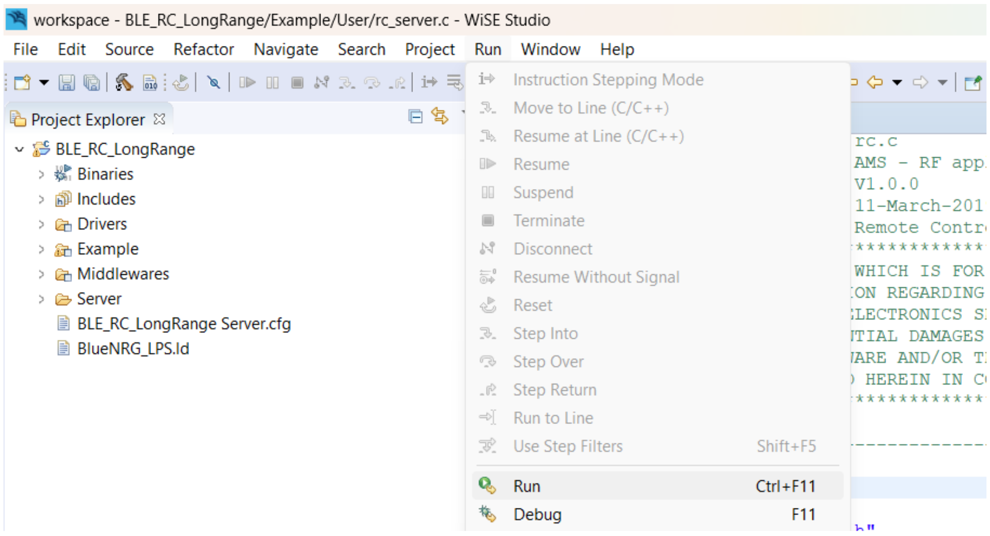

workspace - BLE_RC_LongRange/Example/User/rc_server.c - WiSE Studio

Figure 3.30: Run or Debug options are available to boot the code into the board.

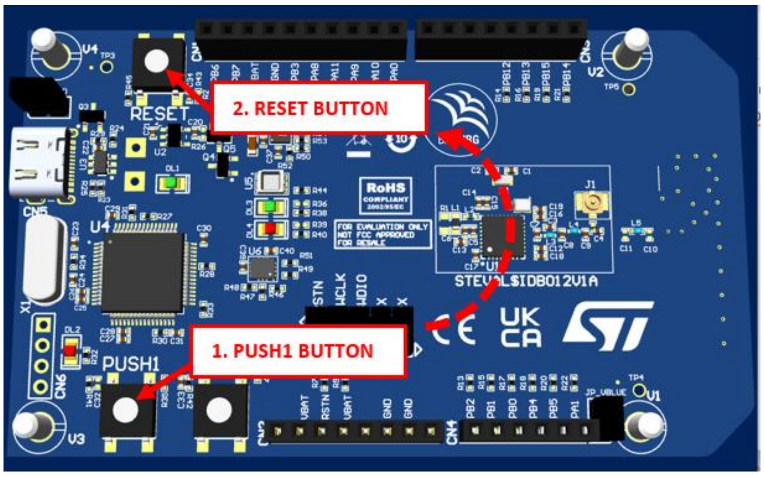
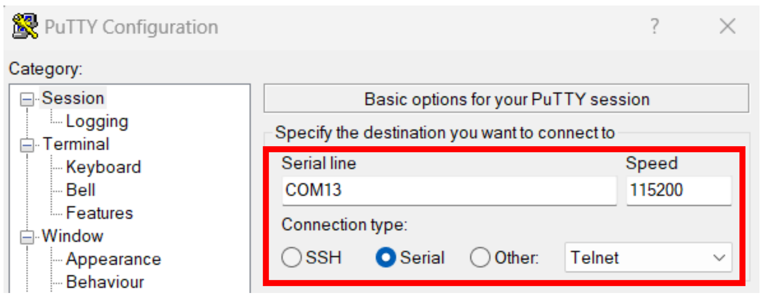


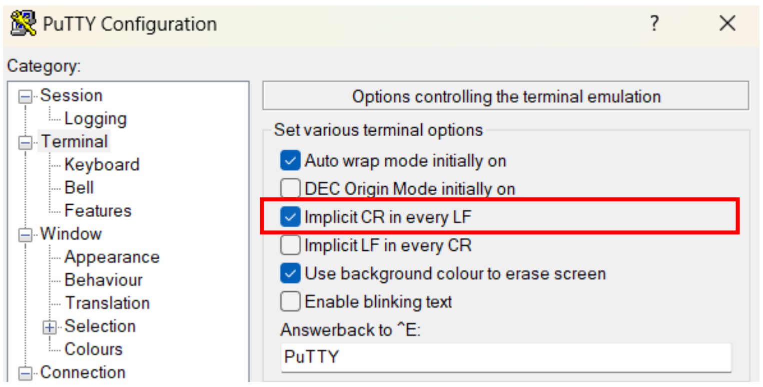
Figure 3.31: PUSH1 button is pressed and held, RESET button is then pressed.

By connecting the board to the laptop through a USB cable, the application of Putty is then implemented to open the serial terminal to read the information from the board. With the settings as shown in Figure 3.32 and Figure 3.33, the information printed from the board can be read. Once both boards are connected, the latest transmit power can be read in the serial terminal by reading the characteristic value as shown in Figure 3.34, the serial terminal also shows the changes in the transmit power for debugging and validation of the transmit power after updating the transmit power as shown in Figure 3.35.



Serial line: COM13
Speed: 115200
Connection type: SSH [✓ Serial] Other: Telnet

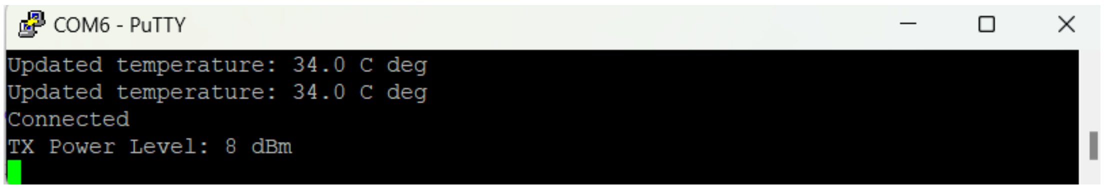
Figure 3.32: Settings of Serial line (COM) and Speed.



- [✓] Auto wrap mode initially on
- [ ] DEC Origin Mode initially on
- [✓] Implicit CR in every LF
- [ ] Implicit LF in every CR
- [✓] Use background colour to erase screen
- [ ] Enable blinking text

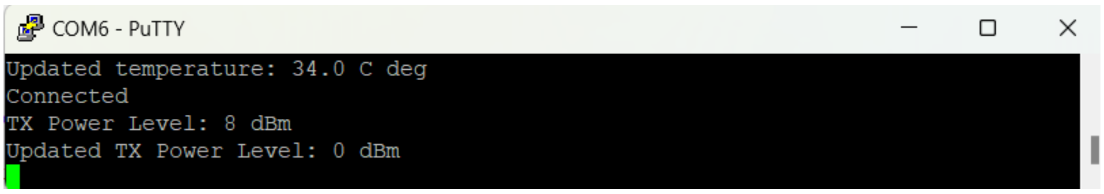
Answerback to ^E: PuTTY

Figure 3.33: Enabling [Implicit CR in every LF] option to increase readability.



```
Updated temperature: 34.0 C deg
Updated temperature: 34.0 C deg
Connected
TX Power Level: 8 dBm
```

Figure 3.34: Transmit power is observed if the characteristic value is read.



```
Updated temperature: 34.0 C deg
Connected
TX Power Level: 8 dBm
Updated TX Power Level: 0 dBm
```

Figure 3.35: Transmit power is updated if a new characteristic value is written.

Once the modified code is successfully booted into the boards, the transmit power can be easily updated based on reading or writing the relevant characteristic value, then the distance measurement under the specific transmit power (0 dBm or 8 dBm) can be conducted.

3.4 Distance and Direction Estimation

In this section, the distance and direction estimation is focused to meet the purpose of asset tracking. The Coded PHY is the physical layer to be focused on in this case. The RSSI-based estimation technique which is the trilateration method, performs the distance and position estimation based on RSSI values among beacons. Other than that, the distance and direction estimation is also performed based on the developed Android application using Android Studio. This will be the combination of RSSI reading and sensors' data to determine distance and direction, which will collect the sensors' value from the mobile device and predict the direction of the mobile device relative to the evaluation board.

Both methods perform distance estimation based on RSSI readings; the log-distance path loss model is implemented to estimate the distance between the devices. Based on Equation (2.3), the distance estimation formula requires inputs such as RSSI readings and path loss exponent (n). These are the parameters that are highly affected by obstacles like walls, multipath interference, and signal reflection. In other words, fluctuations can easily occur in these environmental impact factors (RSSI reading and path loss exponent), leading to inaccurate distance estimation and wrong information for users. Hence, the optimal path-loss exponent and stable RSSI readings are essential to obtain an accurate distance estimation.

The optimal path-loss exponent (n) is obtained by performing linear regression after collecting the RSSI values at different distances and the reference RSSI value. To perform the linear regression, Equation (2.1) can be rearranged into Equation (3.1), then the linear equation can be defined as Equation (3.2).

$RSSI = RSS_{d0} - 10n \times \log_{10}\left(\frac{d}{d0}\right) + X_\sigma$ from Equation (2.1)

$$\Downarrow$$

$$RSSI - RSSI_{d0} = -10n \times \log_{10}\left(\frac{d}{d0}\right) + X_\sigma \tag{3.1}$$

$$RSSI' = -10n \times D + X_\sigma \tag{3.2}$$

where

$RSSI' = RSSI - RSSI_{d0}, D = \log_{10} d$

$Assuming\ X_\sigma = 0, Setting\ d0 = 1m, RSSI_{d0} = RSSI_{1m}$

Based on Equation (3.2), the slope (b) of the line can be written as $-10 \times n$, which the path loss exponent can be recalculated based on the slope value obtained based on RSSI and distance values. In this case, the RSSI values at 1 m, 2 m, 5 m, and 10 m are recorded, and the unknowns such as $RSSI'$ and D are recorded. After that, the mean of D and the mean of $RSSI'$ are calculated as shown in Equation (3.3) and Equation (3.4).

$$\overline{D} = \frac{1}{n}\sum_{i=1}^{n} D_i \tag{3.3}$$

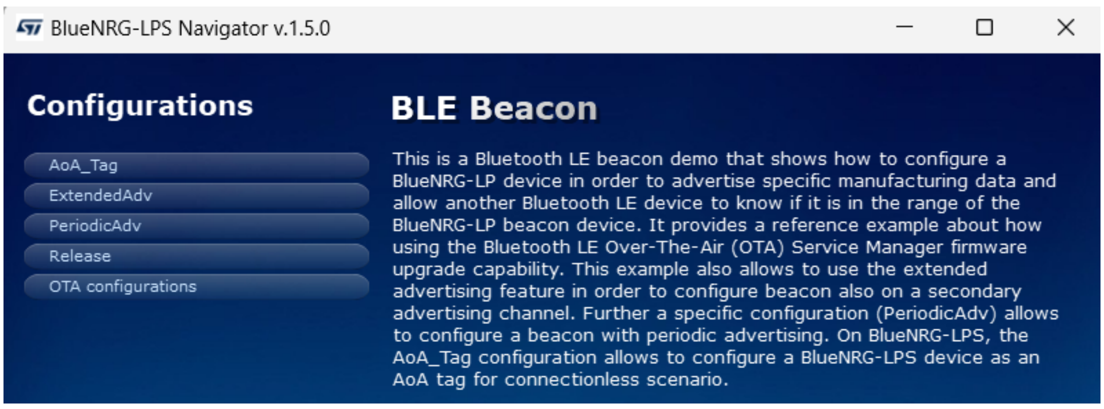
$$\overline{RSSI'} = \frac{1}{n}\sum_{i=1}^{n} RSSI'_i \tag{3.4}$$

The slope (b) in linear regression can be calculated based on Equation (3.5). By solving Equation (3.5) using these mean values, the slope (b) is obtained and used to get the path loss exponent by dividing the value by ten. The path loss exponent will be suitable for the particular spaces and is then used in Equation (2.3) to estimate the distance between two devices.

$$b = \frac{\sum_{i=1}^{n}(D_i - \overline{D}) \times (RSSI' - \overline{RSSI'})}{\sum_{i=1}^{n}(D_i - \overline{D})^2} \tag{3.5}$$

3.4.1 RSSI based only -Trilateration

This method is implemented using the Samsung Tab S7 and one evaluation board, the RSSI readings are collected using nRF Connect for Mobile from the Samsung Tab S7. In this case, multiple boards are needed to send the signal from different positions to the mobile device for trilateration. Hence, the implementation of beacons to the board is important as it allows the devices to communicate without needing a connection, making the collection of RSSI values from different devices available. The software package (STSW-BNRGLP-DK) provides an extended advertising example for users to code the boards as beacons using Coded PHY as shown in Figure 3.36. By uploading the code into the board, the board will be iBeacon and advertise the signal to surrounding Bluetooth devices. The WiSE-STUDIO is also needed in this case to modify, build, and run the code into the boards, allowing multiple boards to perform as the beacons simultaneously.



BLE Beacon

This is a Bluetooth LE beacon demo that shows how to configure a BlueNRG-LP device in order to advertise specific manufacturing data and allow another Bluetooth LE device to know if it is in the range of the BlueNRG-LP beacon device. It provides a reference example about how using the Bluetooth LE Over-The-Air (OTA) Service Manager firmware upgrade capability. This example also allows to use the extended advertising feature in order to configure beacon also on a secondary advertising channel. Further a specific configuration (PeriodicAdv) allows to configure a beacon with periodic advertising. On BlueNRG-LPS, the AoA_Tag configuration allows to configure a BlueNRG-LPS device as an AoA tag for connectionless scenario.

Figure 3.36: Extended Advertising Beacon example is available.

Once the beacon is booted with the relevant code to perform the beacon application, it can be found in the nRF Connect for Mobile during the scanning stage from the Samsung Tab S7 as shown in Figure 3.37. The option for connection is not available in this case, the RSSI graph is generated based on the beacon in the application for users to record the RSSI readings as shown in Figure 3.38. In this case, there are 100 RSSI values obtained at different distances (1 m, 2 m, 5 m, and 10 m), and the average value at each distance is counted for generating the optimum path loss exponent.

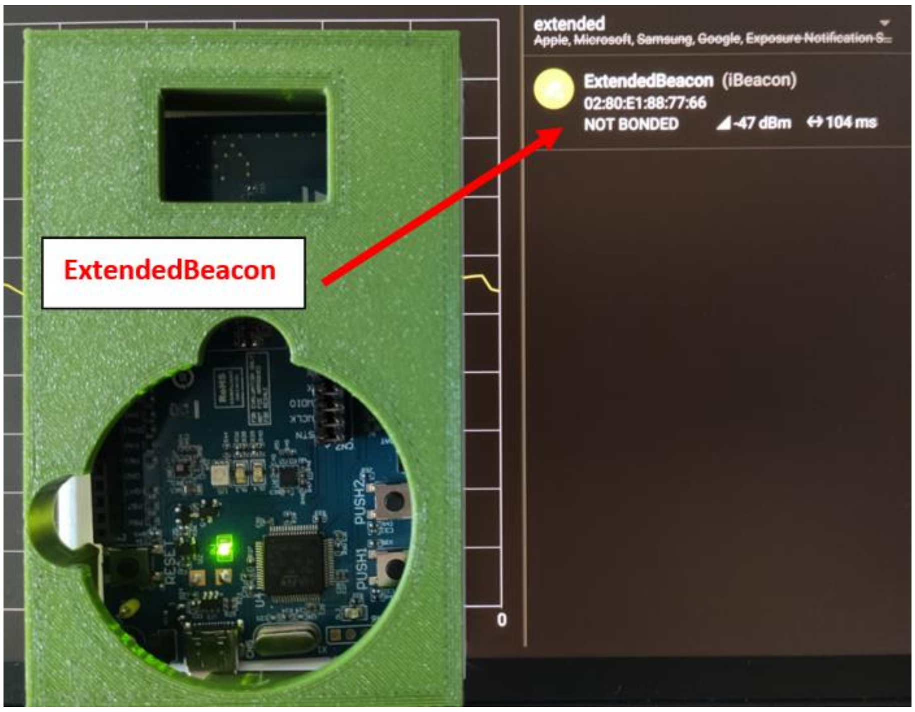


Figure 3.37: Beacon is detected in the application.

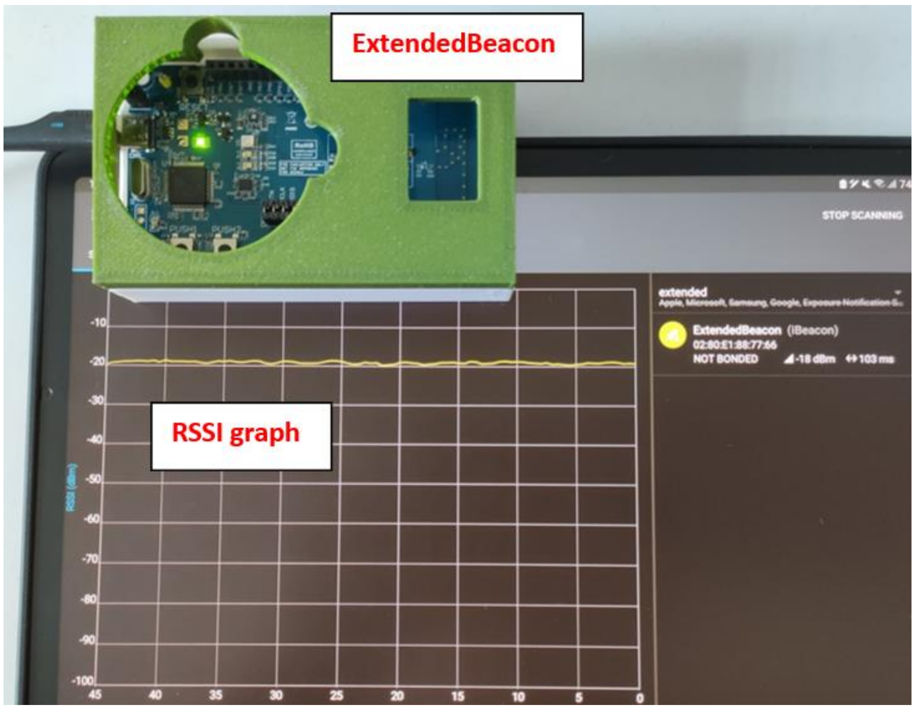


Figure 3.38: RSSI graph is available to read the RSSI of beacons.

The distance between the mobile device and the beacon can be obtained based on Equation (2.3) after calculating the path loss exponent. To perform the Trilateration algorithm into the calculation, Equation (3.6) will be considered. By understanding the position of three beacons, and their distance relative to the mobile device based on RSSI values and Equation (2.3), the position of the mobile device can be predicted. In this case, the test is performed in an outdoor environment to determine the advantage of Coded PHY using the trilateration method. The position of three beacons is found and pointed using Google Maps. Their position is fixed, the moving object or the asset is the mobile device itself.

$$(x - x_i)^2 + (y - y_i)^2 = d_i^2 \tag{3.6}$$

Equation (3.6) is modified to Equation (3.7), Equation (3.8), and Equation (3.9) to suit three beacons application in the implementation of trilateration. These equations are further expanded and simplified as shown in Equation (3.10), Equation (3.11), and Equation (3.12). Based on Equation (3.11) and (3.12), the position (x, y) of the asset or mobile device can be calculated using a method such as substitution. However, the positions of the beacons are obtained using Google Maps as mentioned earlier, which are in the format of geographical coordinates (latitude and longitude). They cannot be directly pasted into the calculation to get the expected results. Hence, the conversion for latitude and longitude to metres is crucial in this case by considering the approximate constants in Equation (3.13) and Equation (3.14).

$$(x-x_1)^2+(y-y_1)^2=d_1^2 \tag{3.7}$$

$$(x-x_2)^2+(y-y_2)^2=d_2^2 \tag{3.8}$$

$$(x-x_3)^2+(y-y_3)^2=d_3^2 \tag{3.9}$$

$$(x-x_1)^2+(y-y_1)^2-(x-x_2)^2-(y-y_2)^2=d_1^2-d_2^2 \tag{3.10}$$

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

$$\therefore \boldsymbol{A_1x+B_1y=C_1} \text{ ----------------} \tag{3.11}$$

where

$$A_1=x_2-x_1, B_1=y_2-y_1, C_1=\frac{d_1^2-d_2^2+x_2^2-x_1^2+y_2^2-y_1^2}{2}$$

$$(x_3-x_1)x+(y_3-y_1)y=\frac{d_1^2-d_3^2+x_3^2-x_1^2+y_3^2-y_1^2}{2}$$

$$\therefore \boldsymbol{A_2x+B_2y=C_2} \text{ ----------------} \tag{3.12}$$

where

$$A_2=(x_3-x_1), B_2=(y_3-y_1), C_2=\frac{d_1^2-d_3^2+x_3^2-x_1^2+y_3^2-y_1^2}{2}$$

$$1\ degree\ of\ latitude \approx 111{,}320m \tag{3.13}$$

$$1\ degree\ of\ longtitude \approx 111{,}320 \times \cos\left(average\ latitude \times \frac{\pi}{180}\right) \tag{3.14}$$

where

$$average\ latitude=\frac{x_1+x_2+x_3}{3}$$

After converting the latitude and longitude data of the beacons' position to metres, the trilateration algorithm is used to calculate the position (x,y) of the object successfully. Once the position of the object is obtained, the values of x and y in metres need to be converted back into latitude and longitude values using the conversion values for searching the real positions in Google Maps. To locate the points in Google Maps, the latitude and longitude data in Decimal Degrees (DD) still need to be converted to Degrees Minutes Seconds (DMS) using Equation (3.15), Equation (3.16) and Equation (3.17). A comparison between the real position and the position estimate for the mobile device is made to determine the accuracy of trilateration using Coded PHY in an outdoor environment.

$$Degrees, D = integer\ of\ DD \tag{3.15}$$

Where

$$DD = Decimal\ Degrees$$

$$Minutes, M = integer\ of\ (|DD - D|) \times 60 \tag{3.16}$$

$$Seconds, S = \left(|DD| - |D| - \frac{M}{60}\right) \times 3600 \tag{3.17}$$

3.4.2 RSSI based and Sensors

This method is carried out using the Mi 9T Pro and one evaluation board, the RSSI readings are collected using a developed Android application. By implementing the filters to reduce the noise in obtaining RSSI readings, the RSSI fluctuation can be avoided. There are hundreds of EMA-filtered RSSI values at different distances (1 m, 2 m, 5 m, and 10 m), the average of the RSSI values are recorded to be the RSSI value for each distance and used to obtain a suitable path loss exponent for the area. It is done by placing the board away from the mobile device at a specific distance, and then run the Android application to record the RSSI values. Once the suitable path loss exponent is collected, it will be substituted into Equation (2.3) to estimate the distance.

To further improve the distance estimation and obtain stable distance readings for users, the mean of ten distance values is taken to ensure the variation of distance values between two devices under the same distance is small. The flowchart is generated to illustrate how to obtain the distance as shown in Figure 3.39.

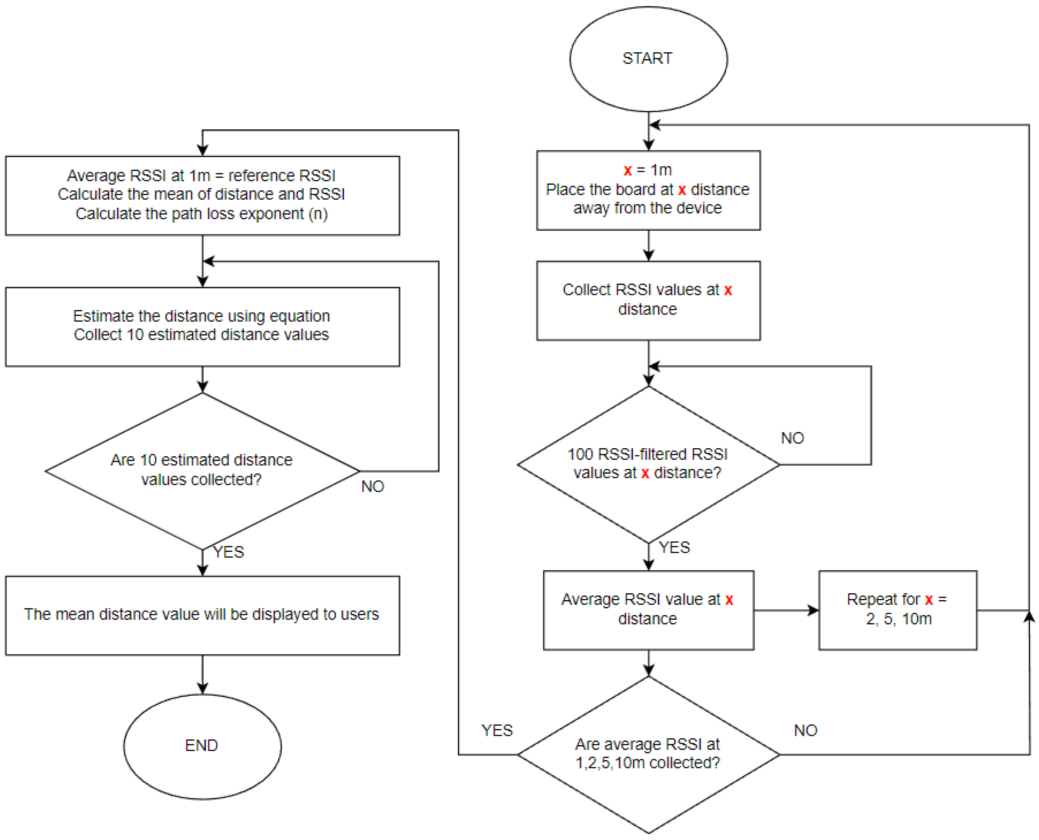

START
x = 1m
Place the board at x distance away from the device
Collect RSSI values at x distance
100 RSSI-filtered RSSI values at x distance?
NO
YES
Average RSSI value at x distance
Repeat for x = 2, 5, 10m
Are average RSSI at 1,2,5,10m collected?
NO
YES
Average RSSI at 1m = reference RSSI
Calculate the mean of distance and RSSI
Calculate the path loss exponent (n)
Estimate the distance using equation
Collect 10 estimated distance values
Are 10 estimated distance values collected?
NO
YES
The mean distance value will be displayed to users
END

Figure 3.39: Flowchart illustrates the process of estimating the distance.

To obtain the direction of the mobile device relative to the board, the phone's sensors such as accelerometer and magnetometer are needed. The mobile device will be required to rotate in eight angles on the horizontal level, then sensors are used to detect the current angle and the placement of the mobile device, then collect twenty RSSI values and the mean of RSSI values at each angle will be compared. The angle that records the highest mean RSSI value is predicted to be the direction of the phone relative to the board. A flowchart is generated to capture the operation of estimating the direction as shown in Figure 3.40.

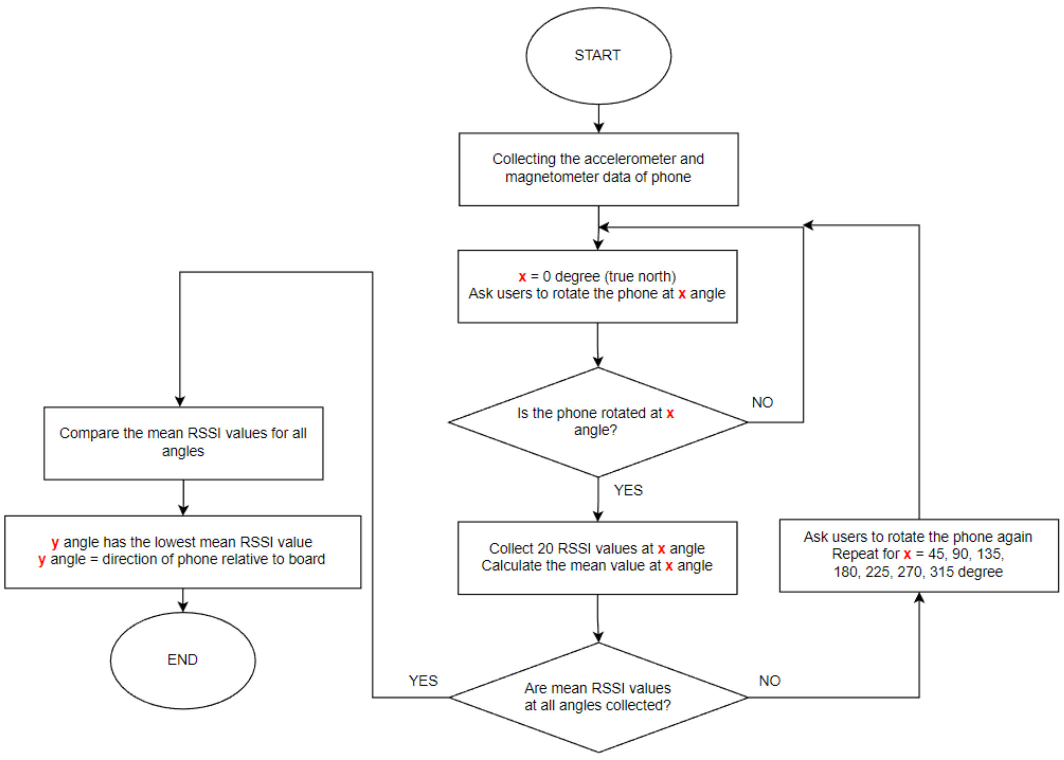
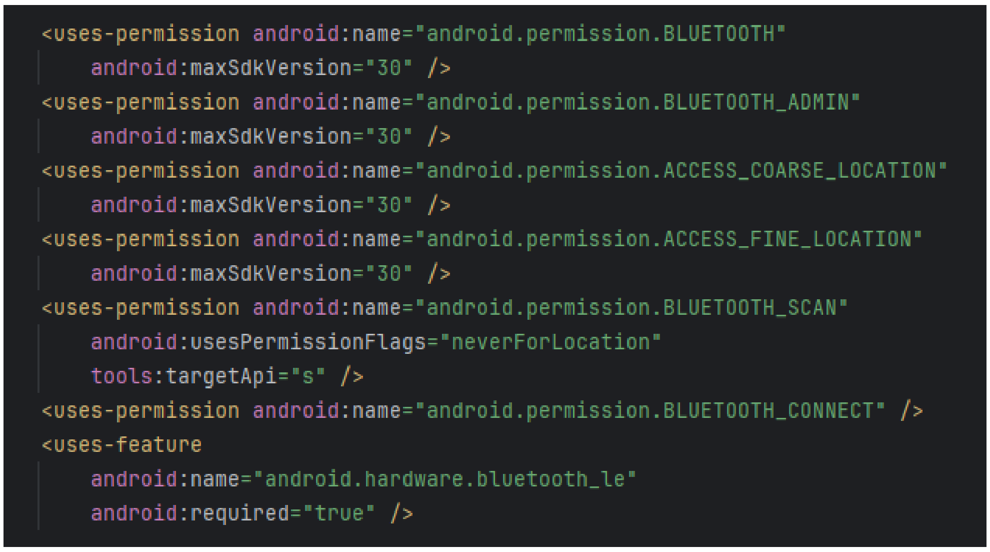

START
Collecting the accelerometer and magnetometer data of phone
x = 0 degree (true north)
Ask users to rotate the phone at x angle
Is the phone rotated at x angle?
NO
YES
Collect 20 RSSI values at x angle
Calculate the mean value at x angle
Are mean RSSI values at all angles collected?
NO
Ask users to rotate the phone again
Repeat for x = 45, 90, 135, 180, 225, 270, 315 degree
YES
Compare the mean RSSI values for all angles
y angle has the lowest mean RSSI value
y angle = direction of phone relative to board
END

Figure 3.40: Flowchart illustrates the process of estimating the direction.

3.4.2.1 Android Application Development

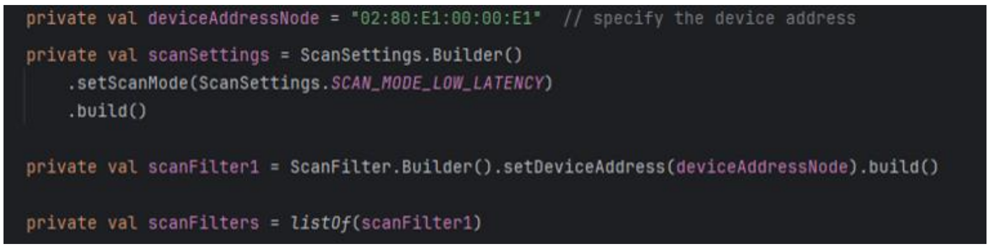
The Android application is designed using Android Studio, providing user interfaces for users to access the features of scanning, connecting, and controlling the ble devices. The permissions are critical in developing an Android application to perform the Bluetooth features as shown in Figure 3.41. In this case, the BLE-related permission is inserted, ensuring the BLE parameters can be applied in the code.



```xml
<uses-permission android:name="android.permission.BLUETOOTH"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
    android:usesPermissionFlags="neverForLocation"
    tools:targetApi="s" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-feature
    android:name="android.hardware.bluetooth_le"
    android:required="true" />
```

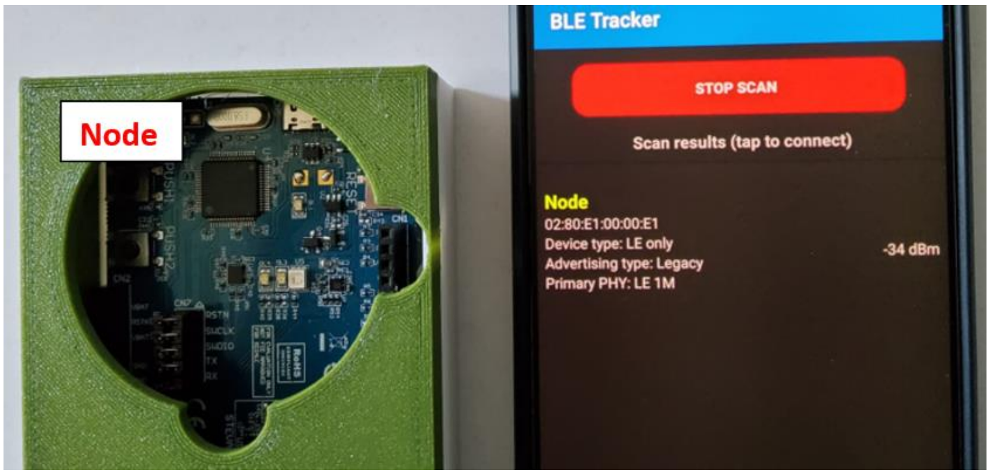
Figure 3.41: Essential permissions to perform Bluetooth features.

The scanning features available in the application, allow users to detect the surrounding ble devices. In this case, the device address of the board is known based on the C code provided in long-range applications from the software package. Alternatively, the device address can be found using nRF Connect for Mobile. Hence, the device address is highlighted in the code to ensure the Android application is only scanning for the specific board by filtering the device address using ScanFiters as shown in Figure 3.42. A *scan* button is inserted in the first layout of the Android application for scanning the board, which can be illustrated as shown in Figure 3.43.



```kotlin
private val deviceAddressNode = "02:80:E1:00:00:E1"  // specify the device address
private val scanSettings = ScanSettings.Builder()
    .setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY)
    .build()

private val scanFilter1 = ScanFilter.Builder().setDeviceAddress(deviceAddressNode).build()

private val scanFilters = listOf(scanFilter1)
```

Figure 3.42: ScanFilters allows the devices to be filtered based on device address.

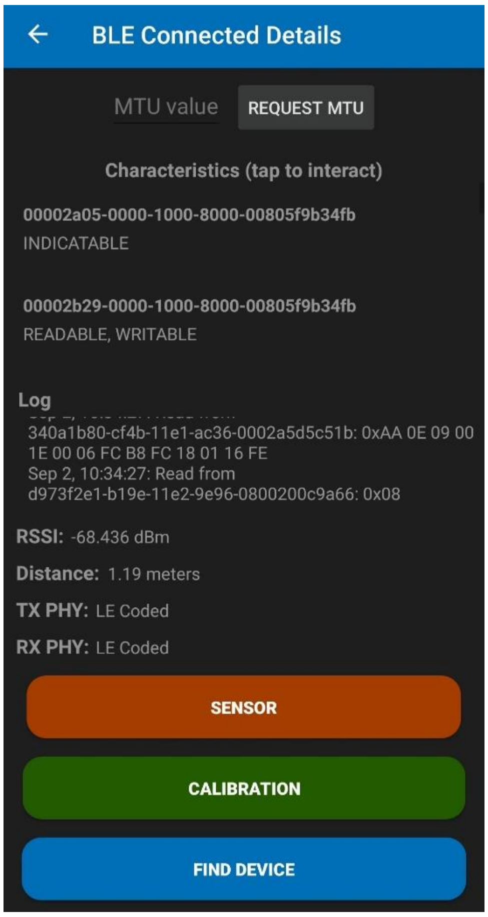


Node
02:80:E1:00:00:E1
Device type: LE only
Advertising type: Legacy
Primary PHY: LE 1M
-34 dBm

Figure 3.43: The board is the only device to be detected.

The details such as device type, advertising type, and RSSI values can be observed once the board is scanned. By clicking on the board from the page, the board will be connected to the mobile device. A new layout will be linked to show users more details, as shown in Figure 3.44. For instance, the services and characteristics of the board are discovered, and the MTU size can be requested after the connection is made. By finding all services and characteristics offered by the board, users are allowed to click on the characteristics to perform the actions that are set in the characteristic. The log section is prepared to display information such as reading, writing, and notifying events in the system. For example, the update of the value in a specific characteristic will be displayed in the log after the writing action is done, making users easily debug the error or observe the changes in the value.

Furthermore, the RSSI values and the distance estimation can also be observed and recorded in the application. Moreover, the physical layer of the board can also be reviewed, in this case, the Android application is coded to set the physical layer of the board to be Coded PHY once the connection is made by using setPreferredPhy as shown in Figure 3.45. The LED1 on the board is also powered on after the connection is made, indicating that the physical layer is successfully updated to Coded PHY as shown in Figure 3.46.



Characteristics (tap to interact)

00002a05-0000-1000-8000-00805f9b34fb
INDICATABLE

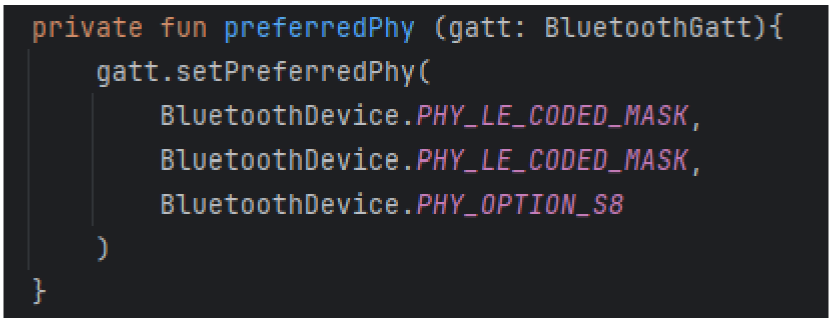
00002b29-0000-1000-8000-00805f9b34fb
READABLE, WRITABLE

Log

```
340a1b80-cf4b-11e1-ac36-0002a5d5c51b: 0xAA 0E 09 00
1E 00 06 FC B8 FC 18 01 16 FE
Sep 2, 10:34:27: Read from
d973f2e1-b19e-11e2-9e96-0800200c9a66: 0x08
```

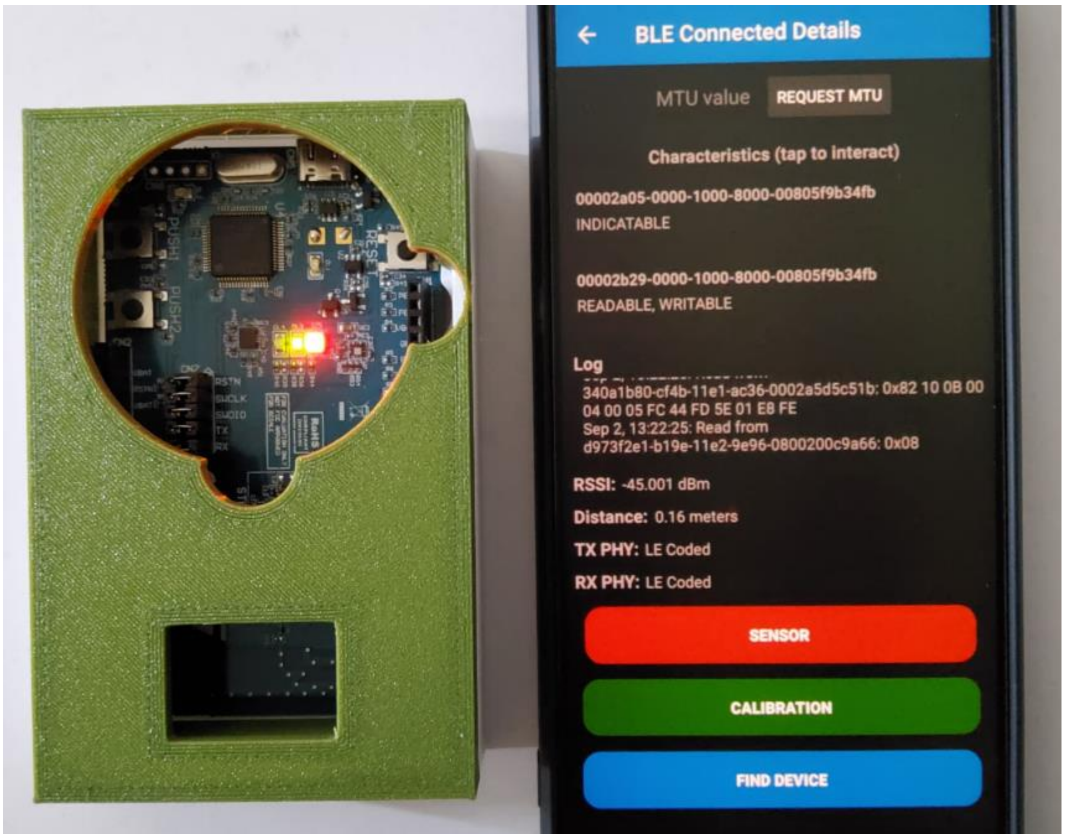
RSSI: -68.436 dBm
Distance: 1.19 meters
TX PHY: LE Coded
RX PHY: LE Coded

Figure 3.44: User interface linked after the connection is made.



```kotlin
private fun preferredPhy (gatt: BluetoothGatt){
    gatt.setPreferredPhy(
        BluetoothDevice.PHY_LE_CODED_MASK,
        BluetoothDevice.PHY_LE_CODED_MASK,
        BluetoothDevice.PHY_OPTION_S8
    )
}
```

Figure 3.45: setPreferredPhy allows the physical layer of the device to be modified.



Characteristics (tap to interact)

00002a05-0000-1000-8000-00805f9b34fb
INDICATABLE

00002b29-0000-1000-8000-00805f9b34fb
READABLE, WRITABLE

Log

```
340a1b80-cf4b-11e1-ac36-0002a5d5c51b: 0x82 10 0B 00
04 00 05 FC 44 FD 5E 01 E8 FE
Sep 2, 13:22:25: Read from
d973f2e1-b19e-11e2-9e96-0800200c9a66: 0x08
```

RSSI: -45.001 dBm

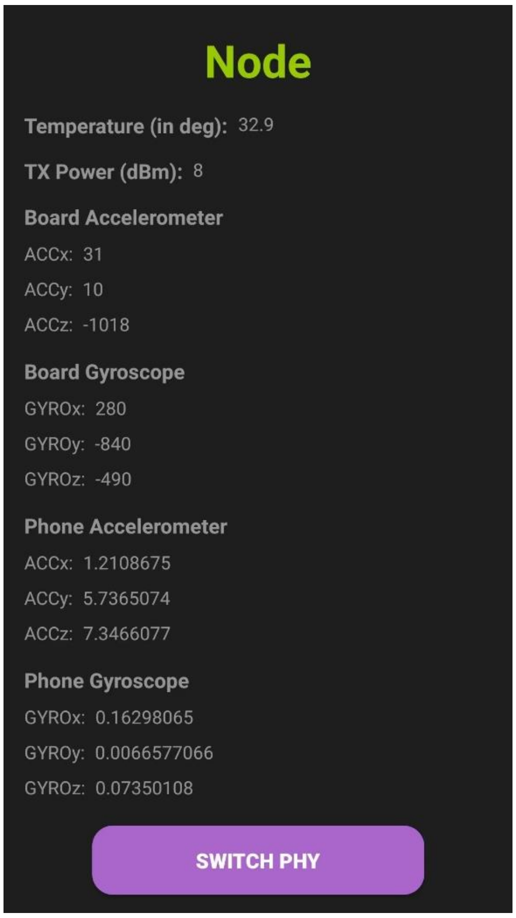
Distance: 0.16 meters

TX PHY: LE Coded

RX PHY: LE Coded

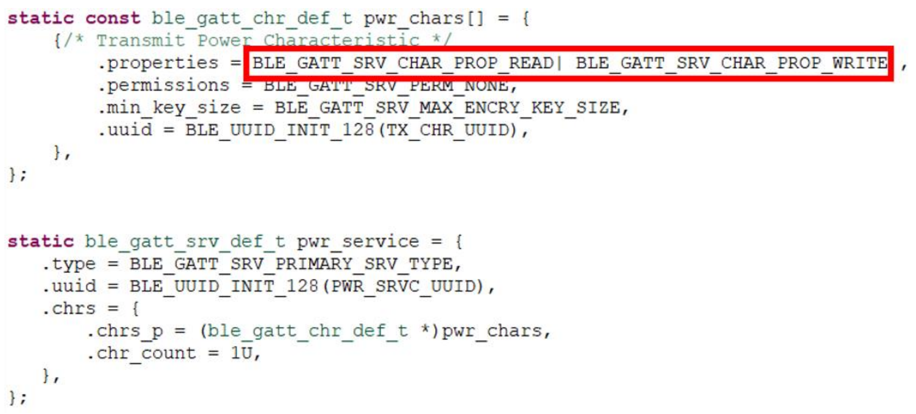
Figure 3.46: LED1 of the board is on once connecting to the mobile device.

Three interactive buttons are available for users after the connection between the mobile device and the board is established which are *sensor*, *calibration*, and *find device*. Users are allowed to select one of the buttons to explore more details regarding the board or the phone. A new layout will be presented to users by clicking the *sensor* button, as shown in Figure 3.47. This page gathers the sensors' data of the connected board as well as the phone sensors' data. In this study, the evaluation board (STEVAL-IDB012V1) is used to be the BLE device to communicate with mobile devices, it is equipped with multiple sensors such as the accelerometer, gyroscope, and temperature sensors. There are several services and characteristics created for these sensors, to store the sensors' data. By setting the operation to trigger and read these characteristics after clicking the *sensor* button, the real-time sensors' data can always be updated for users. The method of presenting the transmit power of the board is also done by generating a specific characteristic and reading the value stored in the characteristic. The characteristic is created to allow users to perform the read and write action as shown in Figure 3.48. Hence, users can easily set the transmit power value of the board to control the communication distance or power consumption.



```
Node

Temperature (in deg):  32.9

TX Power (dBm):  8

Board Accelerometer
ACCx:  31
ACCy:  10
ACCz:  -1018

Board Gyroscope
GYROx:  280
GYROy:  -840
GYROz:  -490

Phone Accelerometer
ACCx:  1.2108675
ACCy:  5.7365074
ACCz:  7.3466077

Phone Gyroscope
GYROx:  0.16298065
GYROy:  0.0066577066
GYROz:  0.07350108
```

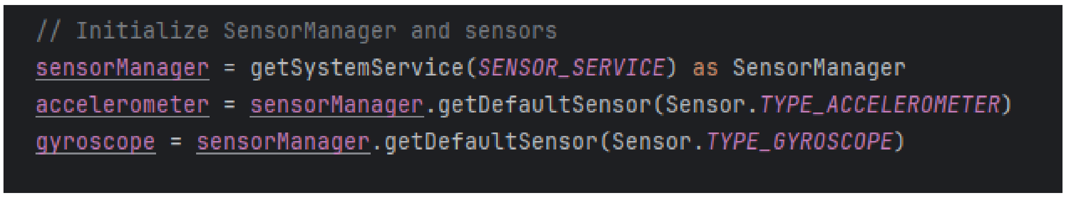
Figure 3.47: User interface linked after pressing the *sensor* button.



```
static const ble_gatt_chr_def_t pwr_chars[] = {
    {/* Transmit Power Characteristic */
        .properties = BLE_GATT_SRV_CHAR_PROP_READ| BLE_GATT_SRV_CHAR_PROP_WRITE ,
        .permissions = BLE_GATT_SRV_PERM_NONE,
        .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
        .uuid = BLE_UUID_INIT_128(TX_CHR_UUID),
    },
};


static ble_gatt_srv_def_t pwr_service = {
   .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
   .uuid = BLE_UUID_INIT_128(PWR_SRVC_UUID),
   .chrs = {
       .chrs_p = (ble_gatt_chr_def_t *)pwr_chars,
       .chr_count = 1U,
   },
};
```

Figure 3.48: Transmit power characteristic allows users to read and write.

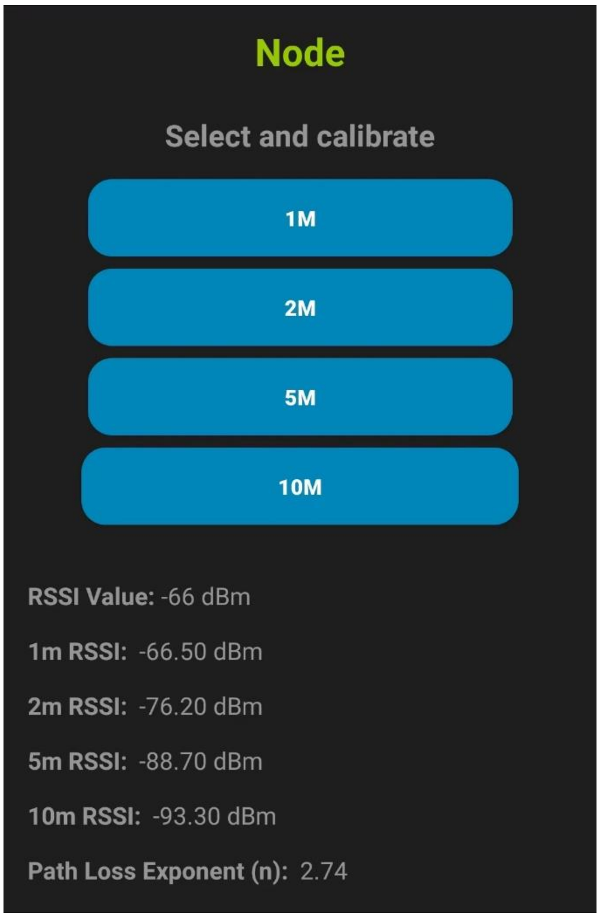
The phone sensors' data are obtained using getSystemService as shown in Figure 3.49, showing the orientation and the movement of the phone based on the accelerometer and gyroscope inputs. In addition, a *switch PHY* button at the bottom of the screen allows the physical layer to switch between 1M PHY and Coded PHY by toggling the button. For example, the board is initially set at 1M PHY, and updated to Coded PHY once the application is run and connected to it, the button allows the physical layer to be switched remotely without pressing the PUSH1 button of the board physically. Overall, this page is developed to collect the information related to the board and the phone, and a simple control button on the physical layer, providing convenience for users to perform debugging and developing future project



```
// Initialize SensorManager and sensors
sensorManager = getSystemService(SENSOR_SERVICE) as SensorManager
accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
```

Figure 3.49: getSystemService allows the phones' data to be accessed.

In the case of pressing the *calibration* button, the user interface connected to this button can be illustrated as shown in Figure 3.50. This page has mainly operated for determining the optimum path loss exponent for the system to get better results in distance estimation, which is to carry out the operation explained in the flowchart as shown in Figure 3.39. By considering different suitable path loss exponent for different areas, and users cannot reach the designing software to modify the constants of RSSI at different distances to get the path loss exponent, this page is designed for this scenario. Four buttons indicate different distances, users need to move the board away from the phone at a specific distance, and then press the relevant distance button to perform calibration. The system will start to collect a hundred of RSSI values, and the average value at the specific distance will be updated and displayed to the user.



RSSI Value: -66 dBm

1m RSSI: -66.50 dBm

2m RSSI: -76.20 dBm

5m RSSI: -88.70 dBm

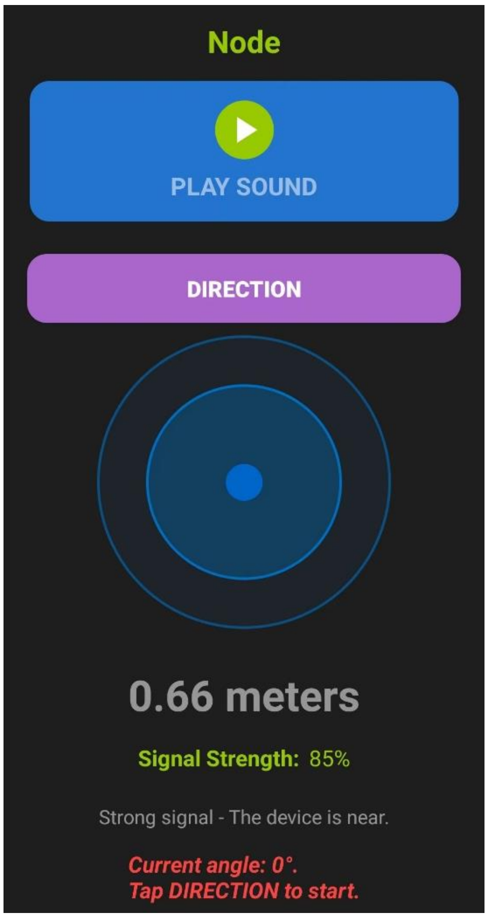
10m RSSI: -93.30 dBm

Path Loss Exponent (n): 2.74

Figure 3.50: User interface linked after pressing the *calibration* button.

Other than displaying the real-time RSSI value, the RSSI value at different distances and the path loss exponent value are displayed to users. The number of RSSI values at different distances is initially set in the software and the path loss exponent calculated based on them is only suitable for a particular space. The results in distance estimation may be not desired if they are tested in other environments without conducting calibration. Hence, users perform the calibration for four distances on this page, a new and optimum path loss exponent is determined and the accuracy of distance estimation for the current space is now improved.

In the case of pressing the *find device* button, the layout displayed to users can be illustrated as shown in Figure 3.51. There are two buttons available on this page which are *play sound* and *direction*. The play sound button is created to control one of the I/O pins on the board. The code for controlling the I/O pin can be illustrated as shown in Figure 3.52. A service and characteristic are defined for it, and the read and write actions are permitted, the output of the I/O pin is based on the characteristic value. In this case, if the characteristic value is inputted by 0x04, then a blinking output is performed by the I/O pin, otherwise, a zero output will be observed if the characteristic value is stored by 0x00. A flowchart is created to demonstrate the operation of controlling the I/O pin as shown in Figure 3.53.



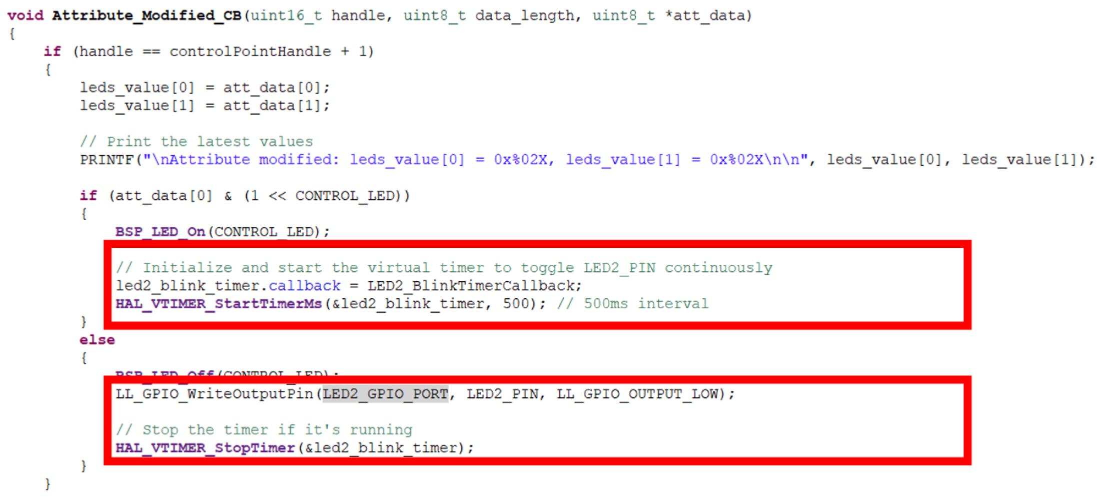
0.66 meters

Signal Strength: 85%

Strong signal - The device is near.

Current angle: 0°.
Tap DIRECTION to start.

Figure 3.51: User interface linked after pressing the *find device* button.



```
void Attribute_Modified_CB(uint16_t handle, uint8_t data_length, uint8_t *att_data)
{
    if (handle == controlPointHandle + 1)
    {
        leds_value[0] = att_data[0];
        leds_value[1] = att_data[1];

        // Print the latest values
        PRINTF("\nAttribute modified: leds_value[0] = 0x%02X, leds_value[1] = 0x%02X\n\n", leds_value[0], leds_value[1]);

        if (att_data[0] & (1 << CONTROL_LED))
        {
            BSP_LED_On(CONTROL_LED);
            // Initialize and start the virtual timer to toggle LED2_PIN continuously
            led2_blink_timer.callback = LED2_BlinkTimerCallback;
            HAL_VTIMER_StartTimerMs(&led2_blink_timer, 500); // 500ms interval
        }
        else
        {
            BSP_LED_Off(CONTROL_LED);
            LL_GPIO_WriteOutputPin(LED2_GPIO_PORT, LED2_PIN, LL_GPIO_OUTPUT_LOW);

            // Stop the timer if it's running
            HAL_VTIMER_StopTimer(&led2_blink_timer);
        }
    }
```

Figure 3.52: Code fragment of controlling the I/O pin.

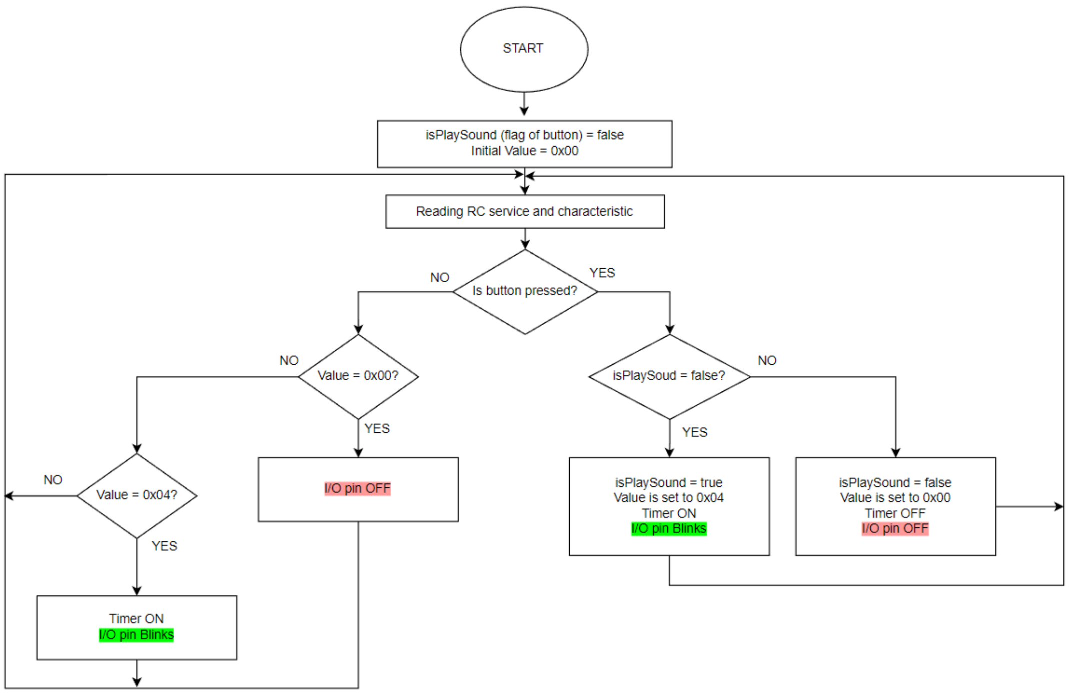


Figure 3.53: Flowchart illustrates the operation of controlling the I/O pin.

A buzzer is connected to that I/O pin as shown in Figure 3.54, the buzzer will be powered on and beeping once the *play sound* button is pressed. The operation of the buzzer can be controlled by toggling the button, allowing users to discover the board or asset instantly when it is near to the user. The *direction* button is used to determine the direction of the mobile device relative to the board. The bottom of the screen displays the information that is related to the direction button such as the current angle that is determined based on the magnetometer of the phone. By pressing this button, the system will display the instruction message at the bottom of the screen to ask users to rotate the phone to different angles on the horizontal plane and collect the mean RSSI value from different angles. The comparison is done to predict the direction of the phone relative to the board, which performs the operation that was explained earlier using the flowchart as shown in Figure 3.40. Figure 3.55 shows how users performs the direction finding on their mobile devices using the Android application. The distances between the mobile device and the board are set at 1 m and 5 m respectively and three orientations of the mobile device will be tested to determine the effect of phone placement on the accuracy of the direction estimation as shown in Figure 3.56.

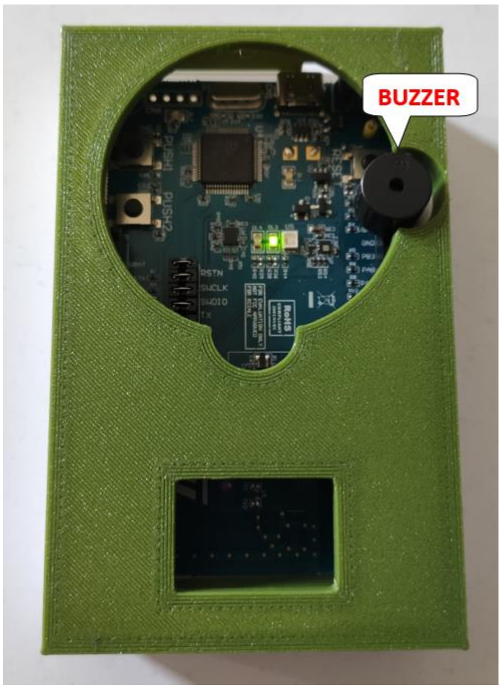
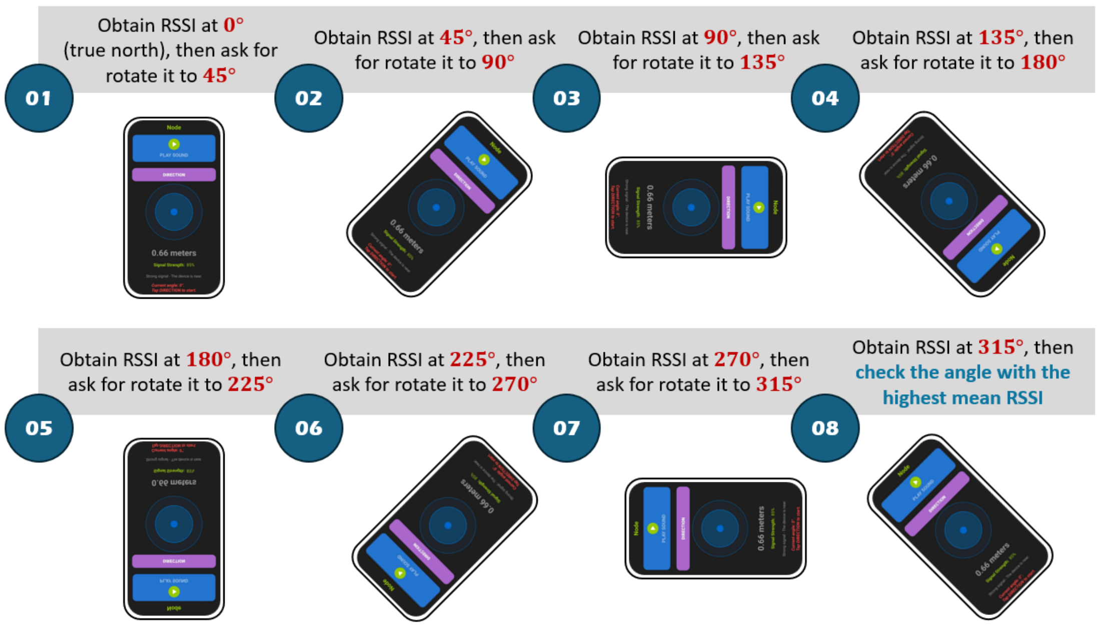


Figure 3.54: A buzzer is attached to the I/O pin.



01
Obtain RSSI at 0° (true north), then ask for rotate it to 45°
02
Obtain RSSI at 45°, then ask for rotate it to 90°
03
Obtain RSSI at 90°, then ask for rotate it to 135°
04
Obtain RSSI at 135°, then ask for rotate it to 180°
05
Obtain RSSI at 180°, then ask for rotate it to 225°
06
Obtain RSSI at 225°, then ask for rotate it to 270°
07
Obtain RSSI at 270°, then ask for rotate it to 315°
08
Obtain RSSI at 315°, then check the angle with the highest mean RSSI
Node
PLAY SOUND
DIRECTION
0.66 meters

Figure 3.55: Operation in rotating the mobile device to estimate the direction.

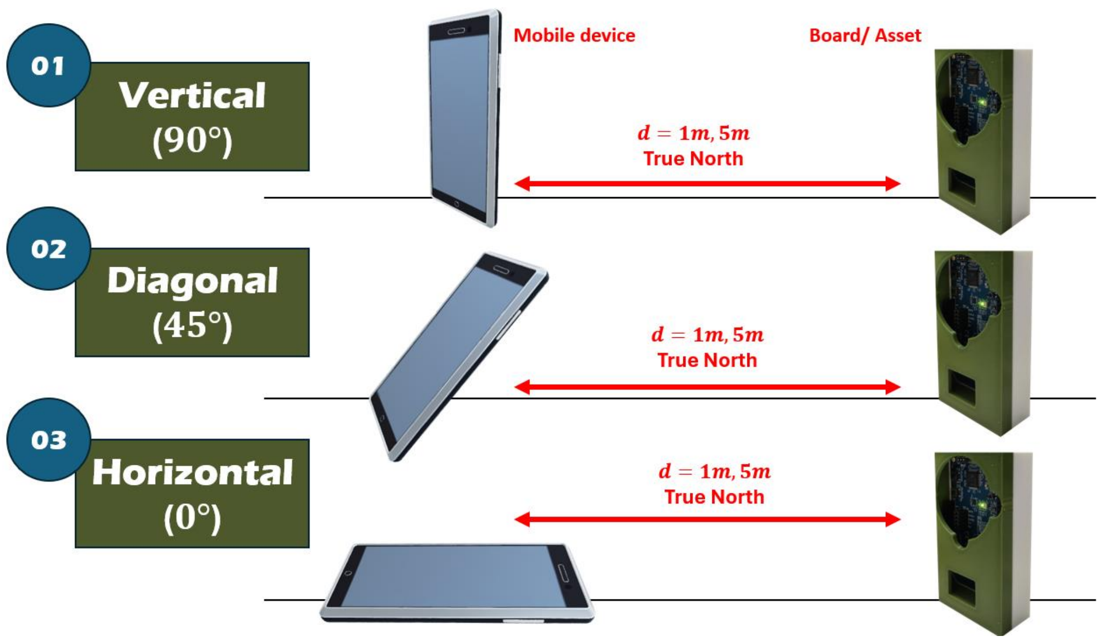
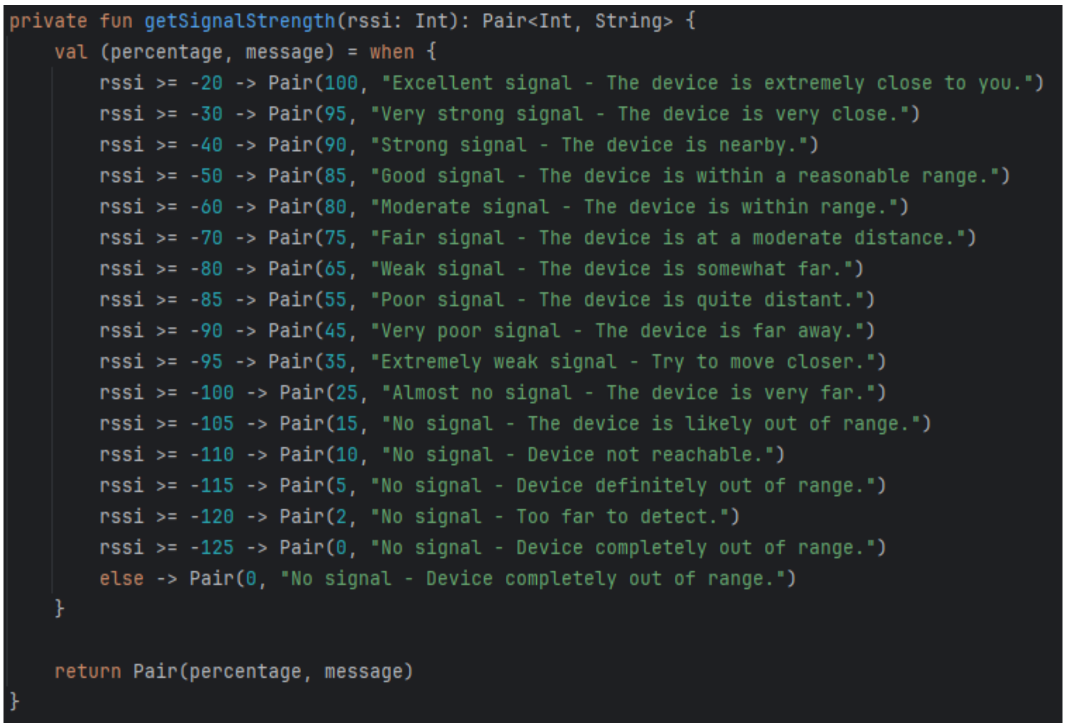


Figure 3.56: Orientation of the mobile device in direction estimation.

The circles at the centre of the screen provide an animation to show users based on the RSSI received. In the case of getting a higher signal strength or RSSI readings, the visibility of the circles is increased. Furthermore, the distance estimation is also performed in this case, and displays the estimated distance message to users; the operation is explained previously in the flowchart as shown in Figure 3.39.

Moreover, the signal strength percentage and message are printed based on the RSSI value. Based on the converted messages from RSSI values, users can easily identify the communication strength between the board and the mobile device. The range of RSSI value is set from -125 dBm to -20 dBm, the converted messages are defined as shown in Figure 3.57. The RSSI value that is smaller than -125 dBm is assumed to be extremely low signal strength, indicating that the mobile device cannot detect the board or asset.



```
private fun getSignalStrength(rssi: Int): Pair<Int, String> {
    val (percentage, message) = when {
        rssi >= -20 -> Pair(100, "Excellent signal - The device is extremely close to you.")
        rssi >= -30 -> Pair(95, "Very strong signal - The device is very close.")
        rssi >= -40 -> Pair(90, "Strong signal - The device is nearby.")
        rssi >= -50 -> Pair(85, "Good signal - The device is within a reasonable range.")
        rssi >= -60 -> Pair(80, "Moderate signal - The device is within range.")
        rssi >= -70 -> Pair(75, "Fair signal - The device is at a moderate distance.")
        rssi >= -80 -> Pair(65, "Weak signal - The device is somewhat far.")
        rssi >= -85 -> Pair(55, "Poor signal - The device is quite distant.")
        rssi >= -90 -> Pair(45, "Very poor signal - The device is far away.")
        rssi >= -95 -> Pair(35, "Extremely weak signal - Try to move closer.")
        rssi >= -100 -> Pair(25, "Almost no signal - The device is very far.")
        rssi >= -105 -> Pair(15, "No signal - The device is likely out of range.")
        rssi >= -110 -> Pair(10, "No signal - Device not reachable.")
        rssi >= -115 -> Pair(5, "No signal - Device definitely out of range.")
        rssi >= -120 -> Pair(2, "No signal - Too far to detect.")
        rssi >= -125 -> Pair(0, "No signal - Device completely out of range.")
        else -> Pair(0, "No signal - Device completely out of range.")
    }

    return Pair(percentage, message)
}
```

Figure 3.57: Percentage and message converted based on RSSI readings.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Comparison between 1M PHY and Coded PHY

The comparison between different physical layers which are 1M PHY and Coded PHY is done in terms of maximum distance. Some factors are considered in conducting the distance measurement such as environment, height, and transmit power (Tx). The devices involved in this test are evaluation boards, and mobile devices such as Mi 9T Pro, and Samsung Tab S7.

Table 4.1: Combination of devices to be tested.

Combination	**Devices**
A	Two evaluation boards
B	Samsung Tab S7 + 1 evaluation board
C	Mi 9T pro + 1 evaluation board

The classification of the combination to be applied in the test can be illustrated as shown in Table 4.1. The following sections show the results collected under indoor and outdoor environments with the variable factors (height and Tx power) that are mentioned above, and the discussion is done to verify the advantage of Coded PHY over 1M PHY.

4.1.1 Maximum distance in an indoor environment

4.1.1.1 Tested in different heights

The variable settings in this test are the physical layers (1M PHY and Coded PHY) and heights (1.4 m, 2.0 m, and 3.0 m). The transmit power of the evaluation board is set to 0 dBm for this test.

Table 4.2: Comparison between PHYs at different heights.

PHY	Height, h (m)	Combination	Distance, d (m)
1M	1.4	A	61.80
		B	65.80
		C	63.00
	2.0	A	121.10
		B	124.80
		C	118.30
	3.0	A	125.60
		B	130.30
		C	123.30
Coded	1.4	A	80.40
		B	82.90
		C	77.40
	2.0	A	179.90
		B	177.60
		C	176.50
	3.0	A	185.40
		B	178.10
		C	184.00

The results of different devices under different physical layers against different heights (h) are recorded as shown in Table 4.2. The comparison of physical layers in distance at different heights is plotted in the bar chart as shown in Figure 4.1.

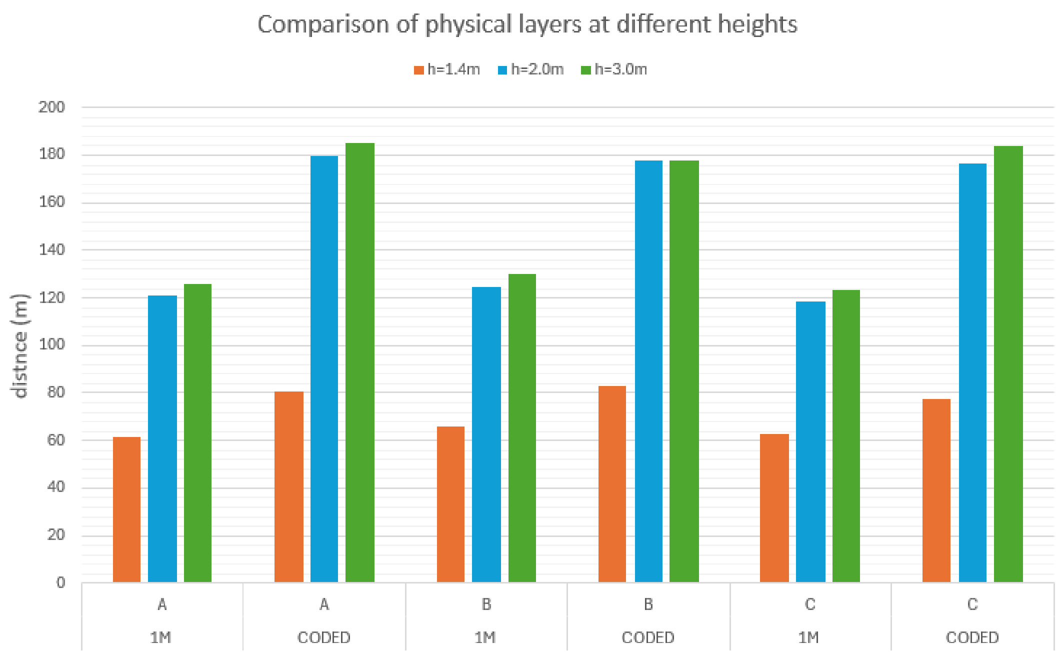


Figure 4.1: Comparison of PHYs at different heights.

Table 4.3: Percent change between PHYs at different heights.

Height, h (m)	Combination	Distance, d (m)				Percent change (%)
		1M		Coded		
1.4	A	61.80	63.53	80.40	80.23	26.29
	B	65.80		82.90		
	C	63.00		77.40		
2.0	A	121.10	121.4	179.90	178.00	46.62
	B	124.80		177.60		
	C	118.30		176.50		
3.0	A	125.60	126.4	185.40	182.50	44.38
	B	130.30		178.10		
	C	123.30		184.00		

The comparison of physical layers among different heights is summarized and the percent change in distance is calculated using Equation (4.1) as shown in Table 4.3.

The average distance of different combinations is obtained to calculate the percent difference between different physical layers.

$$percent\ change\ (\%) = \frac{d_{Coded} - d_{1M}}{d} \times 100 \tag{4.1}$$

Table 4.4: Percent change between transmit powers at the same PHY.

PHY	**Distance, d (m)**			**Percent change (%)**
	$\boldsymbol{h = 1.4m}$	$\boldsymbol{h = 2.0m}$	$\boldsymbol{h = 3.0m}$	
1M	63.53	121.4	▨	91.09
	63.53	▨	126.4	98.96
	▨	121.4	126.4	4.19
Coded	80.23	178.00	▨	121.86
	80.23	▨	182.50	127.47
	▨	178.00	182.50	2.52

Based on the observation in Table 4.3, the percent change shows that the distance under Coded PHY increases up to 26.29%, compared to 1M PHY at 1.4 m. The percent change recorded for 2.0 m and 3.0 m, shows the Coded PHY increase the distance above 40 % compared to 1M PHY. Based on these bar charts and positive percent change, the Coded PHY is found to have a longer distance compared to 1M PHY at the same height. Comparing the percent change at different heights shows that the increase in height can further increase the communication range and the difference in distance between Coded PHY and 1M PHY. Furthermore, based on the observation in Table 4.4, the distance is increased when the height is increased under the same physical layer such as the distance under 1.4 m is doubled when the height is increased to 2.0 m or 3.0 m, concludes the height is one of the important factors that affect the communication range between the devices.

The longer distance is recorded by applying a higher height, which can be explained by the reduction in obstructions such as walls and people that leads to blocking or attenuating the signals. Furthermore, the interference from other devices that operate in a similar frequency range (2.4 GHz) as Bluetooth such as Wi-Fi, and cordless phones can be reduced, so the distance data can be improved.

Other than focusing on the distance difference between the physical layers, the difference distance in different combinations is noticed in this test. It can be found that under the same settings such as physical layer and height, the distance measurement of each combination is observed to be slightly different compared to others. This is because, the factors such as hardware like antenna design and quality, Bluetooth version, firmware, and software optimization of these devices are different from each other, resulting in the signal strength between devices being different compared to others at identical heights and physical layers.

4.1.1.2 Tested in different transmit (Tx) powers

The variable settings in this test are the physical layers (1M PHY and Coded PHY) and transmit powers (0 dBm and 8 dBm) The height setting for the evaluation board is set to 1.4 m for this test.

Table 4.5: Comparison between PHYs at different transmit powers.

PHY	Tx Power (dBm)	Combination	Distance, d (m)
1M	0.0	A	61.80
		B	65.80
		C	63.00
	8.0	A	130.80
		B	144.80
		C	132.60
Coded	0.0	A	80.40
		B	82.90
		C	77.40
	8.0	A	159.80
		B	177.60
		C	162.60

The results of different devices under different physical layers against different transmit power are recorded as shown in Table 4.5. The comparison of physical layers in distance at different transmit powers is plotted in the bar chart as shown in Figure 4.2.

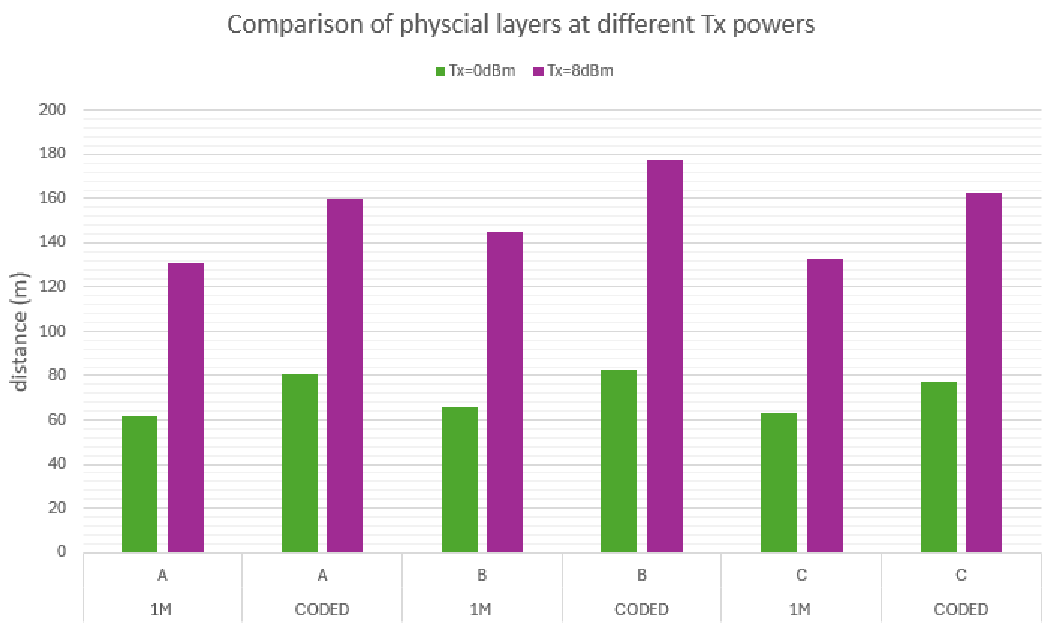


Figure 4.2: Comparison of PHYs at different transmit powers.

Table 4.6: Percent change between PHYs at different transmit powers.

Tx Power (dBm)	**Combination**	**Distance, d (m)**				**Percent change (%)**
		1M		**Coded**		
0.0	A	61.80	63.53	80.40	80.23	26.29
	B	65.80		82.90		
	C	63.00		77.40		
8.0	A	130.80	136.07	159.80	166.67	22.49
	B	144.80		177.60		
	C	132.60		162.60		

The comparison of physical layers among different transmit powers is summarized and the percent change in distance is calculated using as shown in Table 4.6. The average distance of different combinations is obtained to calculate the percentage difference between different physical layers. Based on the observation in Table 4.6, the percent change shows that the distance under Coded PHY increases up to 26.29 %, compared to 1M PHY at 0 dBm transmit power. The percent change recorded for 8 dBm transmit power, shows the Coded PHY increase the distance above 22.49 % compared to 1M PHY.

Table 4.7: Percent change between transmit powers at the same PHY.

PHY	**Combination**	**Distance, d (m)**				**Percent change (%)**
		Tx = 0 dBm		**Tx = 8 dBm**		
1M	A	61.80	63.53	130.80	136.07	114.18
	B	65.80		144.80		
	C	63.00		132.60		
Coded	A	80.40	80.23	159.80	166.67	107.74
	B	82.90		177.60		
	C	77.40		162.60		

By observing the percent changes of 0 dBm and 8 dBm, the difference between the Coded PHY and 1M PHY is smaller at 8 dBm Tx power, but the distance under the same physical layer is doubled when it is switched from 0 dBm to 8 dBm, showing the effect of extending the communication range of the devices as shown in Table 4.7. In addition, based on these bar charts and positive percent change, the Coded PHY is determined to have a longer distance compared to 1M PHY at the same transmit power.

The higher transmit power allows the Bluetooth signal to travel further and increase the communication range because the stronger signal exists to penetrate obstacles compared to 0 dBm transmit power. Furthermore, the higher transmit power also ensures signal reliability to reduce packet loss. However, there is a trade-off that needs to be considered when increasing the transmit power, which leads to higher power consumption, shortening the battery life of the devices.

4.1.2 Maximum distance in an outdoor environment

In this case, combination A and combination B are included in the testing of measuring the maximum distance in an outdoor environment.

4.1.2.1 Tested in different heights

The variable settings in this test are the physical layers (1M PHY and Coded PHY) and heights (1.4 m, 2.0 m, and 3.0 m). The transmit power of the evaluation board is set to 0 dBm for this test.

Table 4.8: Comparison between PHYs at different heights.

PHY	Height, h (m)	Combination	Distance, d (m)
1M	1.4	A	197.62
		B	230.41
	2.0	A	419.91
		B	466.17
	3.0	A	479.91
		B	525.17
Coded	1.4	A	387.48
		B	416.59
	2.0	A	878.23
		B	911.48
	3.0	A	961.23
		B	963.48

The results of different devices under different physical layers against different heights (h) are recorded as shown in Table 4.8. The comparison of physical layers in distance at different heights is plotted in the bar chart as shown in Figure 4.3.

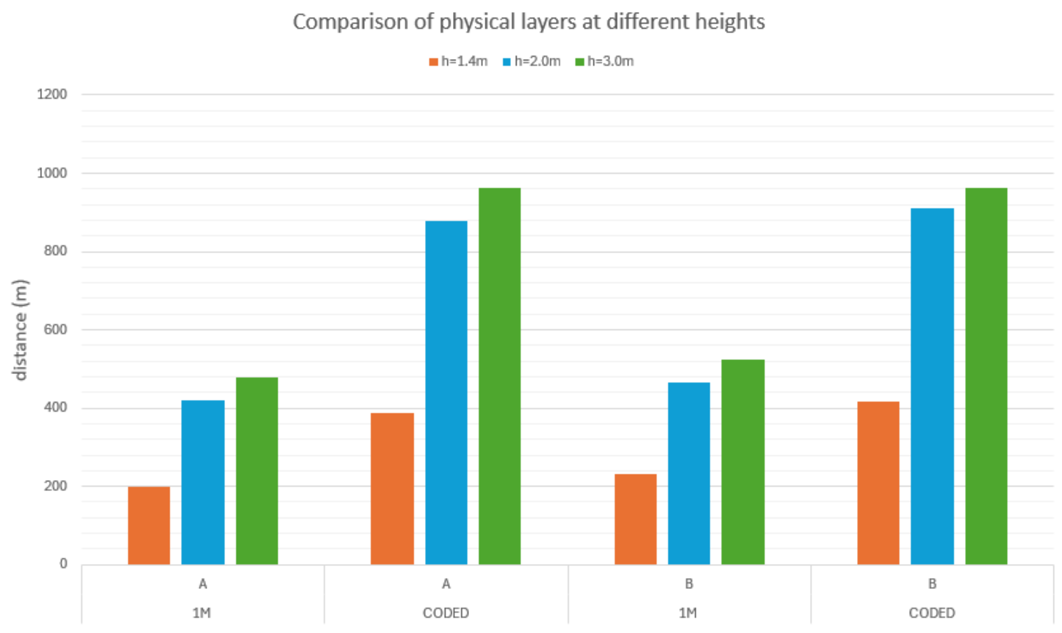


Figure 4.3: Comparison of PHYs at different heights.

Table 4.9: Percent change between PHYs at different heights.

Height, h (m)	**Combination**	**Distance, d (m)**				**Percent change (%)**
		1M		**Coded**		
1.4m	A	197.62	214.02	387.48	402.04	87.85
	B	230.41		416.59		
2.0m	A	419.91	443.31	878.23	894.86	101.86
	B	466.17		911.48		
3.0m	A	479.91	502.54	961.23	962.36	91.43
	B	525.17		963.48		

Based on the observation in Table 4.9, the percent change shows that the distance under Coded PHY increases up to 87.85 %, compared to 1M PHY at 1.4 m. The percent change recorded for 2.0 m and 3.0 m, showing the Coded PHY increased the distance by around 100 % compared to 1M PHY. Based on these bar charts and positive percent change, the Coded PHY is found to have a longer distance compared to 1M PHY at the same height, which further verifies the observation conducted on height changes to distance in the indoor environment. Based on Figure 4.4, a comparison is done for the results at different heights in the indoor and outdoor environments, finding an increase in the distance under the same physical layers. The distance data collected in the indoor environment is found to be smaller than in the outdoor environment. This is because the indoor results are collected within the building, and the Bluetooth signals are attenuated based on many factors such as obstructions and physical barriers, multipath interference, and material absorption, resulting in the distance measured in the indoor environment being shorter than the outdoor results.

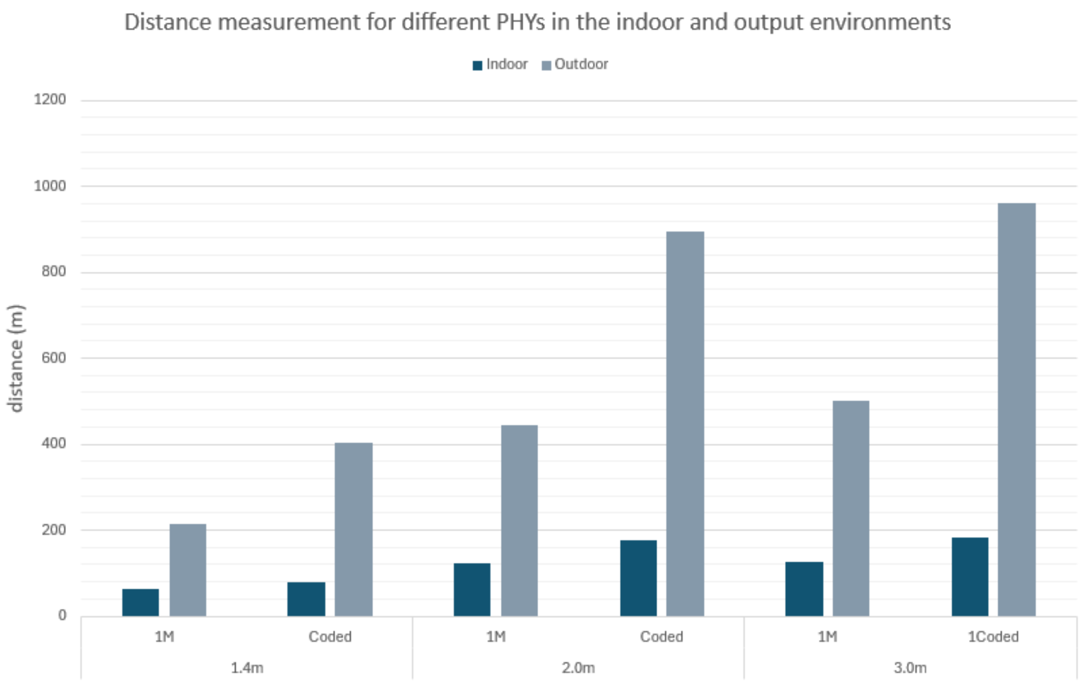


Figure 4.4: Comparison of PHYs and environments at each height.

4.1.2.2 Tested in different transmit (Tx) powers

The variable settings in this test are the physical layers (1M PHY and Coded PHY) and transmit powers (0 dBm and 8 dBm) The height setting for the evaluation board is set to 1.4 m for this test.

Table 4.10: Comparison between PHYs at different transmit powers.

PHY	Tx Power (dBm)	Combination	Distance, d (m)
1M	0.0	A	197.62
		B	230.41
	8.0	A	368.62
		B	394.41
Coded	0.0	A	387.48
		B	416.59
	8.0	A	781.48
		B	800.59

The results of different devices under different physical layers against different transmit power are recorded as shown in Table 4.10. The comparison of physical layers in distance at different transmit powers is plotted in the bar chart as shown in Figure 4.5.

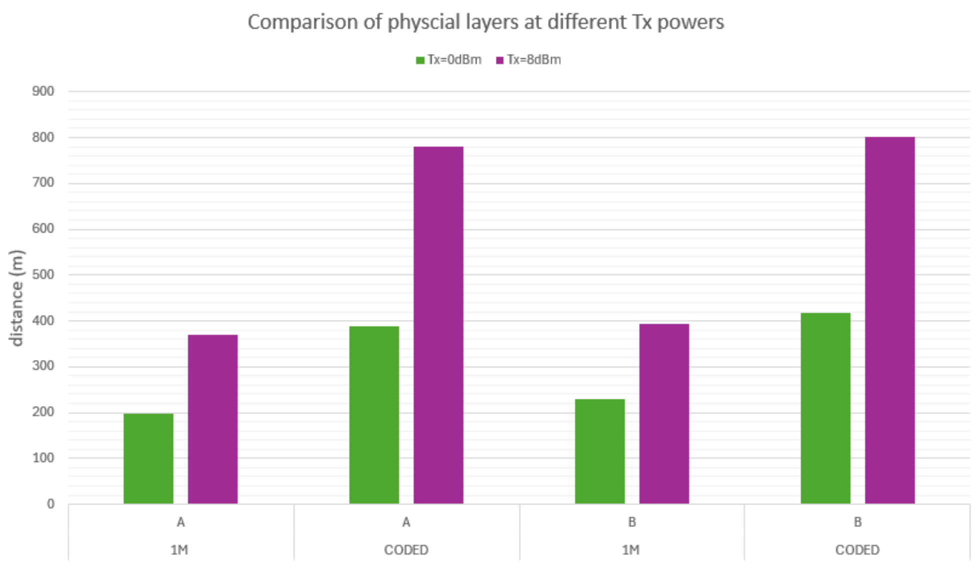


Figure 4.5: Comparison of PHYs at different transmit powers.

Table 4.11: Percent change between PHYs at different heights.

Tx Power (dBm)	Combination	Distance, d (m)				Percent change (%)
		1M		Coded		
0.0	A	197.62	214.02	387.48	402.04	87.85
	B	230.41		416.59		
8.0	A	368.62	381.52	781.48	791.04	107.34
	B	394.41		800.59		

The comparison of physical layers among different transmit powers is summarized and the percent change in distance is calculated using as shown in Table 4.11. The average distance of different combinations is obtained to calculate the percentage difference between different physical layers. Based on the observation in Table 4.11, the percent change shows that the distance under Coded PHY increases up to 87.85 %, compared to 1M PHY at 0 dBm transmit power. The percent change recorded for 8 dBm transmit power, shows the Coded PHY increase the distance above 107.34 % compared to 1M PHY. Based on Figure 4.6, a comparison is done for the results at different transmit powers in the indoor and outdoor environments, finding an increase in the distance under the same physical layers. It can also be discovered that the outdoor results are longer than the indoor results, due to less attenuation in signal transmission as mentioned previously, which needs to be considered in the indoor environment.

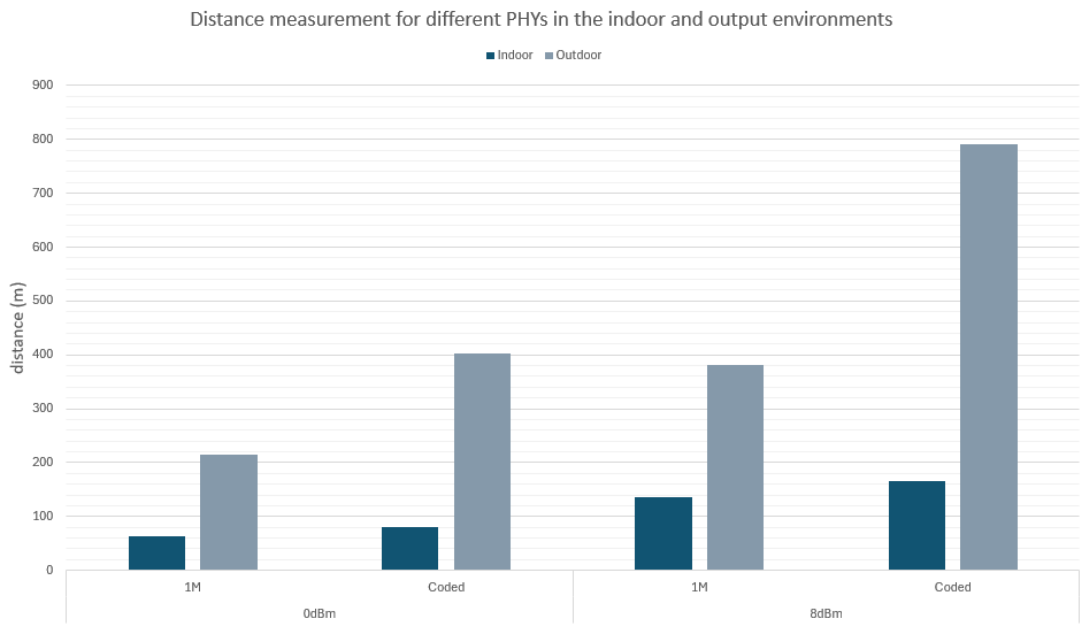


Figure 4.6: Comparison of PHYs and environments at each transmit power.

4.2 Comparison of filters applied to RSSI readings

Based on the studies reviewed in the previous section, the RSSI readings are unstable and unreliable, as the RSSI can be varied by environmental interference and noise, and the fading effect in the signal leads to fluctuation. In this case, some filters are used to smooth the RSSI value, preventing the unexpected fluctuation in RSSI values. The filters implemented are the moving average (MA) filter, exponential moving average (EMA) filter, Kalman filter, and median filter. The tests conducted can be classified into two scenarios: placing the board at a fixed distance and carrying the mobile device to different points. There are 120 values of unfiltered RSSI values and filtered RSSI values to be collected and compared.

4.2.1 Fixed distance

A mobile device (Mi 9T Pro) and a board are involved in the test, the board is placed 5 m away from the mobile device as shown in Figure 4.7, the height is 1.4 m and the transmit power is 0 dBm. There are 100 values of unfiltered RSSI values and filtered RSSI values to be collected and compared. The results collected for unfiltered and filtered RSSI values are plotted as shown in Figure 4.8. Based on the plot, the fluctuation of raw RSSI values exists although the board is treated to be a stationary object. Observing the effect of each filter on the RSSI values as shown in Figure 4.9, shows that the filtering methods are useful and critical to alleviate the fluctuation of RSSI values, as the spikes in raw RSSI values are reduced or lowered compared to filtered RSSI values.

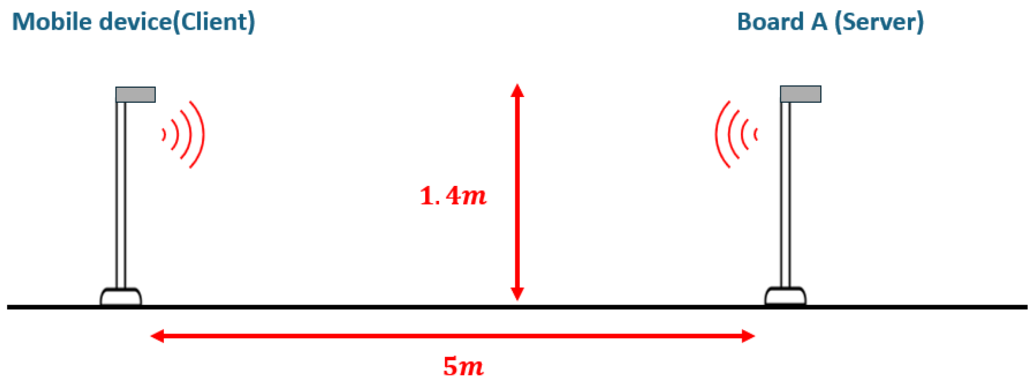


Figure 4.7: Placement of the board and mobile device in the fixed distance test.

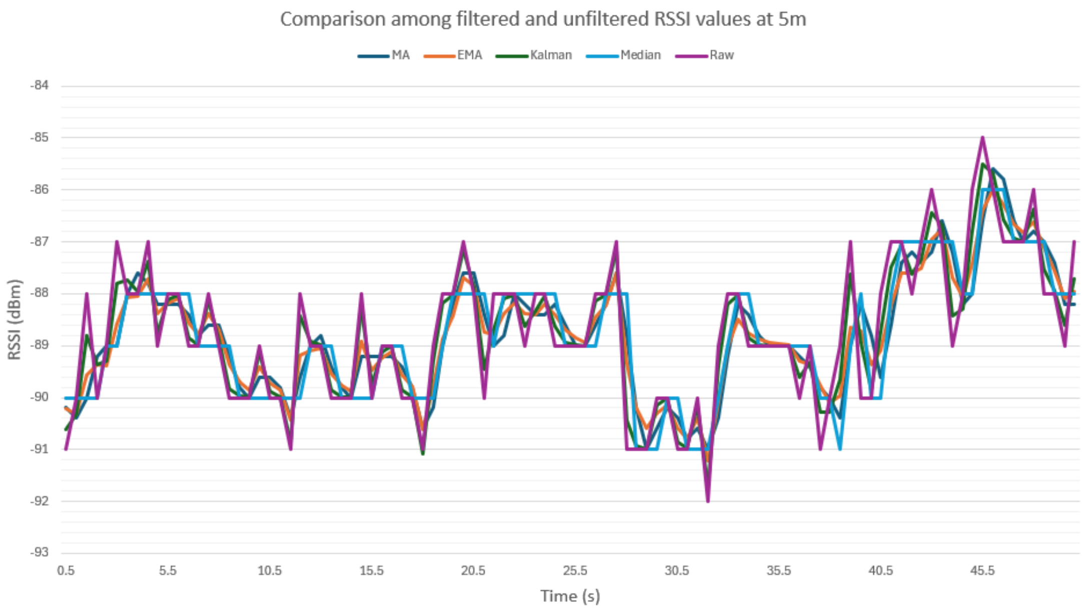


Figure 4.8: The unfiltered and filtered RSSI values at 5 m.

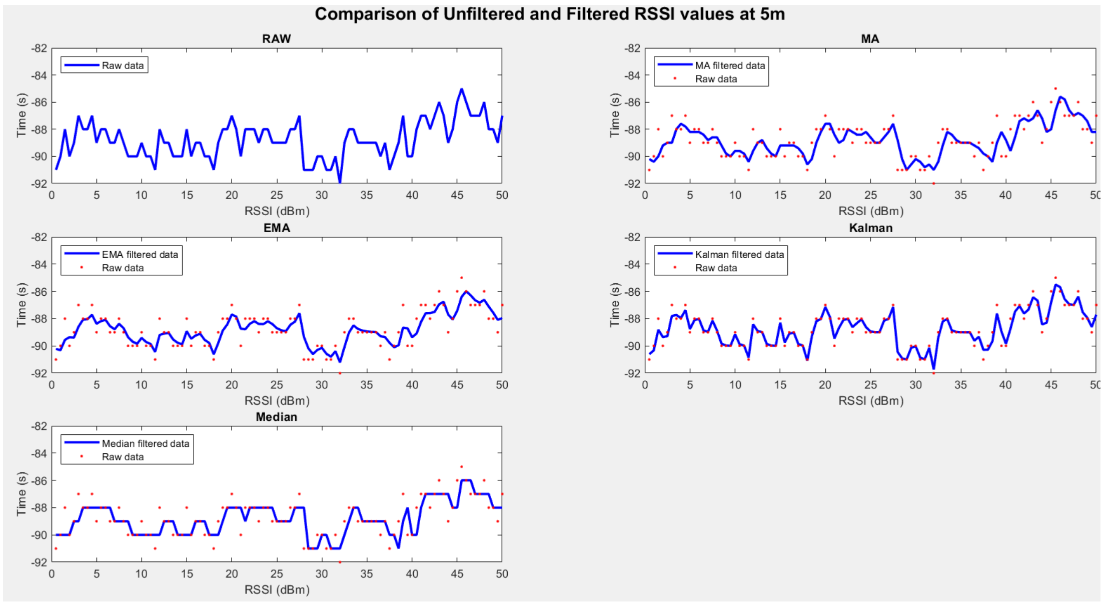


Figure 4.9: Comparison between filtered and unfiltered data.

4.2.2 Variable distance

A mobile device (Mi 9T Pro) and a board are involved in the test, the mobile device is carried to move to other points as shown in Figure 4.10, and the height is 1.4 m and the transmit power is 0 dBm. There are 120 values of unfiltered RSSI values and filtered RSSI values to be collected and compared. The results collected for unfiltered and filtered RSSI values are plotted as shown in Figure 4.11. Based on the plot, the shape of the RSSI curve is close to a U-shaped graph which is expected based on the test conducted. Observing the effect of each filter on the RSSI values as shown in Figure 4.12, shows that the filtering methods are still useful to smooth the signal output when the asset is moving.

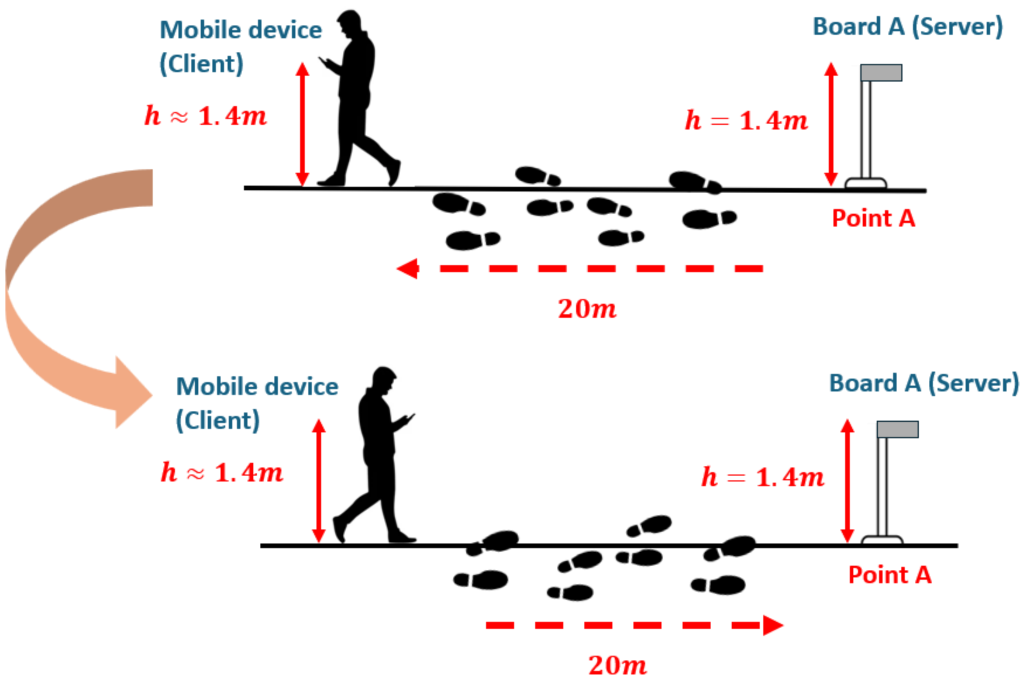


Figure 4.10: Placement of the board and mobile device in the variable distance test.

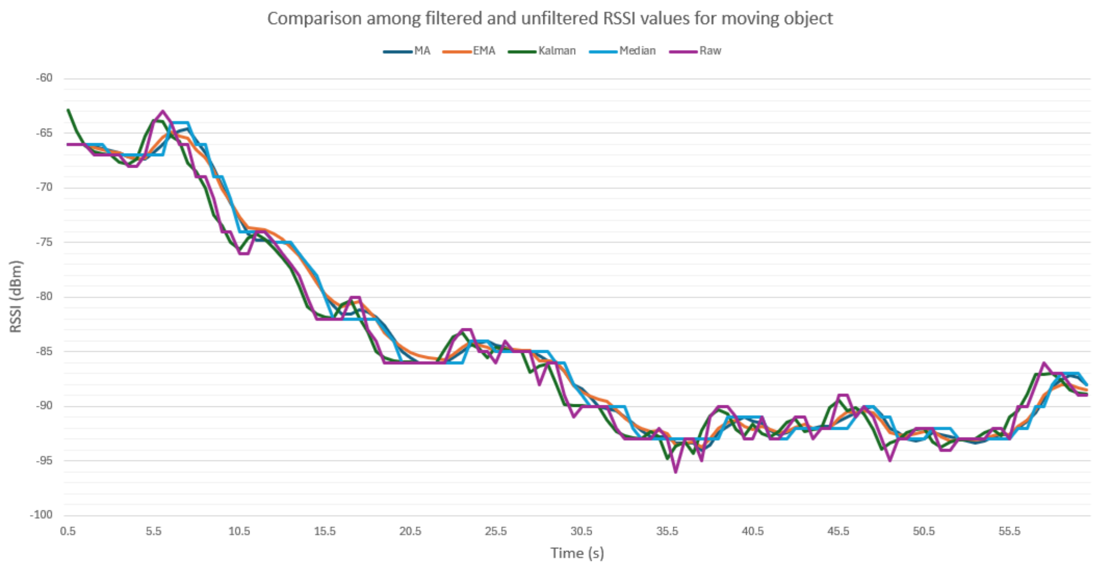


Figure 4.11: The unfiltered and filtered RSSI values at variable distance test.

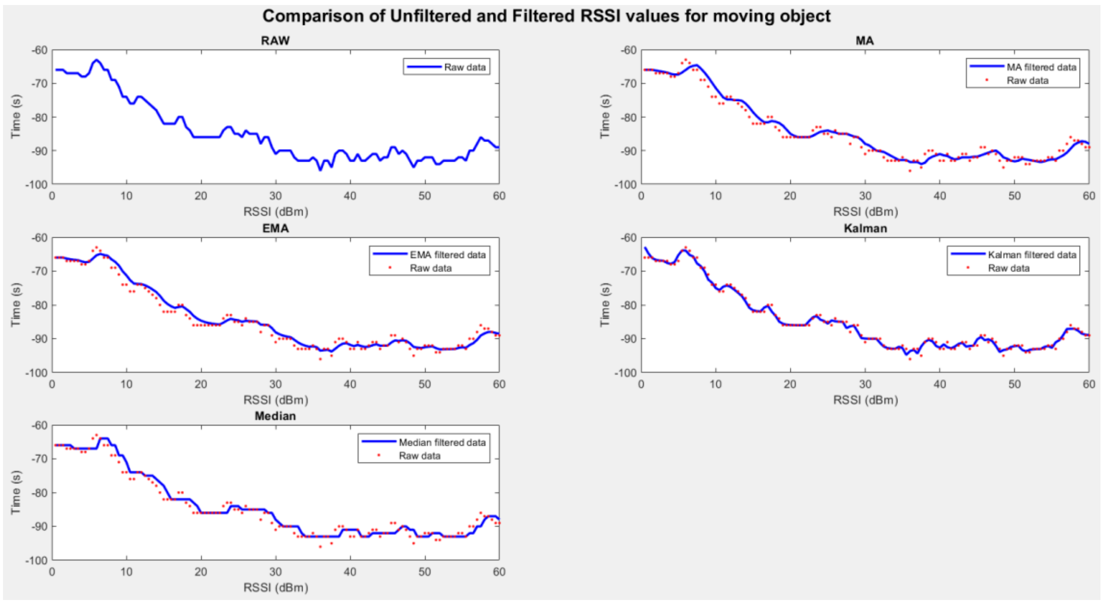


Figure 4.12: Comparison between filtered and unfiltered data.

4.2.3 Summary

Based on the previous results collected as shown in Figure 4.9 and Figure 4.12, all the filtered RSSI values are better than the raw RSSI values as all of them are observed to reduce the noise or spike of the raw RSSI values, improving the reliability. Furthermore, their characteristics are found, in the moving average (MA) filter, it is easy to implement, but a fixed delay is found as it is dependent on the size of the window to get the average of the data. The larger size of window can be used to further smooth the data, but the lag is increased. For the exponential moving average (EMA) filter, there will be less lag as it is not dependent on the window size, which will also be more responsive to recent data compared to the MA filter. Furthermore, a smoothing factor (alpha) is available to further smooth the data based on the requirements.

In addition, the Kalman filter is introduced to be an adaptive filter that estimates the current state based on the previous state estimates and noisy measurements, making it effective in dynamic environments which can be proved based on the Kalman filtered RSSI values in Figure 4.12, showing that it is better than in generating the similar RSSI graph from raw data compared to other filtering techniques while smoothing the data. Based on the advantage of adaptive filtering and predictive in the Kalman filter, it is suitable for tracking but the implementation is more complex than others.

In terms of the median filter, it obtains the median of neighbouring data points in window size, it can effectively remove the outliers or the noise spikes. However, unlike the other filters, it is a non-linear filter, it preserves edges and sharp transitions, making the results to be not smooth compared to other filters.

4.3 Trilateration (RSSI based only)

The distance estimation for the beacon and mobile device (Samsung Tab S7) is obtained using Equation (2.3) while the position of the mobile device or the asset is determined using Equation (3.6). The mobile device is moved to five different positions and the RSSI values to other beacons are recorded. The distance and position estimated will be compared to the real distance and position that are obtained in Google Maps for the outdoor environment test.

4.3.1 Distance

Table 4.12: Path loss exponent for each beacon.

Beacon	$\boldsymbol{RSSI_{1m}(dBm)}$	$\boldsymbol{RSSI_{100m}(dBm)}$	$\boldsymbol{n}$
A	-53.0	-85.5	$n = \frac{-53-(-85.5)}{10 \times \log_{10}(100)} \rightarrow \quad 1.625$
B	-51.5	-92.4	$n = \frac{-51.5-(-92.4)}{10 \times \log_{10}(100)} \rightarrow \quad 2.045$
C	-48.0	-92.7	$n = \frac{-48-(-92.7)}{10 \times \log_{10}(100)} \rightarrow \quad 2.235$

Table 4.13: RSSI collected at different points.

Position of the object	RSSI (dBm)		
	Beacon A	Beacon B	Beacon C
V	-87.1	-103.0	-101.3
W	-91.3	-99.8	-98.1
X	-90.0	-102.0	-98.4
Y	-94.5	-92.3	-96.8
Z	-93.2	-95.8	-96.2

The beacons are placed at three different points as shown in Figure 4.13, and the mobile device is moved to five different positions to record the RSSI values. The path loss exponent in this case is recalculated for each beacon based on their environment using Equation (2.2), with the reference RSSI value at 1 m. The data of RSSI value at 100 m is recorded and used to estimate the optimum path loss exponent based on the environment as shown in Table 4.12. The RSSI values obtained by the object to each beacon at different positions are recorded as shown in Table 4.13. Based on the RSSI values, the distance between the mobile device and the beacon can be estimated using Equation (2.3).

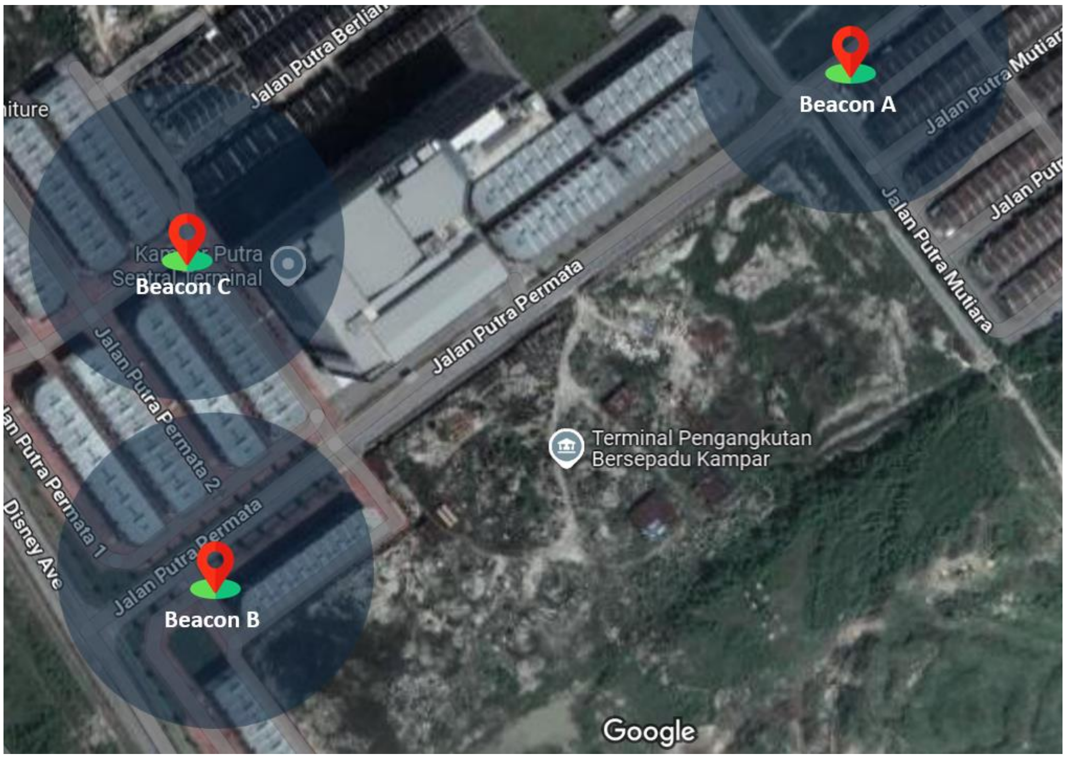


Figure 4.13: Position of beacons.

Table 4.14: Distance estimated of a mobile device and beacons.

Position of the object	Distance, d (m)		
	Beacon A	Beacon B	Beacon C
V	125.45	329.87	242.54
W	227.47	230.07	174.43
X	189.20	294.74	179.90
Y	357.97	98.88	152.56
Z	297.75	146.64	143.42

The distances of the mobile device to each beacon are estimated and recorded in Table 4.14. The real distances between the mobile device and beacons are determined using the software as shown in Figure 4.14, Figure 4.15, Figure 4.16, Figure 4.17, and Figure 4.18.

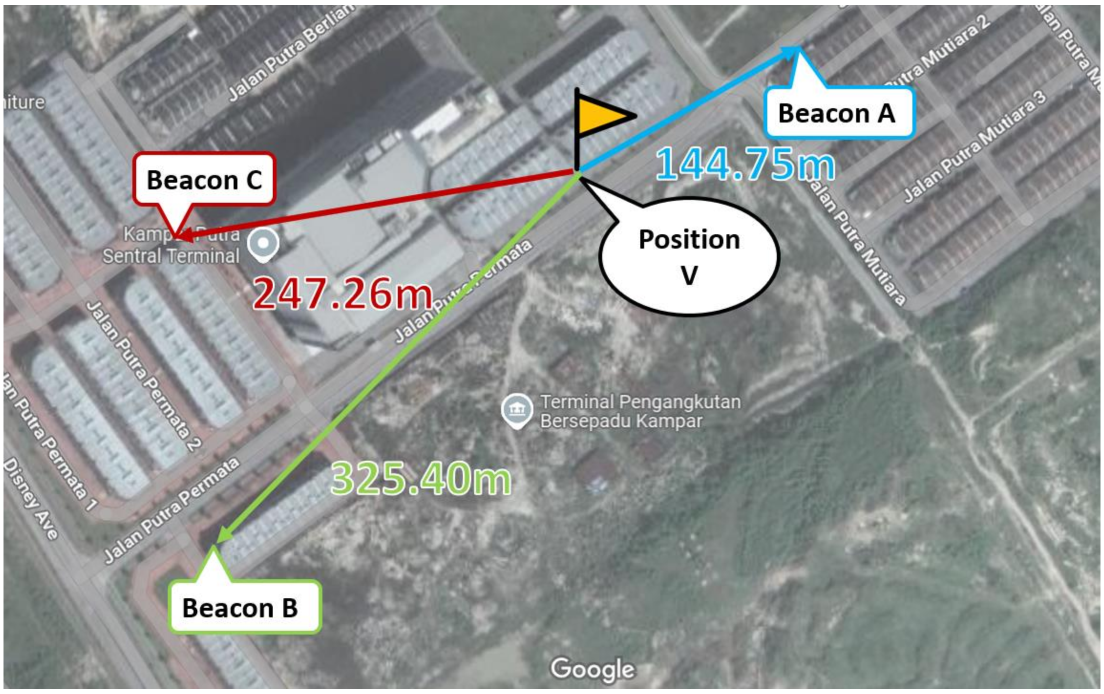


Figure 4.14: Distances between the beacons and the object at position V.

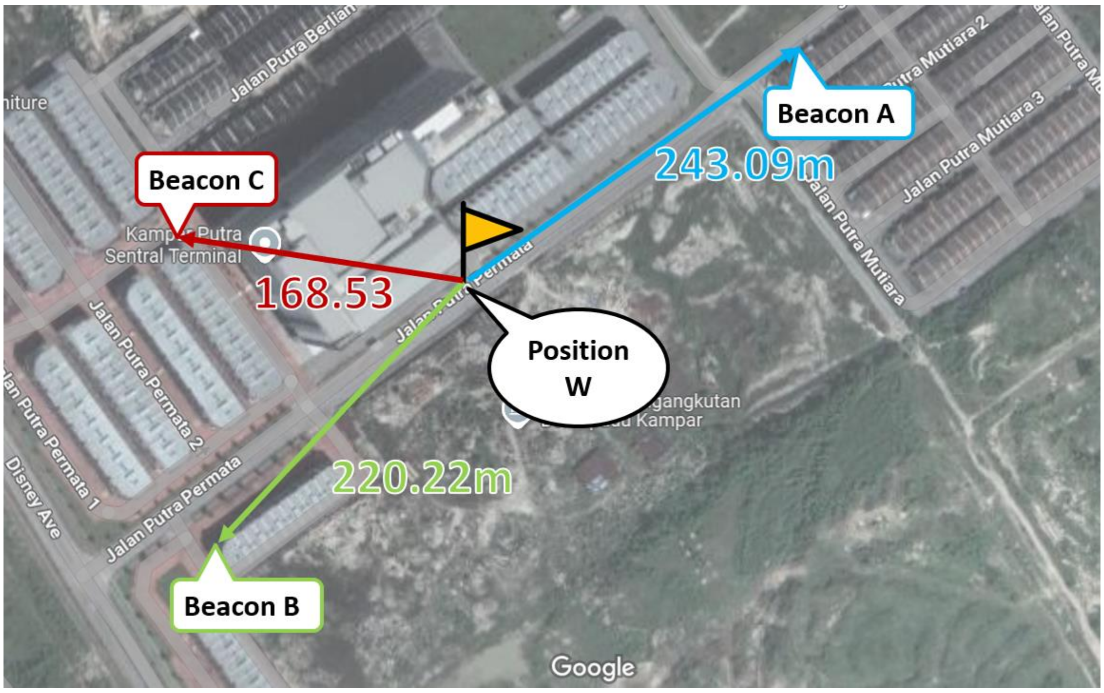


Figure 4.15: Distances between the beacons and the object at position W.

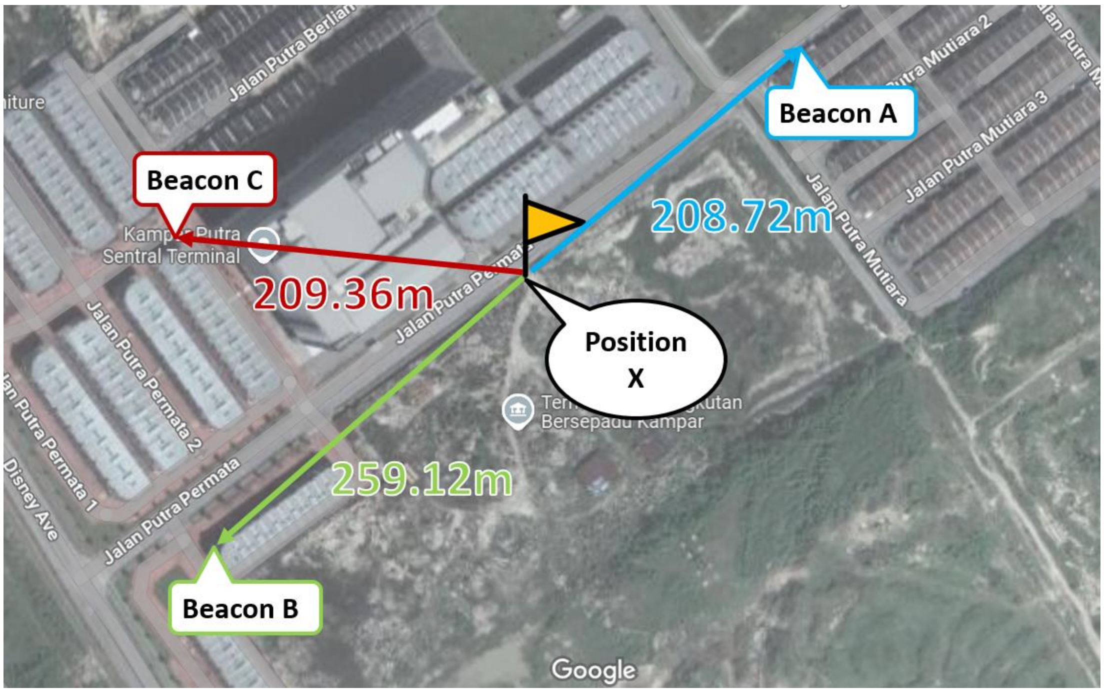


Figure 4.16: Distances between the beacons and the object at position X.

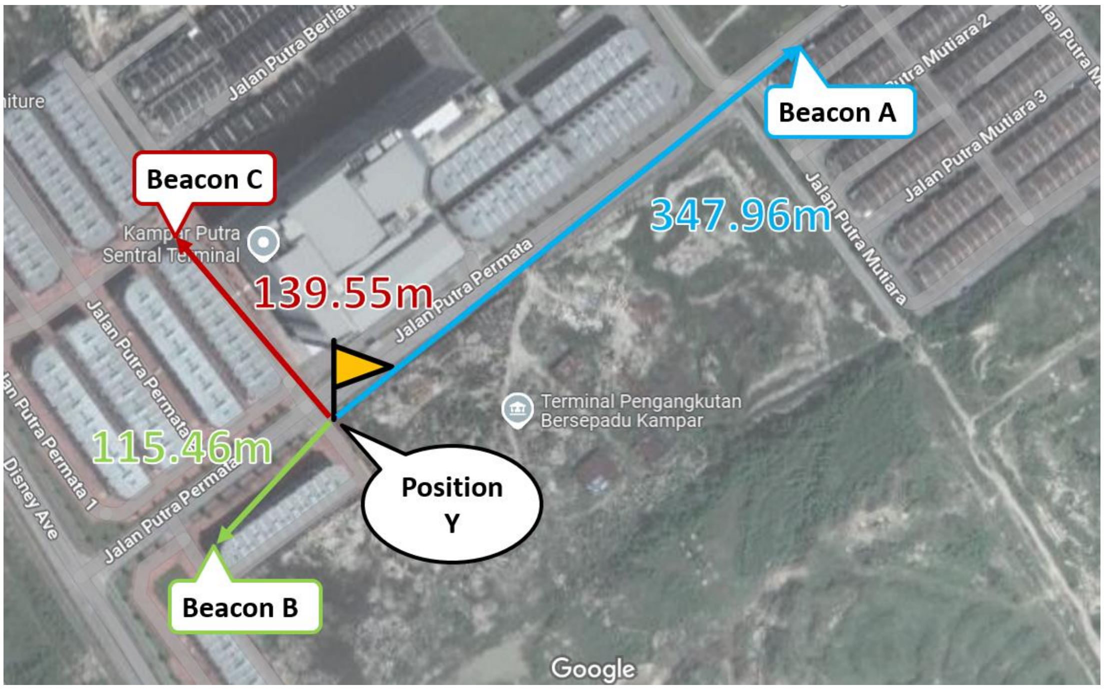


Figure 4.17: Distances between the beacons and the object at position Y.

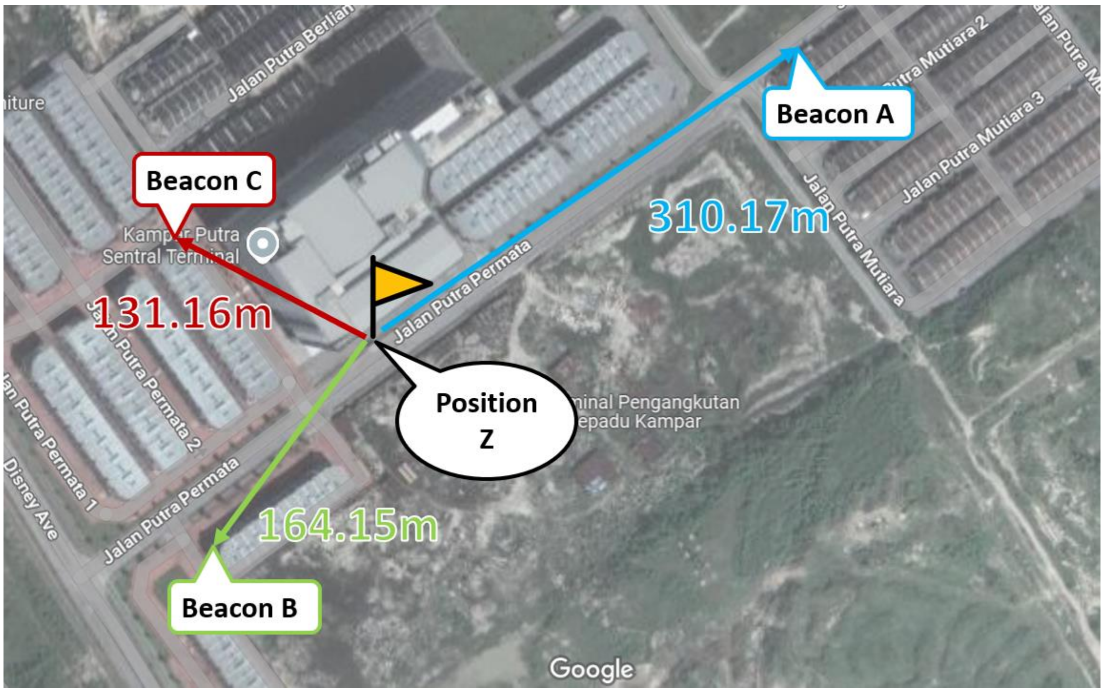


Figure 4.18: Distances between the beacons and the object at position Z.

Table 4.15: Comparison between real distance and estimated values.

Position of the object	Beacon	Distance, d (m)		Percent error (%)
		Estimated	Real	
V	A	125.45	144.75	13.33
	B	329.87	325.40	1.37
	C	242.54	247.26	1.91
W	A	227.47	243.09	6.43
	B	230.07	220.22	4.47
	C	174.43	168.53	3.50
X	A	189.20	208.72	9.35
	B	294.74	259.12	13.75
	C	179.90	209.36	14.07
Y	A	357.97	347.96	2.88
	B	98.88	115.46	14.36
	C	152.56	139.55	9.33
Z	A	297.75	310.17	4.00
	B	146.64	164.15	10.67
	C	143.42	131.16	9.35

A comparison between the estimated distance and the real distance is done in Table 4.15, and their percent error is calculated using Equation (4.2). Percent error on the distance between each beacon to different positions shows the difference between the estimated distance and real distance is lower than 15 % as shown in Figure 4.19, indicating the suitable path loss exponent helps to estimate the distance but this variation is still focused as it will significantly affect the position estimation using trilateration algorithms. Furthermore, the overall percent error value of the distance between position X and different beacons is found to be higher than in other cases, so it may lead to more error in position calculation compared to others. Moreover, the overall percent error at position W is lower than in other cases, so the position estimation will be expected to be more accurate compared to others.

$$percent\ error\ (\%) = \frac{|d_{estimated} - d_{real}|}{d_{real}} \times 100 \tag{4.2}$$

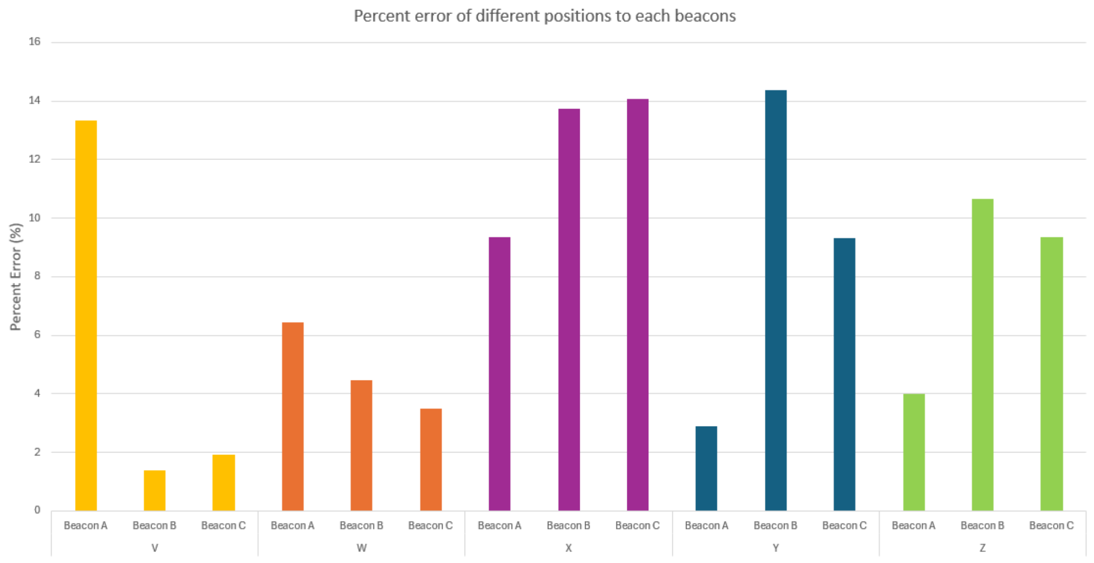


Figure 4.19: Percent error of different positions to each beacon.

4.3.2 Position

The positions of each beacon are recorded in latitude and longitude format as shown in Figure 4.20. The process of estimating the position can be illustrated in the figures below. Figure 4.21 shows the converted positions' value of each beacon from latitude and longitude data using Equation (3.13) and Equation (3.14) before performing the trilateration algorithm. By collecting the converted positions' values and the estimated distance using the path loss model in the previous section, the trilateration is applied using Equation (3.11) and Equation (3.12), as shown in Figure 4.22. The substitution method is selected to solve the position of the mobile device, and the results are converted back into latitude and longitude data as shown in Figure 4.23. The results are converted into Degrees Minutes Seconds (DMS) from coordinate data using Equation (3.15), Equation (3.16), and Equation (3.17) as shown in Figure 4.24 for determining the estimated positions on Google Maps. The DMS data of the estimated and real positions of the mobile device can be illustrated as shown in Figure 4.25.

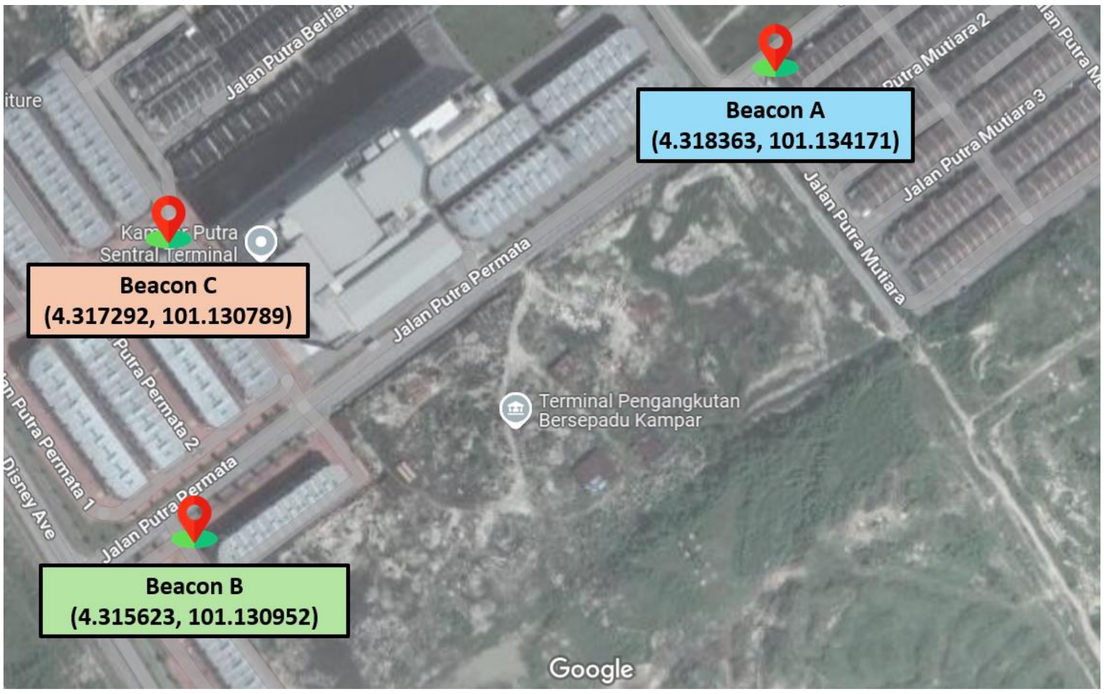
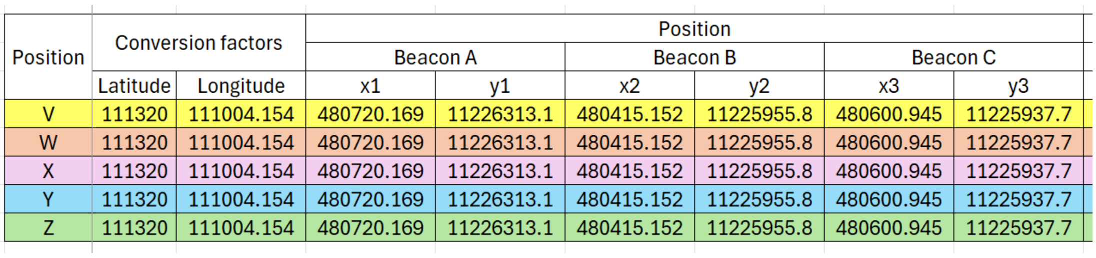
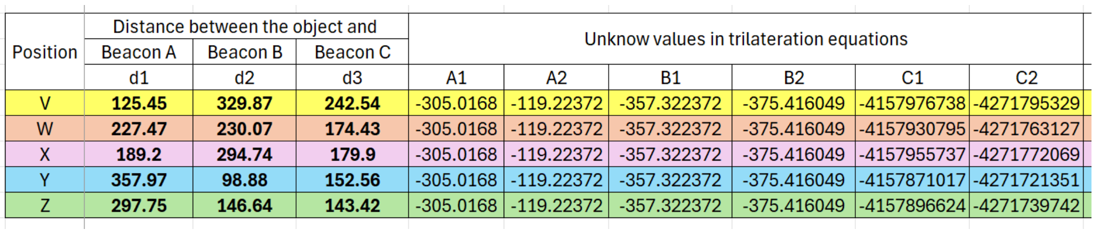


Figure 4.20: Coordinate of the beacons.



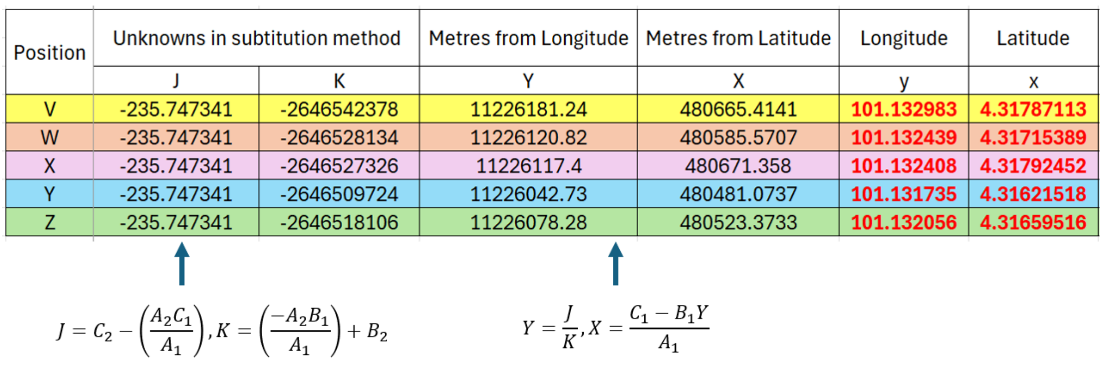
Position	Conversion factors		Position					
			Beacon A		Beacon B		Beacon C	
	Latitude	Longitude	x1	y1	x2	y2	x3	y3
V	111320	111004.154	480720.169	11226313.1	480415.152	11225955.8	480600.945	11225937.7
W	111320	111004.154	480720.169	11226313.1	480415.152	11225955.8	480600.945	11225937.7
X	111320	111004.154	480720.169	11226313.1	480415.152	11225955.8	480600.945	11225937.7
Y	111320	111004.154	480720.169	11226313.1	480415.152	11225955.8	480600.945	11225937.7
Z	111320	111004.154	480720.169	11226313.1	480415.152	11225955.8	480600.945	11225937.7

Figure 4.21: Converted positions' value of the beacons.



Position	Distance between the object and			Unknow values in trilateration equations					
	Beacon A	Beacon B	Beacon C						
	d1	d2	d3	A1	A2	B1	B2	C1	C2
V	**125.45**	**329.87**	**242.54**	-305.0168	-119.22372	-357.322372	-375.416049	-4157976738	-4271795329
W	**227.47**	**230.07**	**174.43**	-305.0168	-119.22372	-357.322372	-375.416049	-4157930795	-4271763127
X	**189.2**	**294.74**	**179.9**	-305.0168	-119.22372	-357.322372	-375.416049	-4157955737	-4271772069
Y	**357.97**	**98.88**	**152.56**	-305.0168	-119.22372	-357.322372	-375.416049	-4157871017	-4271721351
Z	**297.75**	**146.64**	**143.42**	-305.0168	-119.22372	-357.322372	-375.416049	-4157896624	-4271739742

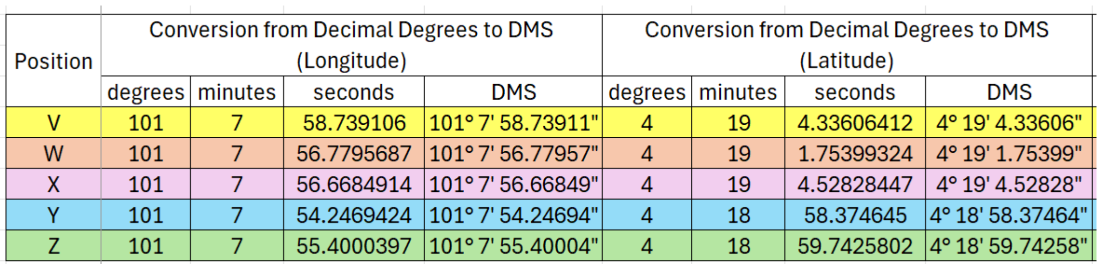
Figure 4.22: Solving trilateration using estimated distance and converted position.



Position	Unknowns in subtitution method		Metres from Longitude	Metres from Latitude	Longitude	Latitude
	J	K	Y	X	y	x
V	-235.747341	-2646542378	11226181.24	480665.4141	**101.132983**	**4.31787113**
W	-235.747341	-2646528134	11226120.82	480585.5707	**101.132439**	**4.31715389**
X	-235.747341	-2646527326	11226117.4	480671.358	**101.132408**	**4.31792452**
Y	-235.747341	-2646509724	11226042.73	480481.0737	**101.131735**	**4.31621518**
Z	-235.747341	-2646518106	11226078.28	480523.3733	**101.132056**	**4.31659516**

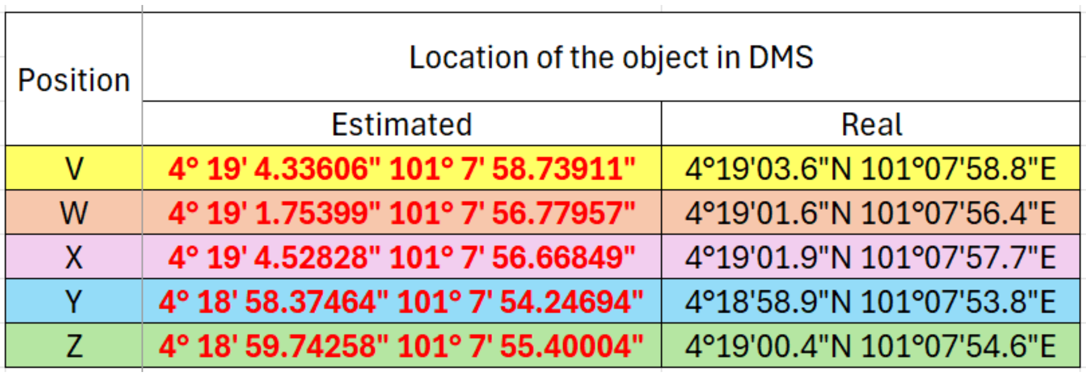
$$J = C_2 - \left(\frac{A_2C_1}{A_1}\right), K = \left(\frac{-A_2B_1}{A_1}\right) + B_2 \qquad Y = \frac{J}{K}, X = \frac{C_1 - B_1Y}{A_1}$$

Figure 4.23: Substitution is applied to solve the position of the object.



Position	Conversion from Decimal Degrees to DMS (Longitude)				Conversion from Decimal Degrees to DMS (Latitude)			
	degrees	minutes	seconds	DMS	degrees	minutes	seconds	DMS
V	101	7	58.739106	101° 7' 58.73911"	4	19	4.33606412	4° 19' 4.33606"
W	101	7	56.7795687	101° 7' 56.77957"	4	19	1.75399324	4° 19' 1.75399"
X	101	7	56.6684914	101° 7' 56.66849"	4	19	4.52828447	4° 19' 4.52828"
Y	101	7	54.2469424	101° 7' 54.24694"	4	18	58.374645	4° 18' 58.37464"
Z	101	7	55.4000397	101° 7' 55.40004"	4	18	59.7425802	4° 18' 59.74258"

Figure 4.24: Coordinates of the object are converted into DMS.



Position	Location of the object in DMS	
	Estimated	Real
V	**4° 19' 4.33606" 101° 7' 58.73911"**	4°19'03.6"N 101°07'58.8"E
W	**4° 19' 1.75399" 101° 7' 56.77957"**	4°19'01.6"N 101°07'56.4"E
X	**4° 19' 4.52828" 101° 7' 56.66849"**	4°19'01.9"N 101°07'57.7"E
Y	**4° 18' 58.37464" 101° 7' 54.24694"**	4°18'58.9"N 101°07'53.8"E
Z	**4° 18' 59.74258" 101° 7' 55.40004"**	4°19'00.4"N 101°07'54.6"E

Figure 4.25: DMS data of the estimated and real positions.

Table 4.16: Distance difference between estimated and real positions.

Position	Coordinate in DMS		Distance difference (m)
	Estimated	Real	
V	4° 19' 4.33606" 101° 7' 58.73911"	4°19'03.6"N 101°07'58.8"E	23.75
W	4° 19' 1.75399" 101° 7' 56.77957"	4°19'01.6"N 101°07'56.4"E	16.29
X	4° 19' 4.52828" 101° 7' 56.66849"	4°19'01.9"N 101°07'57.7"E	88.81
Y	4° 18' 58.37464" 101° 7' 54.24694"	4°18'58.9"N 101°07'53.8"E	21.60
Z	4° 18' 59.74258" 101° 7' 55.40004"	4°19'00.4"N 101°07'54.6"E	32.73

The estimated values for different positions of the mobile device are pointed on Google Maps and recorded as shown in Figure 4.26, Figure 4.27, Figure, 4.28, Figure 4.29, and Figure 4.30. The difference in distance between the estimated and real positions is tabulated in Table 4.16. Based on the observation in Table 4.16, displaying position X has a higher error compared to other cases as expected due to its higher error in distance estimation discussed in the previous section. Vice-versa, the estimated position W is found to have a lower difference in distance to the real point due to better performance in distance estimation.

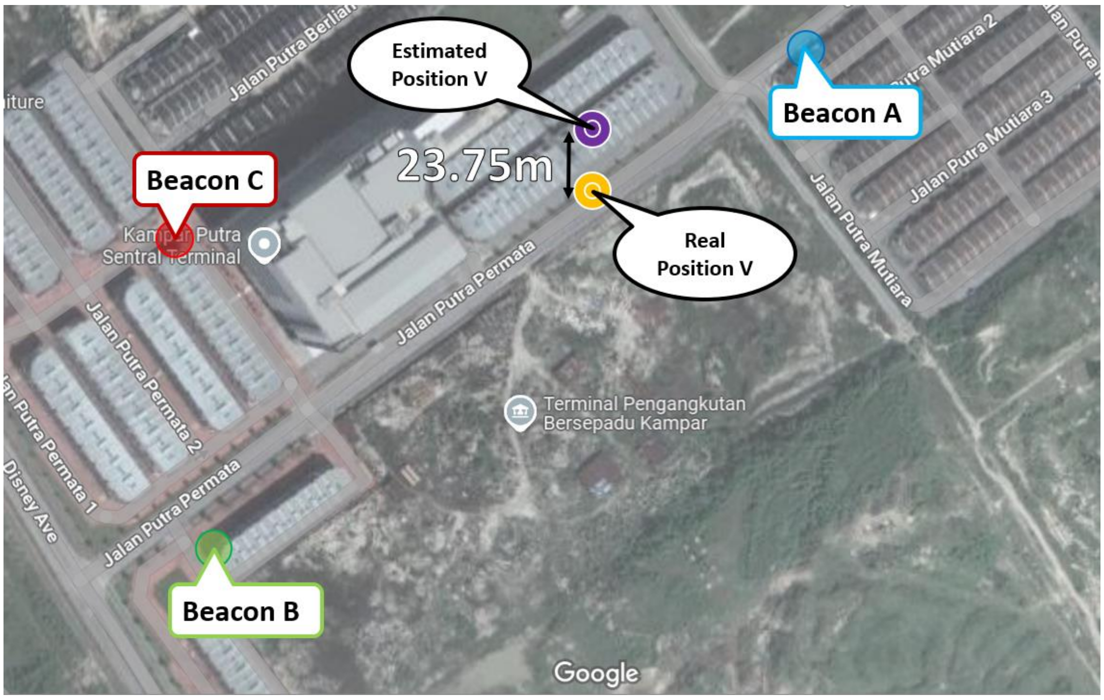


Figure 4.26: Estimated and real position V.

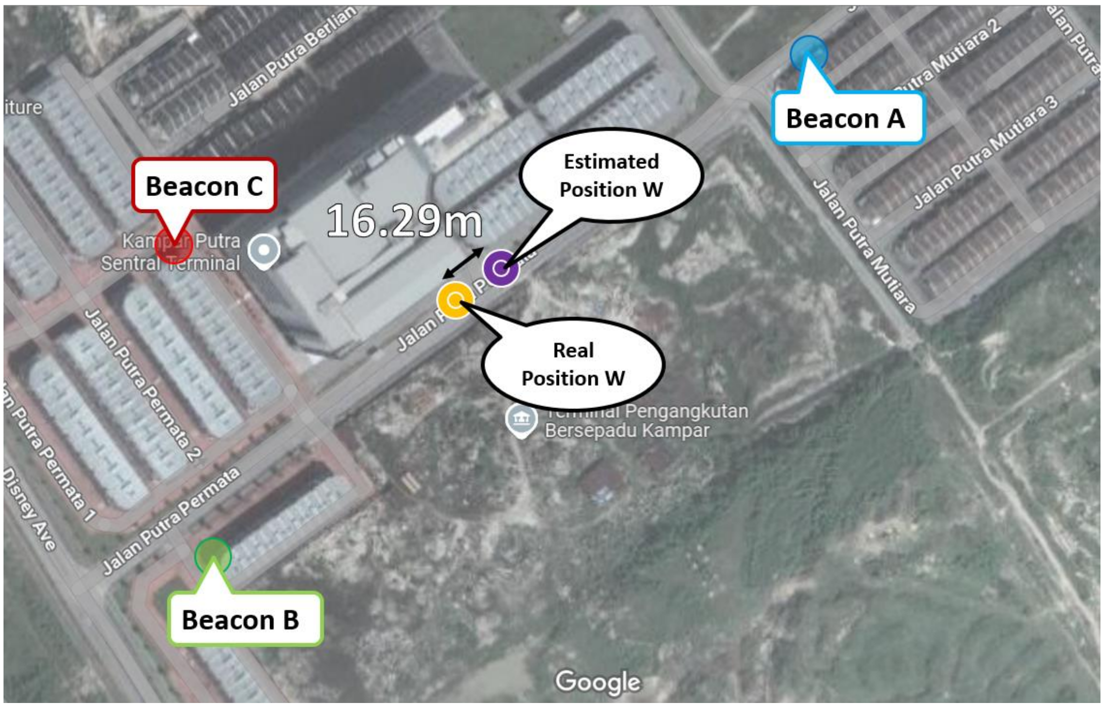


Figure 4.27: Estimated and real position W.

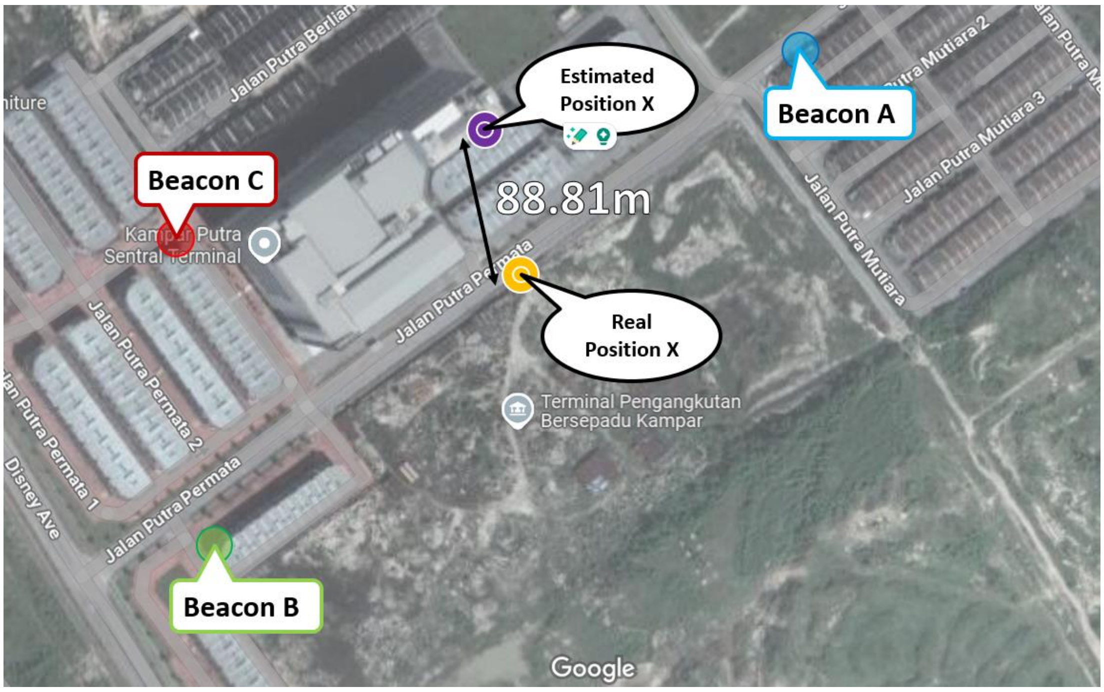


Figure 4.28: Estimated and real position X.

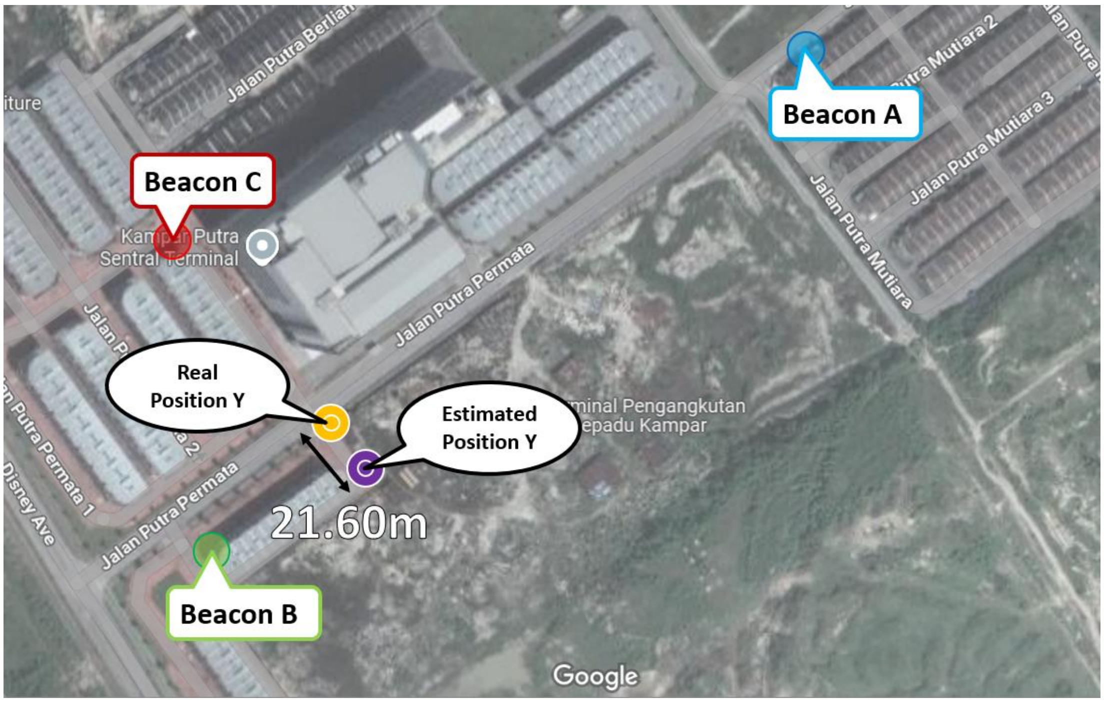


Figure 4.29: Estimated and real position Y.

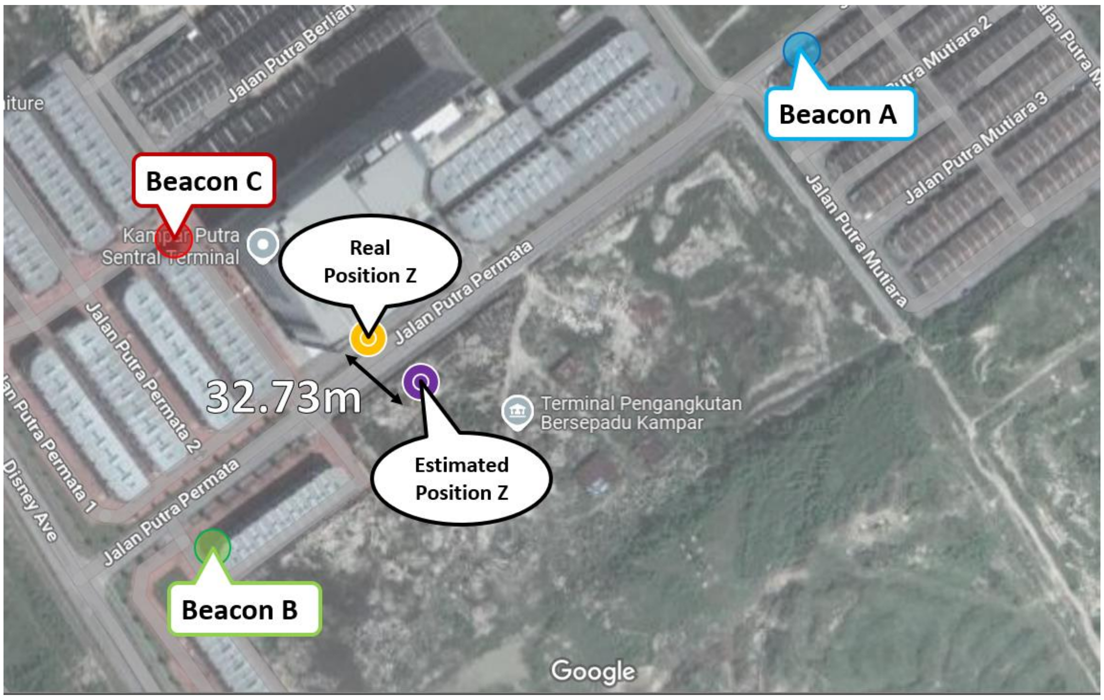


Figure 4.30: Estimated and real position Z.

4.3.3 Summary

Performing the trilateration for the outdoor environment involves the known positions of three beacons, and estimating the distance from these points to predict an unknown position of the object. The RSSI values are recorded and the distance is determined based on path loss mode. The suitable path loss exponent is calculated to improve the accuracy. By deploying the beacons at known positions, their coordinate can be used to estimate the position of the object or asset.

Selecting the Coded PHY to be the physical layer in communication between devices, helps to get the greater distance as it improves the signal range and penetration through outdoor obstacles. Hence, the presence of Coded PHY in Bluetooth technology provides an opportunity to perform long-distance communication compared to other physical layers, making the test not only restricted in an indoor environment.

Based on the results obtained for distance and position, the distance is proved to affect the position significantly. In other words, the lower the accuracy in the distance measurement, the worse performance in position determination. Furthermore, the distance difference in the estimated and real positions is found to be below 100 m, showing that the combination of trilateration techniques and Coded PHY is executable but the improvement in distance estimation needs to be taken into account for better results.

4.4 Combination of sensors and RSSI values

An Android application is developed to gather the sensor's data and record the RSSI values. The application is installed on a mobile device (Mi 9T Pro) and is used to scan and connect to the evaluation board. The Coded PHY is switched from 1M PHY once the connection is made to ensure that long-range communication between the devices is applicable. The EMA filter is applied on RSSI readings in this case due to its simple implementation and good smoothing techniques.

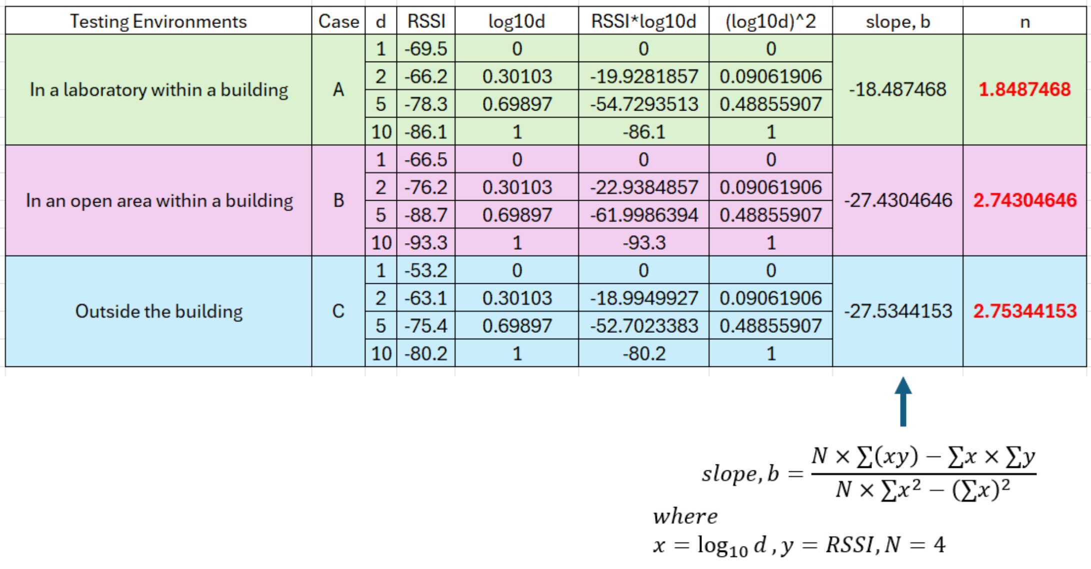
Table 4.17: Testing environments involved in the test.

Case	Testing Environments
A	In a laboratory within a building
B	In an open area within a building
C	Outside the building

The tests are conducted in different environments to check the performance of estimating the distance and direction. The testing environments are in a lab within a building, an open area within a building, and outside the building.

4.4.1 Distance

The distance between the mobile device and the board is estimated using the log distance path loss model as well. By running the Android application and connecting to the board, the filtered RSSI values are recorded, and the optimization of the path loss exponent or calibration implementation in the Android application is done at each testing environment to get better performance in estimating the distance. The reference RSSI at 1 m, 2 m, 5 m, and 10 m are recorded and different path loss exponent values are calculated for different testing environments. Figure 4.31 shows the process of getting the path loss exponent using the linear regression equation and the slope.



Testing Environments	Case	d	RSSI	log10d	RSSI*log10d	(log10d)^2	slope, b	n
In a laboratory within a building	A	1	-69.5	0	0	0	-18.487468	**1.8487468**
		2	-66.2	0.30103	-19.9281857	0.09061906		
		5	-78.3	0.69897	-54.7293513	0.48855907		
		10	-86.1	1	-86.1	1		
In an open area within a building	B	1	-66.5	0	0	0	-27.4304646	**2.74304646**
		2	-76.2	0.30103	-22.9384857	0.09061906		
		5	-88.7	0.69897	-61.9986394	0.48855907		
		10	-93.3	1	-93.3	1		
Outside the building	C	1	-53.2	0	0	0	-27.5344153	**2.75344153**
		2	-63.1	0.30103	-18.9949927	0.09061906		
		5	-75.4	0.69897	-52.7023383	0.48855907		
		10	-80.2	1	-80.2	1		

$$slope, b = \frac{N \times \sum(xy) - \sum x \times \sum y}{N \times \sum x^2 - (\sum x)^2}$$

$where$

$x = \log_{10} d\, , y = RSSI, N = 4$

Figure 4.31: Slope is calculated using linear regression to get path loss exponent.

Table 4.18: Comparison between estimated and real distances.

Case	Real distance (m)	Estimated distance (m)	Percent error (%)
A	3.00	2.148	28.40
	4.00	2.403	39.93
	6.00	5.572	7.13
	8.00	4.532	43.35
B	3.00	3.359	11.97
	4.00	4.467	11.68
	6.00	6.644	10.73
	8.00	7.756	3.05
	12.00	11.435	4.71
C	3.00	2.576	14.13
	4.00	3.896	2.60
	6.00	5.578	7.03
	8.00	7.739	3.26
	12.00	11.355	5.38

Unlike the trilateration applied in the previous section, there is one path loss exponent value needed for each case. Once the path loss exponent is calculated, it is applied to distance estimation. To increase the stability of distance calculation and reduce the effect of small changes in RSSI values, the average distance with 10 samples is used and displayed to users. The board is placed at different distances, and the average of fifty distance values is obtained based on RSSI and compared with the real distance, and the percent error is calculated as shown in Table 4.18. Figure 4.32 shows the overall percent error in case A is higher than in other cases, indicating the accuracy of distance estimation in the laboratory is lower compared to other areas.

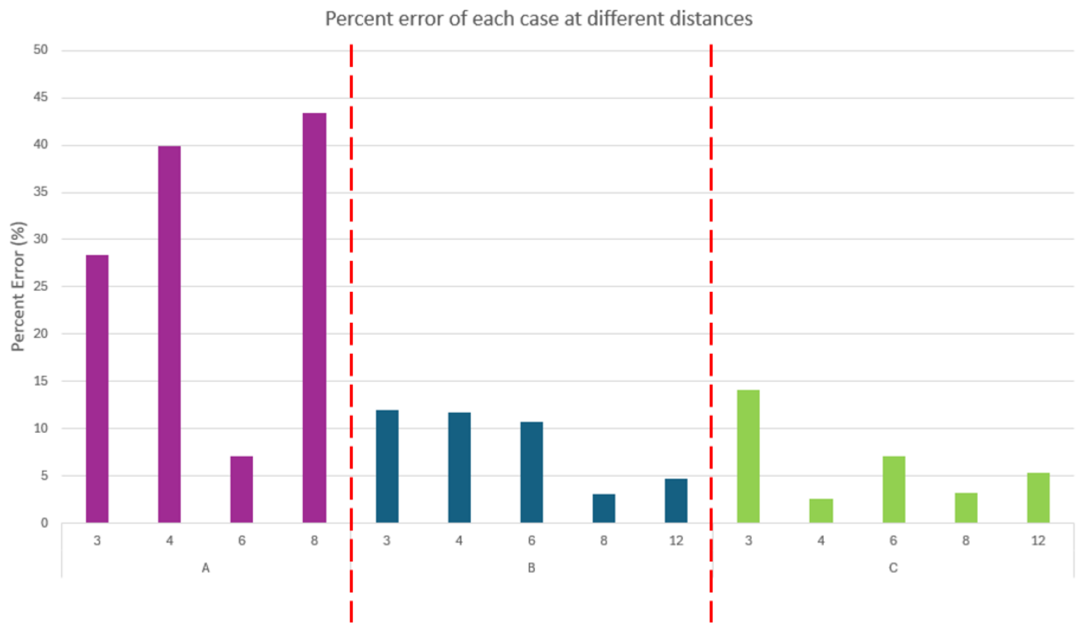


Figure 4.32: Percent error of each case at different distances.

The performance of distance estimation in case A is worse than in other cases due to the multipath interference such as radio signals often bouncing off walls, ceilings, and electronic equipment in a room. Furthermore, interference from other devices exists, such as computers, Wi-Fi routers, and other wireless devices operating in a similar frequency band, leading to unstable RSSI readings.

4.4.2 Direction

The test is conducted using a combination of sensors and RSSI readings to estimate the direction. The sensors used in this case are the accelerometer and magnetometer from the mobile device. The accelerometer is used to determine the orientation of the mobile device; the magnetometer is applied to identify the reference direction (true north), measuring angles to this direction. Three orientations of the mobile devices are tested such as A: vertical (90 degrees), B: diagonal (45 degrees), and C: horizontal (0 degrees) to the horizontal plane.

Table 4.19: Comparison between estimated and real directions at 1 m.

NO.	Estimated Direction		
	Orientation A	**Orientation B**	**Orientation C**
1	315	0	45
2	0	0	0
3	0	0	0
4	0	45	45
5	0	270	0
6	0	0	0
7	315	270	180
8	315	0	0
9	45	45	270
10	0	315	45
11	270	180	45
12	0	0	270
13	0	45	0
14	0	180	270
15	45	0	45
16	315	315	315
17	45	270	270
18	90	0	315
19	0	45	0
20	0	270	0

Table 4.20: Comparison between estimated and real directions at 5 m.

NO.	Estimated Direction		
	Orientation A	Orientation B	Orientation C
1	270	270	225
2	0	225	315
3	315	135	315
4	315	180	45
5	270	135	0
6	45	45	0
7	45	135	45
8	0	270	90
9	270	0	135
10	45	90	180
11	135	135	180
12	0	135	270
13	135	45	180
14	180	270	45
15	45	45	90
16	45	225	45
17	315	0	90
18	45	180	90
19	270	90	0
20	0	45	225

The test is conducted in an indoor environment and the board is placed at 1 m and 5 m respectively on the mobile device with three different orientations of the mobile devices. The board is placed in the true north direction (0 degrees) of the mobile device. The comparison between the estimated and real direction is done at each orientation of the mobile device as shown in Table 4.19 and Table 4.20. The estimated direction results are also plotted as shown in Figure 4.33 and Figure 4.34.

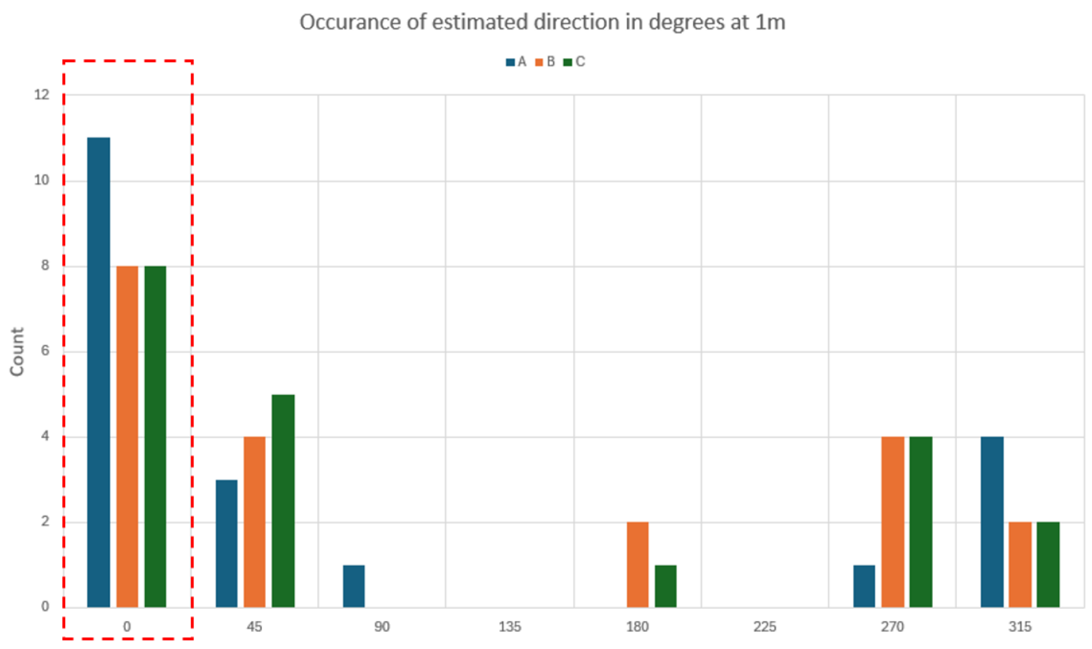


Figure 4.33: Estimated direction at 1 m.

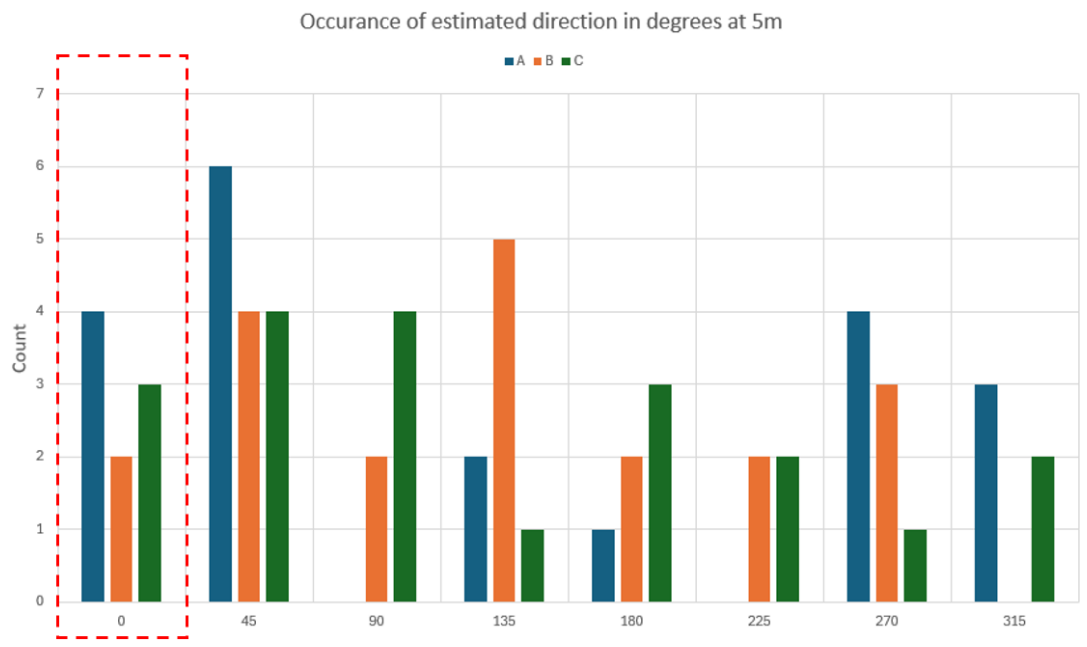


Figure 4.34: Estimated direction at 5 m.

Based on observations in Figure 4.33 and Figure 4.34, the performance of direction estimation is found to be better if the devices are placed in a shorter distance. The orientation A (the mobile device is held horizontally), is found to be more accurate compared to other orientations. By observing the data collected for orientation A at 5 m as shown in Figure 4.34, it can be found that the accuracy of the direction estimation (to 0 degrees) is reduced, but the result is still close which can be determined based on high occurrence at 45 and 315 degrees. The effect of orientation in this direction in this case is determined, there are different outcomes when the phone is held in different orientations; it is considered to be dependent on the hardware such as the design and the quality of the antenna, leading to difference performance in other orientations.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The comparison between 1M PHY and Coded PHY is done in terms of distance, the advantage of Coded PHY over 1M PHY in distance measurement is verified, and an increase in distance is observed once the devices are communicated in Coded PHY from 1M PHY. The factors such as height and transmit powers are included in the tests to determine their effect on distance measurement. Based on the results obtained based on these variable factors in outdoor and indoor environments, they are observed to further increase the distance by increasing these values. However, the power consumption needs to be taken into account as increasing the transmit power of the device, the higher battery drain is required by the device. Furthermore, the different devices with Bluetooth connections are found to have different signal strengths and communication ranges due to the dissimilarity in hardware and software. Furthermore, by comparing the results in the indoor and outdoor environments under the same settings of the height and transmit power, finding that the indoor distance results are shorter than the outdoor environment; it is expected due to the Bluetooth signals in a building are affected and attenuated due to the noise and interference.

In addition, the purpose of the filters applied to the signal is discussed. The filters such as MA, EMA, Kalman, and median filters are implemented, they can smooth the signal by reducing the fluctuation in the signal, improving the stability and reliability of the system.

Moreover, the trilateration is performed using Coded PHY in the communication between devices to estimate the distance and position of the asset. Based on the results, the accuracy of position estimation is highly dependent on the distance. By improving the distance estimation, the position is expected to be estimated more accurately. This test also proves that the Coded PHY and beacons in Bluetooth technology and the application of the trilateration algorithm can be one of the methods for performing long-range asset tracking.

Furthermore, an Android application is developed to gather the sensors' data and utilize them with RSSI values to estimate the distance and direction of the mobile device relative to the board or the asset. Using linear regression and slope to get the optimum path loss exponent, the accuracy of the distance estimation is better than guessing the random path loss exponent. By taking advantage of sensors, the orientation of the phone and the angle of the phone to the board are considered, and the direction estimation is executable by using the sensors and RSSI readings only.

5.2 Limitations

In this study, the application of Coded PHY in asset tracking is highly dependent on the RSSI readings. It is usable to meet the objectives if the connection between the devices is done in environments that are not noisy or have less interference. This is because the RSSI is significantly affected by environmental factors. For instance, signal attenuation and reflection can be found in environmental conditions such as obstacles, walls, and people, leading to signal fluctuations. Furthermore, the signal fading and many similar frequency bands of 2.4 GHz exist in the surroundings, making the data measured based on RSSI less accurate.

In addition, a buzzer solution is introduced to address the problem of finding assets at different height levels. By applying a buzzer and controlling it with a remote button on the Android application, users can easily detect their asset based on the signal strength and feedback of the buzzer if the asset is near to users. The buzzer is designed to be triggered under specific conditions. However, the power consumption needs to be considered, as the battery drains more in this solution, compared to the application without any other outputs.

The direction estimation using the sensors' data of the mobile device and the RSSI readings, is found to be limited at distance. In this case, this technique is used to gather the information from the accelerometer and magnetometer to get the orientation and angle and check the mean RSSI readings, but it is found that the accuracy drops when the distance is increased.

5.3 Recommendations for improvements

The improvement that can be done in the application to get more accurate results is conducting more calibrations to get an optimum path loss exponent for the current environment. By doing calibration when the devices are moved to different environments, the distance estimation based on the path loss model can be better. Furthermore, by increasing the sets of reference RSSI in linear regression and slope methods to get the path loss exponent, a more suitable value can be obtained and used to increase the accuracy of distance estimation.

Furthermore, the combination of filtering techniques can be implemented to get a more smooth and stable RSSI reading. Instead of taking advantage of a single filter, the combination of filters provides different benefits, helping the fluctuation and the noise of the signal to be alleviated or eliminated.

In addition, multiple data sources are recommended if the budget constraint is not mainly focused. For instance, the integration data from the sensors such as accelerometer, gyroscope, magnetometer, and barometer. They can be used in dead reckoning and correcting drift in the estimation, operating with RSSI values, to get better results.

To improve the direction estimation, the direction-specific techniques available in the latest BLE technology such as angle of arrival (AoA) and angle of departure

(AoD). They are advertised to measure the angle at which the signal is received or transmitted, ensuring a more accurate direction estimation. However, to apply AoA and AoD, devices need specialized hardware, making this technology is still not commonly supported by most mobile devices, resulting in the implementation not being easily accessible.

REFERENCES

Afaneh, M., 2022. *Bluetooth Low Energy (BLE): A Complete Guide.* [Online] Available at: https://novelbits.io/bluetooth-low-energy-ble-complete-guide/ [Accessed 14 April 2024].

Afaneh, M., 2023. *Bluetooth 5 speed: How to achieve maximum throughput for your BLE application.* [Online] Available at: https://novelbits.io/bluetooth-5-speed-maximum-throughput/ [Accessed 14 April 2024].

Afaneh, M., 2023. *Coded PHY: Bluetooth's Long-Range Feature.* [Online] Available at: https://novelbits.io/bluetooth-long-range-coded-phy/ [Accessed 15 April 2024].

Agarwal, T., 2020. *Different Types of Wireless Communication with Applications.* [Online]
Available at: https://www.elprocus.com/types-of-wireless-communication-applications/
[Accessed 13 April 2024].

Agarwal, T., 2021. *What is Bluetooth : Architecture & Its Working.* [Online] Available at: https://www.elprocus.com/how-does-bluetooth-work/ [Accessed 13 April 2024].

Argenox, 2020. *INTRODUCTION TO BLUETOOTH CLASSIC.* [Online] Available at: https://www.argenox.com/library/bluetooth-classic/introduction-to-bluetooth-classic/
[Accessed 14 April 2024].

Arponen, K. & Björkman, A., 2023. *Investigating the impact of physical layer transmission for Bluetooth LE Audio.*

BasuMallick, C., 2022. *What Is Bluetooth LE? Meaning, Working, Architecture, Uses, and Benefits.* [Online] Available at: https://www.spiceworks.com/tech/iot/articles/what-is-bluetooth-le/ [Accessed 14 April 2024].

Bluetooth®, n.d.. *Bluetooth® Wireless Technology.* [Online] Available at: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ [Accessed 13 April 2024].

CAMPORESOFT, 2019. *How Bluetooth Asset Tracking Can Enhance Tracking in the Workplace.* [Online] Available at: https://comparesoft.com/assets-tracking-software/bluetooth/#:~:text=What%20Is%20Bluetooth%20Asset%20Tracking%3F,communication%20to%20locate%20the%20object. [Accessed 13 April 2024].

electronicsnotes, n.d.. *Bluetooth Classic: how it works.* [Online] Available at: https://www.electronics-notes.com/articles/connectivity/bluetooth/bluetooth-classic-technology-operation.php [Accessed 14 April 2024].

Ezurio, n.d.. *What is Bluetooth Class?.* [Online] Available at: https://www.ezurio.com/support/faqs/what-bluetooth-class [Accessed 14 April 2024].

Fatima, R., 2023. *What is Asset Tracking? Best Practices, Benefits, and Examples.* [Online] Available at: https://ezo.io/ezofficeinventory/blog/asset-tracking/ [Accessed 14 April 2024].

FinalWire, 2015. *AIDA64 for Android Must-have app for hard core users of the most popular mobile platform around the world.* [Online] Available at: https://www.aida64.com/news/aida64-android#:~:text=AIDA64%20for%20Android%20features%20include,Fi%20and%20cellular%20network%20information [Accessed 17 April 2024].

Franklin, C. & Pollette, C., n.d.. *How Bluetooth Works.* [Online] Available at: https://electronics.howstuffworks.com/bluejacking.htm [Accessed 14 April 2024].

Gao, V., 2015. *Proximity and RSSI.* [Online] Available at: https://www.bluetooth.com/blog/proximity-and-rssi/ [Accessed 15 April 2024].

GeeksforGeeks, 2024. *What is Bluetooth?.* [Online] Available at: https://www.geeksforgeeks.org/bluetooth/ [Accessed 13 April 2024].

How To Electronics, 2023. *Bluetooth Low Energy Basics: Classic Bluetooth Vs. Bluetooth LE.* [Online] Available at: https://how2electronics.com/classic-bluetooth-vs-bluetooth-low-energy-comparison/ [Accessed 14 April 2024].

Intel, n.d.. *What Is Bluetooth® Technology?.* [Online] Available at: https://www.intel.com/content/www/us/en/products/docs/wireless/what-is-

bluetooth.html
[Accessed 13 April 2024].

JIMBLOM, n.d.. *Bluetooth Basics.* [Online]
Available at: https://learn.sparkfun.com/tutorials/bluetooth-basics/all
[Accessed 14 April 2024].

Jones, S., 2020. *5 WAYS BLUETOOTH 5 MAKES WIRELESS AUDIO BETTER.* [Online]
Available at: https://pro.harman.com/insights/harman-pro/5-ways-bluetooth-5-makes-wireless-audio-better/
[Accessed 13 April 2024].

Jonker, A., 2023. *What is asset tracking?.* [Online]
Available at: https://www.ibm.com/topics/asset-tracking
[Accessed 14 April 2024].

Li, M., 2022. *Understanding the Measures of Bluetooth RSSI.* [Online]
Available at: https://www.mokoblue.com/measures-of-bluetooth-rssi/#:~:text=Bluetooth%20RSSI%20(Received%20Signal%20Strength,device%20scans%20for%20Bluetooth%20devices.
[Accessed 15 April 2024].

Link Labs, 2021. *5 Fast Facts About Bluetooth Low Energy (LE) for Asset Tracking.* [Online]
Available at: https://www.link-labs.com/blog/5-fast-facts-about-bluetooth-low-energy-asset-tracking
[Accessed 14 April 2024].

MathWorks, n.d.. *Bluetooth LE Channel Selection Algorithms.* [Online]
Available at: https://www.mathworks.com/help/bluetooth/ug/bluetooth-le-channel-selection-algorithms.html
[Accessed 14 April 2024].

MathWorks, n.d.. *Comparison of Bluetooth BR/EDR and Bluetooth LE Specifications.* [Online]
Available at: https://www.mathworks.com/help/bluetooth/gs/comparison-of-bluetooth-bredr-and-bluetooth-le.html
[Accessed 14 April 2024].

Moozakis, C., 2023. *What is wireless communications? Everything you need to know.* [Online]
Available at: https://www.techtarget.com/searchmobilecomputing/definition/wireless
[Accessed 13 April 2024].

muRata, 2023. *Basic Knowledge of Wireless Communication: Wireless Mechanism (1).* [Online]
Available at: https://article.murata.com/en-sg/article/basics-of-wireless-communication-1
[Accessed 13 April 2024].

Nordic Semiconductor, 2021. *The Difference Between Classic Bluetooth and Bluetooth Low Energy.* [Online]
Available at: https://blog.nordicsemi.com/getconnected/the-difference-between-classic-bluetooth-and-bluetooth-low-energy
[Accessed 14 April 2024].

Nordic Semiconductor, n.d.. *nRF Connect for Mobile.* [Online]
Available at: https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-mobile
[Accessed 17 April 2024].

PROCTOR, B., 2023. *Bluetooth Vs. Bluetooth Low Energy: What's The Difference? [2023 Update].* [Online]
Available at: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy
[Accessed 14 April 2024].

Rittenberg, J. & Bottorff, C., 2022. *What Is Asset Tracking? Benefits & How It Works.* [Online]
Available at: https://www.forbes.com/advisor/business/what-is-asset-tracking/
[Accessed 14 April 2024].

Rushax, n.d.. *What is PuTTY? A Comprehensive Guide.* [Online]
Available at: https://rushax.com/what-is-putty-a-comprehensive-guide/
[Accessed 16 April 2024].

Sokolova, B., 2023. *Everything you need to know about BLE asset tracking.* [Online]
Available at: https://www.mapon.com/en/blog/2023/05/everything-to-know-about-ble-asset-tracking
[Accessed 14 April 2024].

STMicroelectronics, n.d.. *BlueNRG-LP, BlueNRG-LPS DK SW package.* [Online]
Available at: https://www.st.com/en/embedded-software/stsw-bnrglp-dk.html#overview
[Accessed 17 April 2024].

STMicroelectronics, n.d.. *Evaluation platform based on the BlueNRG-LPS system-on-chip.* [Online]
Available at: https://www.st.com/en/evaluation-tools/steval-idb012v1.html#overview
[Accessed 16 April 2024].

STMicroelectronics, n.d.. *Programmable Bluetooth Low Energy 5.3 Wireless SoC.* [Online]
Available at: https://www.st.com/en/wireless-connectivity/bluenrg-lps.html
[Accessed 17 April 2024].

STMicroelectronics, n.d.. *WiSE-Studio free IDE for Windows, Linux, MAC OS.* [Online]
Available at: https://www.st.com/en/embedded-software/stsw-wise-studio.html
[Accessed 16 April 2024].

Teja, R., 2024. *Wireless Communication: Introduction, Types and Applications.* [Online]
Available at: https://www.electronicshub.org/wireless-communication-introduction-types-applications/
[Accessed 13 April 2024].

ToolSense, 2023. *How to Get Your Assets Under Control With Bluetooth Equipment Tracking.* [Online]
Available at: https://toolsense.io/equipment-management/bluetooth-equipment-tracking/#:~:text=Bluetooth%20equipment%20tracking%20is%20a,volume%20design%20and%20security%20measures.
[Accessed 13 April 2024].

Woolley, M., 2017. *Exploring Bluetooth 5 - Going the Distance.* [Online]
Available at: https://www.bluetooth.com/blog/exploring-bluetooth-5-going-the-distance/
[Accessed 14 April 2024].

Boros, M., Kuffa, R. and Skýpalová, E., 2022, November. Testing the Communication Range of Ibeacon Technology. In 2022 6th International Conference on System Reliability and Safety (ICSRS) (pp. 400-403). IEEE.

Lyatuu, C.A., 2022. An analysis of bluetooth 5 in comparison to bluetooth 4.2. Europub Journal of Education Research, 3(1), pp.112-120.

Vanzin, L. and Oyamada, M.S., 2021, November. Calibration of BLE beacons and its impact on distance estimation using the log-distance path loss model. In 2021 10th Latin-American Symposium on Dependable Computing (LADC) (pp. 1-4). IEEE.

Harmanda, T.T., Priandana, K. and Hardhienata, M.K., 2020, February. Development of Localization Technique using Trilateration Algorithm for E-Puck2 Robot. In 2020 International Conference on Smart Technology and Applications (ICoSTA) (pp. 1-6). IEEE.

Venkatesh, R., Mittal, V. and Tammana, H., 2021, June. Indoor localization in BLE using mean and median filtered RSSI values. In 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 227-234). IEEE.

Hany, U., Akter, L. and Hossain, M.F., 2016, December. Moving averaging method of RSSI based distance estimation for wireless capsule localization. In 2016 International Conference on Medical Engineering, Health Informatics and Technology (MediTec) (pp. 1-5). IEEE.

Zhang, K., Zhang, Y. and Wan, S., 2016, October. Research of RSSI indoor ranging algorithm based on Gaussian-Kalman linear filtering. In 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) (pp. 1628-1632). IEEE.

Gani, M.O., OBrien, C., Ahamed, S.I. and Smith, R.O., 2013, July. RSSI based indoor localization for smartphone using fixed and mobile wireless node. In 2013 IEEE 37th Annual Computer Software and Applications Conference (pp. 110-117). IEEE.

APPENDICES

APPENDIX A: Code needed for BLE RC Long Range application

BLE_RC_LongRange_main.c

```
#include <stdio.h>
#include <string.h>
#include "rf_device_it.h"
#include "ble_const.h"
#include "bluenrg_lp_stack.h"
#include "rf_driver_hal_power_manager.h"
#include "rf_driver_hal_vtimer.h"
#include "bluenrg_lp_evb_com.h"
#include "rc.h"
#include "RemoteControl_config.h"
#include "bleplat.h"
#include "nvm_db.h"
#include "pka_manager.h"
#include "rng_manager.h"
#include "aes_manager.h"
#include "ble_controller.h"
#include "rf_driver_ll_rcc.h"
#include "rf_driver_ll_bus.h"
#include "rf_driver_ll_gpio.h"
#include "rf_driver_ll_system.h"
#include "rf_driver_ll_utils.h"
#include "rf_driver_hal_rcc.h"
#include "rf_driver_hal_gpio.h"

#ifndef DEBUG
#define DEBUG 1
#endif
```

BLE_RC_LongRange_main.c

```
#if DEBUG
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#else
#define PRINTF(...)
#endif

#define BLE_RC_VERSION_STRING "1.0.0"

NO_INIT(uint32_t dyn_alloc_a[DYNAMIC_MEMORY_SIZE>>2]);

#define LED2_PIN                    LL_GPIO_PIN_6
#define LED2_GPIO_PORT              GPIOB
#define LED2_GPIO_CLK_ENABLE()
LL_AHB_EnableClock(LL_AHB_PERIPH_GPIOB)

void ModulesInit(void)
{
  uint8_t ret;
  BLE_STACK_InitTypeDef BLE_STACK_InitParams =
BLE_STACK_INIT_PARAMETERS;

  LL_AHB_EnableClock(LL_AHB_PERIPH_PKA|LL_AHB_PERIPH_RNG);

  BLECNTR_InitGlobal();

  HAL_VTIMER_InitType VTIMER_InitStruct = {HS_STARTUP_TIME,
INITIAL_CALIBRATION, CALIBRATION_INTERVAL};
  HAL_VTIMER_Init(&VTIMER_InitStruct);

  BLEPLAT_Init();
  if (PKAMGR_Init() == PKAMGR_ERROR)
  {
     while(1);
  }
  if (RNGMGR_Init() != RNGMGR_SUCCESS)
  {
     while(1);
  }

  AESMGR_Init();

  ret = BLE_STACK_Init(&BLE_STACK_InitParams);
```

BLE_RC_LongRange_main.c

```
  if (ret != BLE_STATUS_SUCCESS) {
    printf("Error in BLE_STACK_Init() 0x%02x\r\n", ret);
    while(1);
  }
}

void ModulesTick(void)
{
  HAL_VTIMER_Tick();
  BLE_STACK_Tick();
  NVMDB_Tick();
}

void Configure_GPIO(void);
static void MX_GPIO_Init(void);

int main(void)
{
  uint8_t ret;
  WakeupSourceConfig_TypeDef wakeupIO;
  PowerSaveLevels stopLevel;

  if (SystemInit(SYSCLK_64M, BLE_SYSCLK_32M) != SUCCESS)
  {
    while(1);
  }
  BSP_IO_Init();

  BSP_PB_Init(BSP_PUSH1, BUTTON_MODE_EXTI);
#if CLIENT
  BSP_PB_Init(BSP_PUSH2, BUTTON_MODE_EXTI);
#endif

  BSP_COM_Init(BSP_COM_RxDataUserCb);

  BSP_LED_Init(BSP_LED1);
  BSP_LED_Init(BSP_LED2);
  BSP_LED_Init(BSP_LED3);

  ModulesInit();

  ret = RC_DeviceInit();
  if (ret != BLE_STATUS_SUCCESS) {
```

BLE_RC_LongRange_main.c

```
    PRINTF("RC_DeviceInit() failed: 0x%02x\r\n", ret);
    while(1);
  }

  PRINTF("BlueNRG-LP BLE Remote Control Application (version: %s)\r\n", 
BLE_RC_VERSION_STRING);

  wakeupIO.IO_Mask_High_polarity = WAKEUP_PA10;
  wakeupIO.IO_Mask_Low_polarity = 0;
  wakeupIO.RTC_enable = 0;
  wakeupIO.LPU_enable = 0;

  Configure_GPIO();
  MX_GPIO_Init();
  LL_GPIO_WriteOutputPin(LED2_GPIO_PORT, 
LED2_PIN,LL_GPIO_OUTPUT_LOW);

  while(1) {

		ModulesTick();

    APP_Tick();

#if ENABLE_LOW_POWER_MODE
    HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_STOP_WITH_TIMER, 
wakeupIO, &stopLevel);
#else
    HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_CPU_HALT, 
wakeupIO, &stopLevel);
#endif
  }
}

PowerSaveLevels App_PowerSaveLevel_Check(PowerSaveLevels level)
{
  if(BSP_COM_TxFifoNotEmpty() || BSP_COM_UARTBusy())
    return POWER_SAVE_LEVEL_RUNNING;

  return POWER_SAVE_LEVEL_STOP_NOTIMER;
}

void hci_hardware_error_event(uint8_t Hardware_Code)
{
```

BLE_RC_LongRange_main.c

```
  if (Hardware_Code <= 0x03)
  {
    NVIC_SystemReset();
  }
}

void aci_hal_fw_error_event(uint8_t FW_Error_Type,
                            uint8_t Data_Length,
                            uint8_t Data[])
{
  if (FW_Error_Type <= 0x03)
  {
    uint16_t connHandle;
    connHandle = LE_TO_HOST_16(Data);

    aci_gap_terminate(connHandle,
BLE_ERROR_TERMINATED_REMOTE_USER);
  }
}

#if ENABLE_LOW_POWER_MODE
void HAL_PWR_MNGR_WakeupIOCallback(uint32_t source)
{
  extern uint8_t button1_pressed;

  if (source & WAKEUP_PA10) {
    button1_pressed = TRUE;
  }
}
#endif


void Configure_GPIO(void)
{
  LED2_GPIO_CLK_ENABLE();

  LL_GPIO_SetPinMode(LED2_GPIO_PORT, LED2_PIN,
LL_GPIO_MODE_OUTPUT);
  LL_GPIO_SetPinOutputType(LED2_GPIO_PORT, LED2_PIN,
LL_GPIO_OUTPUT_PUSHPULL);
  LL_GPIO_SetPinSpeed(LED2_GPIO_PORT, LED2_PIN,
LL_GPIO_SPEED_FREQ_LOW);
  LL_GPIO_SetPinPull(LED2_GPIO_PORT, LED2_PIN, LL_GPIO_PULL_NO);
```

BLE_RC_LongRange_main.c

```
}

static void MX_GPIO_Init(void)
{
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  LED2_GPIO_CLK_ENABLE();

  LL_GPIO_SetOutputPin(LED2_GPIO_PORT, LED2_PIN);

  GPIO_InitStruct.Pin = LED2_PIN;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED2_GPIO_PORT, &GPIO_InitStruct);
}

#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
    while (1)
    {}
}
#endif
```

rc_server.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "rf_device_it.h"
#include "ble_const.h"
#include "bluenrg_lp_stack.h"
#include "rf_driver_hal_vtimer.h"
#include "bluenrg_lp_evb_com.h"
#include "rc.h"
#include "gatt_db.h"
#include "app_state.h"
#include "gap_profile.h"
#include "rf_driver_hal_gpio.h"
#include "clock.h"
```

rc_server.c

```
#ifndef SENSOR_EMULATION /* User Real sensor: lps22hh (pressure and
temperature) */
#include "bluenrg_lp_evb_config.h"
#endif

uint8_t button1_pressed = FALSE;

#define DEBOUNCE_TIMEOUT_MS     300

#ifndef DEBUG
#define DEBUG 1
#endif

#if DEBUG
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#else
#define PRINTF(...)
#endif

#define UPDATE_CONN_PARAM 1 //0

#define INT(x)    ((int)(x))
#define FRACTIONAL(x)  (x>0)? ((int) (((x) - INT(x)) * 10)) : ((int) ((INT(x) -
(x)) * 10))

#define BLE_RC_VERSION_STRING "1.0.0"

uint32_t start_time = 0;

static uint8_t sensor_update_timer_expired = FALSE;
volatile int app_flags = SET_CONNECTABLE;
volatile uint16_t connection_handle = 0;
static uint8_t debounce_timeout_occurred = TRUE;
static VTIMER_HandleType debounce_timer;

#if DISCONNECTION_TIMEOUT
static VTIMER_HandleType disconnectTimerHandle;
#endif
static VTIMER_HandleType sensorTimerHandle;
static VTIMER_HandleType advertisingLEDTimerHandle;
static uint8_t scan_resp_data[] = { 0x05,
AD_TYPE_COMPLETE_LOCAL_NAME, 'N', 'o', 'd', 'e' };
```

rc_server.c

```
#define TEMP_OFFSET 8
uint8_t adv_data[2][10] =
                    {
                                        { 0x02, AD_TYPE_FLAGS,
FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE
                                                            |
FLAG_BIT_BR_EDR_NOT_SUPPORTED, 0x06,

          AD_TYPE_MANUFACTURER_SPECIFIC_DATA, 0x30, 0x00, 0x05,
                                                            0xFF, 0xFF }, { 0x02,
AD_TYPE_FLAGS,

          FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE
                                                                                |
FLAG_BIT_BR_EDR_NOT_SUPPORTED, 0x06,

          AD_TYPE_MANUFACTURER_SPECIFIC_DATA, 0x30, 0x00, 0x05,
                                                            0xFF, 0xFF } };
uint8_t adv_data_index = 0;

static uint8_t phy = LE_1M_PHY;

#ifndef SENSOR_EMULATION
lps22hh_ctx_t pressureHandle;

#endif

volatile uint8_t request_free_fall_notify = FALSE;

BOOL sensor_board = FALSE;

#if UPDATE_CONN_PARAM
#define UPDATE_TIMER 2
static VTIMER_HandleType l2cap_TimerHandle;
int l2cap_request_sent = FALSE;
static uint8_t l2cap_req_timer_expired = FALSE;
#endif

#define SENSOR_TIMER 1
#define ACC_UPDATE_INTERVAL_MS 200

#ifndef SENSOR_ACCELEROMETER_EMULATION
```

rc_server.c

```
lsm6dsox_ctx_t inertialHandle;

#endif

#ifndef SENSOR_PRESSURE_TEMPERATURE_EMULATION

lps22hh_ctx_t pressureHandle;

#endif

void SensorUpdateTimeoutCB(void*);
void DisconnectTimeoutCB(void*);
void AdvertisingLEDTimeoutCB(void *param);

void DebounceTimeoutCB(void *param);

static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];

#ifndef SENSOR_ACCELEROMETER_EMULATION
void Init_Accelerometer(void) {
	uint8_t rst;

	inertialHandle.write_reg = BSP_SPI_Write;
	inertialHandle.read_reg = BSP_SPI_Read;

	BSP_SPI_Init();

	lsm6dsox_reset_set(&inertialHandle, PROPERTY_ENABLE);
	do {
		lsm6dsox_reset_get(&inertialHandle, &rst);
	} while (rst);

	lsm6dsox_block_data_update_set(&inertialHandle,
PROPERTY_ENABLE);

	lsm6dsox_xl_full_scale_set(&inertialHandle, LSM6DSOX_2g);
	lsm6dsox_gy_full_scale_set(&inertialHandle, LSM6DSOX_2000dps);

	lsm6dsox_xl_data_rate_set(&inertialHandle,
LSM6DSOX_XL_ODR_12Hz5);
	lsm6dsox_gy_data_rate_set(&inertialHandle,
LSM6DSOX_GY_ODR_12Hz5);
```

rc_server.c

```
}
#endif

#ifndef SENSOR_PRESSURE_TEMPERATURE_EMULATION
void Init_Pressure_Temperature_Sensor(void) {
	uint8_t rst;

	pressureHandle.write_reg = BSP_I2C_Write;
	pressureHandle.read_reg = BSP_I2C_Read;

	BSP_I2C_Init();

	lps22hh_reset_set(&pressureHandle, PROPERTY_ENABLE);
	do {
		lps22hh_reset_get(&pressureHandle, &rst);
	} while (rst);

	lps22hh_block_data_update_set(&pressureHandle,
PROPERTY_ENABLE);

	lps22hh_data_rate_set(&pressureHandle, LPS22HH_1_Hz_LOW_NOISE);

}
#endif


void Init_Temperature_Sensor(void) {
#ifndef SENSOR_EMULATION
	uint8_t rst;

	pressureHandle.write_reg = BSP_I2C_Write;
	pressureHandle.read_reg = BSP_I2C_Read;

	BSP_I2C_Init();

	lps22hh_reset_set(&pressureHandle, PROPERTY_ENABLE);
	do {
		lps22hh_reset_get(&pressureHandle, &rst);
	} while (rst);

	lps22hh_block_data_update_set(&pressureHandle,
PROPERTY_ENABLE);
```

rc_server.c

```
	lps22hh_data_rate_set(&pressureHandle, LPS22HH_1_Hz_LOW_NOISE);

#endif
}

void Update_Temperature(void) {
	float temperature_degC;
	uint8_t status = 1;

#ifdef SENSOR_EMULATION
  temperature_degC = 26 + ((uint64_t)rand()*15)/RAND_MAX;
#else
	axis1bit16_t data_raw_temperature;
	lps22hh_reg_t reg;

	lps22hh_read_reg(&pressureHandle, LPS22HH_STATUS, (uint8_t*)
&reg, 1);
	status = reg.status.t_da;

	if (status) {
		lps22hh_temperature_raw_get(&pressureHandle,
				data_raw_temperature.u8bit);
		temperature_degC = lps22hh_from_lsb_to_celsius(
				data_raw_temperature.i16bit);
	}
#endif
	if (status) {
		HOST_TO_LE_16(adv_data[adv_data_index]+TEMP_OFFSET,
				(int16_t )temperature_degC);
		aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA,
sizeof(adv_data[0]),
				adv_data[adv_data_index]);
		if (++adv_data_index == 2)
			adv_data_index = 0;
		PRINTF("Updated temperature: %d.%d C deg\n",
INT(temperature_degC),
				FRACTIONAL(temperature_degC));
	}
}

int8_t OUTPUT_POWER_LEVEL = 8;
```

rc_server.c

```
void TX_Update(uint8_t tx_power_level) {
        aci_hal_set_tx_power_level(1, tx_power_level);
}

uint8_t RC_DeviceInit(void) {
        uint8_t role = GAP_PERIPHERAL_ROLE;
        uint8_t bdaddr[] = { BD_ADDR_SLAVE };

        uint8_t device_name[] = { 'N', 'o', 'd', 'e' };
        uint8_t ret;
        uint16_t service_handle, dev_name_char_handle, appearance_char_handle;

        Init_Temperature_Sensor();

#ifndef SENSOR_ACCELEROMETER_EMULATION
        Init_Accelerometer();
#endif

#ifndef SENSOR_PRESSURE_TEMPERATURE_EMULATION
        Init_Pressure_Temperature_Sensor();
#endif

        ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
                        CONFIG_DATA_PUBADDR_LEN, bdaddr);
        if (ret != BLE_STATUS_SUCCESS) {
                PRINTF("aci_hal_write_config_data() failed: 0x%02x\r\n", ret);
                return ret;
        }

        TX_Update(OUTPUT_POWER_LEVEL);

        ret = aci_gatt_srv_init();
        if (ret) {
                PRINTF("aci_gatt_srv_init() failed: 0x%02x\r\n", ret);
                return ret;
        }

        ret = aci_gap_init(role, 0, 0x07, 0x00, &service_handle,
                        &dev_name_char_handle, &appearance_char_handle);
        if (ret) {
                PRINTF("aci_gap_Init() failed: 0x%02x\r\n", ret);
                return ret;
        }
```

rc_server.c

```
	ret = Gap_profile_set_dev_name(0, sizeof(device_name), device_name);
	if (ret) {
		PRINTF("Gap_profile_set_dev_name() failed: 0x%02x\r\n", ret);
		return ret;
	}

	ret = aci_gap_set_io_capability(IO_CAP_DISPLAY_ONLY);
	if (ret) {
		PRINTF("aci_gap_set_io_capability() failed: 0x%02x\r\n", ret);
		return ret;
	}

	ret = aci_gap_set_authentication_requirement(BONDING,
	MITM_PROTECTION_REQUIRED,
	SC_IS_NOT_SUPPORTED,
	KEYPRESS_IS_NOT_SUPPORTED, 7, 16,
	USE_FIXED_PIN_FOR_PAIRING, 123456);
	if (ret) {
		PRINTF("aci_gap_set_authentication_requirement failed:
0x%02x\r\n",
				ret);
		return ret;
	}

	ret = Add_RC_Service();
	if (ret != BLE_STATUS_SUCCESS) {
		PRINTF("ADD_RC_Service() failed: 0x%02x\r\n", ret);
		return ret;
	}

	ret = Add_Acc_Service();
	if (ret == BLE_STATUS_SUCCESS) {
		PRINTF("Acceleration service added successfully.\n");
	} else {
		PRINTF("Error while adding Acceleration service: 0x%02x\r\n",
ret);
		return ret;
	}

	ret = Add_Environmental_Sensor_Service();
	if (ret == BLE_STATUS_SUCCESS) {
		PRINTF("Environmental service added successfully.\n");
```

rc_server.c

```
	} else {
		PRINTF("Error while adding Environmental service: 0x%04x\r\n", 
ret);
		return ret;
	}

	ret = Add_Tx_Service();
	if (ret != BLE_STATUS_SUCCESS) {
		PRINTF("ADD_Tx_Service() failed: 0x%02x\r\n", ret);
		return ret;
	}

#if DISCONNECTION_TIMEOUT
  disconnectTimerHandle.callback = DisconnectTimeoutCB;
#endif
	debounce_timer.callback = DebounceTimeoutCB;
	sensorTimerHandle.callback = SensorUpdateTimeoutCB;
	advertisingLEDTimerHandle.callback = AdvertisingLEDTimeoutCB;

	ret = HAL_VTIMER_StartTimerMs(&sensorTimerHandle, 
ACC_UPDATE_INTERVAL_MS);
	if (ret != BLE_STATUS_SUCCESS) {
		PRINTF("HAL_VTIMER_StartTimerMs() failed; 0x%02x\r\n", 
ret);
		return ret;
	} else {
		sensor_update_timer_expired = FALSE;
	}

	return BLE_STATUS_SUCCESS;
}

#if UPDATE_CONN_PARAM
void l2cap_UpdateTimeoutCB(void *param) {
	l2cap_req_timer_expired = TRUE;
}
#endif

void Start_Advertising(void) {
	uint8_t ret;
	uint16_t adv_properties;
	Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
```

rc_server.c

```
	if (phy == LE_1M_PHY) {
		adv_properties = ADV_PROP_CONNECTABLE |
ADV_PROP_SCANNABLE
				| ADV_PROP_LEGACY;
	} else {
		adv_properties = ADV_PROP_CONNECTABLE;
	}

	uint8_t peer_address[6] = { BD_ADDR_MASTER };

	ret = aci_gap_set_advertising_configuration(0,
			GAP_MODE_GENERAL_DISCOVERABLE,
adv_properties,
			ADV_INTERVAL_MIN,
			ADV_INTERVAL_MAX,
			ADV_CH_ALL,
			PUBLIC_ADDR, peer_address,
			ADV_NO_WHITE_LIST_USE, 0,
			phy,
			0,
			phy,
			0,
			0);
	printf("Advertising configuration %02X\n", ret);

	ret = aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA,
			sizeof(adv_data[0]), adv_data[adv_data_index]);
	if (ret != BLE_STATUS_SUCCESS) {
		PRINTF("aci_gap_set_advertising_data() failed: 0x%02x\r\n", ret);
	}

	if (phy == LE_1M_PHY) {
		ret = aci_gap_set_scan_response_data(0, sizeof(scan_resp_data),
				scan_resp_data);
		if (ret != BLE_STATUS_SUCCESS) {
			PRINTF("hci_le_set_scan_response_data() failed:
0x%02x\r\n", ret);
		}
	}

	Update_Temperature();

	Advertising_Set_Parameters[0].Advertising_Handle = 0;
```

rc_server.c

```
        Advertising_Set_Parameters[0].Duration = 0;
        Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;

        ret = aci_gap_set_advertising_enable(ENABLE, 1,
Advertising_Set_Parameters);

        if (ret != BLE_STATUS_SUCCESS) {
                printf("Error in aci_gap_set_advertising_enable(): 0x%02x\r\n",
ret);
                return;
        } else
                printf("aci_gap_set_advertising_enable() --> SUCCESS\r\n");
        printf("Start Advertising phy %d\r\n", phy);

        HAL_VTIMER_StartTimerMs(&advertisingLEDTimerHandle,
                        ADVSCAN_LED_INTERVAL_MS);
}

void Stop_Advertising(void) {
        aci_gap_set_advertising_enable(DISABLE, 0, NULL);
        HAL_VTIMER_StopTimer(&advertisingLEDTimerHandle);
        BSP_LED_Off(ADVSCAN_CONN_LED);
}

void APP_Tick(void) {
        if (APP_FLAG(SET_CONNECTABLE)) {
                sensor_update_timer_expired = TRUE;
                Start_Advertising();
                APP_FLAG_CLEAR(SET_CONNECTABLE);
        }

        if (button1_pressed && debounce_timeout_occurred) {

                button1_pressed = FALSE;
                debounce_timeout_occurred = FALSE;
                HAL_VTIMER_StartTimerMs(&debounce_timer,
DEBOUNCE_TIMEOUT_MS);

                if (APP_FLAG(CONNECTED)) {
                        if (phy == LE_1M_PHY) {
                                PRINTF("Switch to LE CODED PHY\n");
                                hci_le_set_phy(connection_handle, 0,
LE_CODED_PHY_BIT,
```

rc_server.c

```
                                                LE_CODED_PHY_BIT, 2);
                        } else {
                                hci_le_set_phy(connection_handle, 0,
LE_1M_PHY_BIT,
                                                LE_1M_PHY_BIT, 2);
                                PRINTF("Switch to LE 1M PHY\n");
                        }
                } else {
                        Stop_Advertising();
                        if (phy == LE_1M_PHY) {
                                phy = LE_CODED_PHY;
                                BSP_LED_On(LONG_RANGE_LED);
                        } else {
                                phy = LE_1M_PHY;
                                BSP_LED_Off(LONG_RANGE_LED);
                        }
                        APP_FLAG_SET(SET_CONNECTABLE);
                }

        }

        if (sensor_update_timer_expired) {

                sensor_update_timer_expired = FALSE;
                HAL_VTIMER_StartTimerMs(&sensorTimerHandle,
TEMPERATURE_UPDATE_RATE);

                Update_Temperature();

                if (HAL_VTIMER_StartTimerMs(&sensorTimerHandle,
                                ACC_UPDATE_INTERVAL_MS) !=
BLE_STATUS_SUCCESS)
                        sensor_update_timer_expired = TRUE;

                if (APP_FLAG(CONNECTED)) {
                        AxesRaw_t x_axes, g_axes;
                        if (GetAccAxesRaw(&x_axes, &g_axes)) {
                                Acc_Update(&x_axes, &g_axes);
                        }

                        GetFreeFallStatus();
                }
        }
```

rc_server.c

```
	if (request_free_fall_notify == TRUE) {
		request_free_fall_notify = FALSE;
		Free_Fall_Notify();
	}
}

#if DISCONNECTION_TIMEOUT
void DisconnectTimeoutCB(void *param)
{
  aci_gap_terminate(connection_handle,0x13);
}
#endif

void DebounceTimeoutCB(void *param) {
	debounce_timeout_occurred = TRUE;
	button1_pressed = FALSE;
}

void SensorUpdateTimeoutCB(void *param) {
	sensor_update_timer_expired = TRUE;
}

void AdvertisingLEDTimeoutCB(void *param) {
	BSP_LED_Toggle(ADVSCAN_CONN_LED);
	HAL_VTIMER_StartTimerMs(&advertisingLEDTimerHandle,
				ADVSCAN_LED_INTERVAL_MS);
}

void hci_le_connection_complete_event(uint8_t Status,
		uint16_t Connection_Handle, uint8_t Role, uint8_t
Peer_Address_Type,
		uint8_t Peer_Address[6], uint16_t Conn_Interval, uint16_t
Conn_Latency,
		uint16_t Supervision_Timeout, uint8_t Master_Clock_Accuracy)

{
	if (Status != BLE_STATUS_SUCCESS)
		return;

	APP_FLAG_SET(CONNECTED);
	connection_handle = Connection_Handle;
```

rc_server.c

```
	printf("Connected\n");

	BSP_LED_On(ADVSCAN_CONN_LED);

	sensor_update_timer_expired = FALSE;
	HAL_VTIMER_StopTimer(&sensorTimerHandle);
#if DISCONNECTION_TIMEOUT
  HAL_VTIMER_StartTimerMs(&disconnectTimerHandle,
DISCONNECTION_TIMEOUT);
#endif
	HAL_VTIMER_StopTimer(&advertisingLEDTimerHandle);

	start_time = HAL_VTIMER_GetCurrentSysTime();
}

void hci_le_enhanced_connection_complete_event(uint8_t Status,
		uint16_t Connection_Handle, uint8_t Role, uint8_t
Peer_Address_Type,
		uint8_t Peer_Address[6], uint8_t
Local_Resolvable_Private_Address[6],
		uint8_t Peer_Resolvable_Private_Address[6], uint16_t
Conn_Interval,
		uint16_t Conn_Latency, uint16_t Supervision_Timeout,
		uint8_t Master_Clock_Accuracy) {

	hci_le_connection_complete_event(Status, Connection_Handle, Role,
			Peer_Address_Type, Peer_Address, Conn_Interval,
Conn_Latency,
			Supervision_Timeout, Master_Clock_Accuracy);
}

void hci_disconnection_complete_event(uint8_t Status,
		uint16_t Connection_Handle, uint8_t Reason) {
	APP_FLAG_CLEAR(CONNECTED);

	APP_FLAG_SET(SET_CONNECTABLE);

	printf("Disconnected\n");

	BSP_LED_Off(ADVSCAN_CONN_LED);

#if DISCONNECTION_TIMEOUT
```

rc_server.c

```
  HAL_VTIMER_StopTimer(&disconnectTimerHandle);
#endif

}

void aci_l2cap_connection_update_resp_event(uint16_t Connection_Handle,
		uint16_t Result) {
	if (Result) {
		PRINTF("> Connection parameters rejected.\n");
	} else {
		PRINTF("> Connection parameters accepted.\n");
	}
}

void hci_le_phy_update_complete_event(uint8_t Status,
		uint16_t Connection_Handle, uint8_t TX_PHY, uint8_t RX_PHY)
{
	PRINTF("PHY changed: %d %d\n", TX_PHY, RX_PHY);
	if (TX_PHY == LE_CODED_PHY && RX_PHY == LE_CODED_PHY)
{
		BSP_LED_On(LONG_RANGE_LED);
		phy = LE_CODED_PHY;
	} else if (TX_PHY == LE_1M_PHY && RX_PHY == LE_1M_PHY) {
		BSP_LED_Off(LONG_RANGE_LED);
		phy = LE_1M_PHY;
	} else {
		PRINTF("Unexpected\n");
		BSP_LED_Off(LONG_RANGE_LED);
	}
}
```

rc_client.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "rf_device_it.h"
#include "ble_const.h"
#include "bluenrg_lp_stack.h"
#include "rf_driver_hal_vtimer.h"
#include "bluenrg_lp_evb_com.h"
#include "bluenrg_lp_evb_button.h"
#include "rc.h"
```

rc_client.c

```
#include "gatt_db.h"
#include "app_state.h"
#include "gap_profile.h"
#include "gatt_profile.h"

uint8_t button1_pressed = FALSE, button2_pressed = FALSE;

#define DEBOUNCE_TIMEOUT_MS     300
#define STATS_INTERVAL_MS       10000

#ifndef DEBUG
#define DEBUG 1
#endif

#if DEBUG
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#else
#define PRINTF(...)
#endif

#define BLE_RC_VERSION_STRING "1.0.0"

volatile int app_flags = SET_CONNECTABLE;
volatile uint16_t connection_handle = 0;
static uint8_t debounce_timeout_occurred = TRUE;
static VTIMER_HandleType debounce_timer;
static VTIMER_HandleType scanningLEDTimerHandle;
static VTIMER_HandleType writeTimerHandle;
static VTIMER_HandleType statsTimerHandle;
static uint8_t phy = LE_1M_PHY;

static llc_conn_per_statistic_st per_statistic_ptr;

#if DO_NOT_USE_VTIMER_CB
  static uint32_t last_time;
#endif

void ScanningLEDTimeoutCB(void *param);
void writeTimeoutCB(void *param);
void statsTimeoutCB(void *param);
void toggle_LED(void);
```

rc_client.c

```
void DebounceTimeoutCB(void *param);

uint8_t RC_DeviceInit(void)
{

  uint8_t role = GAP_CENTRAL_ROLE;
  uint8_t bdaddr[] = {BD_ADDR_MASTER};
  uint8_t device_name[]={'N', 'o', 'd', 'e'};
  uint8_t ret;
  uint16_t service_handle, dev_name_char_handle, appearance_char_handle;

  ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
CONFIG_DATA_PUBADDR_LEN, bdaddr);
  if(ret != BLE_STATUS_SUCCESS) {
    PRINTF("aci_hal_write_config_data() failed: 0x%02x\r\n", ret);
    return ret;
  }

  aci_hal_set_tx_power_level(0, OUTPUT_POWER_LEVEL);

  ret = aci_gatt_srv_init();
  if(ret){
    PRINTF("aci_gatt_srv_init() failed: 0x%02x\r\n", ret);
    return ret;
  }

  ret = aci_gap_init(role, 0, 0x07,  0x00, &service_handle,
&dev_name_char_handle, &appearance_char_handle);
  if(ret){
    PRINTF("aci_gap_Init() failed: 0x%02x\r\n", ret);
    return ret;
  }

  ret = Gap_profile_set_dev_name(0, sizeof(device_name), device_name);
  if(ret){
    PRINTF("Gap_profile_set_dev_name() failed: 0x%02x\r\n", ret);
    return ret;
  }

  ret = aci_gap_set_io_capability(IO_CAP_DISPLAY_ONLY);
  if(ret){
    PRINTF("aci_gap_set_io_capability() failed: 0x%02x\r\n", ret);
    return ret;
```

rc_client.c

```
  }

  ret = aci_gap_set_authentication_requirement(BONDING,
                                               MITM_PROTECTION_REQUIRED,
                                               SC_IS_NOT_SUPPORTED,
                                               KEYPRESS_IS_NOT_SUPPORTED,
                                               7,
                                               16,
                                               USE_FIXED_PIN_FOR_PAIRING,
                                               123456);
   if(ret){
    PRINTF("aci_gap_set_authentication_requirement failed: 0x%02x\r\n", ret);
    return ret;
  }

  ret = aci_gap_set_scan_configuration(DUPLICATE_FILTER_ENABLED,
SCAN_ACCEPT_ALL, LE_1M_PHY_BIT, PASSIVE_SCAN,
SCAN_INTERVAL, SCAN_WINDOW);
  printf("Scan configuration for LE_1M_PHY: 0x%02X\n", ret);

  ret = aci_gap_set_scan_configuration(DUPLICATE_FILTER_ENABLED,
SCAN_ACCEPT_ALL, LE_CODED_PHY_BIT, PASSIVE_SCAN,
SCAN_INTERVAL, SCAN_WINDOW);
  printf("Scan configuration for LE_CODED_PHY: 0x%02X\n", ret);

  ret = aci_gap_set_connection_configuration(LE_1M_PHY_BIT,
CONN_INTERVAL_MIN, CONN_INTERVAL_MAX, 0,
SUPERVISION_TIMEOUT, CE_LENGTH, CE_LENGTH);
  printf("Connection configuration for LE_1M_PHY:  0x%02X\n", ret);

  ret = aci_gap_set_connection_configuration(LE_CODED_PHY_BIT,
CONN_INTERVAL_MIN, CONN_INTERVAL_MAX, 0,
SUPERVISION_TIMEOUT, CE_LENGTH, CE_LENGTH);
  printf("Connection configuration for LE_CODED_PHY: 0x%02X\n", ret);


  debounce_timer.callback = DebounceTimeoutCB;
  scanningLEDTimerHandle.callback = ScanningLEDTimeoutCB;
  writeTimerHandle.callback = writeTimeoutCB;
  statsTimerHandle.callback = statsTimeoutCB;

  return BLE_STATUS_SUCCESS;
}
```

rc_client.c

```
void Connect(void)
{
  tBleStatus ret;
  uint8_t phy_bit;

  tBDAddr bdaddr = {BD_ADDR_SLAVE};

  if(phy == LE_1M_PHY){
    phy_bit = LE_1M_PHY_BIT;
    BSP_LED_Off(LONG_RANGE_LED);
  }
  else {
    phy_bit = LE_CODED_PHY_BIT;
    BSP_LED_On(LONG_RANGE_LED);
  }

  ret = aci_gap_create_connection(phy_bit, PUBLIC_ADDR, bdaddr);

  if (ret != BLE_STATUS_SUCCESS)
  {
    printf("Error while starting connection: 0x%02x\r\n", ret);
    return;
  }

  printf("Connecting on phy %d...\n", phy);

  HAL_VTIMER_StartTimerMs(&scanningLEDTimerHandle,
ADVSCAN_LED_INTERVAL_MS);
}

void CancelConnect(void)
{

aci_gap_terminate_proc(GAP_DIRECT_CONNECTION_ESTABLISHMENT_P
ROC);
  HAL_VTIMER_StopTimer(&scanningLEDTimerHandle);
  BSP_LED_Off(ADVSCAN_CONN_LED);
}

void APP_Tick(void)
{
  if(APP_FLAG(SET_CONNECTABLE))
```

rc_client.c

```
  {
    Connect();
    APP_FLAG_CLEAR(SET_CONNECTABLE);
  }

  if(button1_pressed && debounce_timeout_occurred){

    button1_pressed = FALSE;
    debounce_timeout_occurred = FALSE;
    HAL_VTIMER_StartTimerMs(&debounce_timer,
DEBOUNCE_TIMEOUT_MS);

    if(APP_FLAG(CONNECTED)){
      if(phy == LE_1M_PHY){
        hci_le_set_phy(connection_handle, 0, LE_CODED_PHY_BIT,
LE_CODED_PHY_BIT, 2);
        PRINTF("Switch to LE CODED PHY\n");
      }
      else {
        hci_le_set_phy(connection_handle, 0, LE_1M_PHY_BIT,
LE_1M_PHY_BIT, 2);
        PRINTF("Switch to LE 1M PHY\n");
      }
    }
    else{
      CancelConnect();
      if(phy == LE_1M_PHY){
        phy = LE_CODED_PHY;
      }
      else {
        phy = LE_1M_PHY;
      }
    }
  }

#if AUTO_TOGGLE_LED == 0

  if(APP_FLAG(CONNECTED) && button2_pressed &&
debounce_timeout_occurred){

    button2_pressed = FALSE;
    debounce_timeout_occurred = FALSE;
```

rc_client.c

```
    HAL_VTIMER_StartTimerMs(&debounce_timer,
DEBOUNCE_TIMEOUT_MS);

    toggle_LED();

  }
#else

#if DO_NOT_USE_VTIMER_CB
  if(APP_FLAG(CONNECTED) &&
HAL_VTIMER_DiffSysTimeMs(HAL_VTIMER_GetCurrentSysTime(),
last_time) > DEBOUNCE_TIMEOUT_MS){
    toggle_LED();
    last_time = HAL_VTIMER_GetCurrentSysTime();
  }
#endif

#endif

}

void toggle_LED(void)
{
  tBleStatus ret;
  static uint8_t val = 1<<CONTROL_LED;

  if(val==0){
    BSP_LED_On(CONTROL_LED);
    val = 1<<CONTROL_LED;
  }
  else {
    BSP_LED_Off(CONTROL_LED);
    val = 0;
  }

  ret = aci_gatt_clt_write(connection_handle, 0x0012, 1, &val);

  PRINTF("Write 0x%02X to handle 0x0012, 0x%02X\n", val, ret);
}

void statsTimeoutCB(void *param)
{
```

rc_client.c

```
  uint8_t CRC_errs_perc =
(uint32_t)per_statistic_ptr.num_crc_err*100/per_statistic_ptr.num_pkts;
  uint8_t missed_pckt_perc =
(uint32_t)per_statistic_ptr.num_miss_evts*100/(per_statistic_ptr.num_pkts+per_st
atistic_ptr.num_miss_evts);
  uint8_t packet_err_rate =
(uint32_t)(per_statistic_ptr.num_crc_err+per_statistic_ptr.num_miss_evts)*100/(p
er_statistic_ptr.num_pkts+per_statistic_ptr.num_miss_evts);

  PRINTF("- CRC errs = %d%%\n- Missed packets = %d%%\n", CRC_errs_perc,
missed_pckt_perc);

  PRINTF("- PER = %d%%\n", packet_err_rate);

  llc_conn_per_statistic(connection_handle, &per_statistic_ptr);

  HAL_VTIMER_StartTimerMs(&statsTimerHandle, STATS_INTERVAL_MS);
}

void DebounceTimeoutCB(void *param)
{
  debounce_timeout_occurred = TRUE;
  button1_pressed = FALSE;
  button2_pressed = FALSE;
}


void ScanningLEDTimeoutCB(void *param)
{
  BSP_LED_Toggle(ADVSCAN_CONN_LED);
  HAL_VTIMER_StartTimerMs(&scanningLEDTimerHandle,
ADVSCAN_LED_INTERVAL_MS);
}

void writeTimeoutCB(void *param)
{
  toggle_LED();
  HAL_VTIMER_StartTimerMs(&writeTimerHandle, WRITE_INTERVAL_MS);
}

void hci_le_connection_complete_event(uint8_t Status,
                                      uint16_t Connection_Handle,
                                      uint8_t Role,
```

rc_client.c

```
                                        uint8_t Peer_Address_Type,
                                        uint8_t Peer_Address[6],
                                        uint16_t Conn_Interval,
                                        uint16_t Conn_Latency,
                                        uint16_t Supervision_Timeout,
                                        uint8_t Master_Clock_Accuracy)

{
  if(Status == BLE_ERROR_UNKNOWN_CONNECTION_ID)
  {
    APP_FLAG_SET(SET_CONNECTABLE);
    return;
  }
  else if(Status != 0)
    return;

  APP_FLAG_SET(CONNECTED);
  connection_handle = Connection_Handle;

  printf("Connected\n");

  BSP_LED_On(ADVSCAN_CONN_LED);

  HAL_VTIMER_StopTimer(&scanningLEDTimerHandle);

  llc_conn_per_statistic(connection_handle, &per_statistic_ptr);

  HAL_VTIMER_StartTimerMs(&statsTimerHandle, STATS_INTERVAL_MS);

#if AUTO_TOGGLE_LED

#if DO_NOT_USE_VTIMER_CB
    last_time = HAL_VTIMER_GetCurrentSysTime();
#else
    HAL_VTIMER_StartTimerMs(&writeTimerHandle,
WRITE_INTERVAL_MS);
#endif

#endif

}

void hci_le_enhanced_connection_complete_event(uint8_t Status,
```

rc_client.c

```
                                        uint16_t Connection_Handle,
                                        uint8_t Role,
                                        uint8_t Peer_Address_Type,
                                        uint8_t Peer_Address[6],
                                        uint8_t Local_Resolvable_Private_Address[6],
                                        uint8_t Peer_Resolvable_Private_Address[6],
                                        uint16_t Conn_Interval,
                                        uint16_t Conn_Latency,
                                        uint16_t Supervision_Timeout,
                                        uint8_t Master_Clock_Accuracy)
{

  hci_le_connection_complete_event(Status,
                              Connection_Handle,
                              Role,
                              Peer_Address_Type,
                              Peer_Address,
                              Conn_Interval,
                              Conn_Latency,
                              Supervision_Timeout,
                              Master_Clock_Accuracy);
}

void hci_disconnection_complete_event(uint8_t Status,
                                uint16_t Connection_Handle,
                                uint8_t Reason)
{
  APP_FLAG_CLEAR(CONNECTED);
  APP_FLAG_SET(SET_CONNECTABLE);

  printf("Disconnected\n");

  BSP_LED_Off(ADVSCAN_CONN_LED);
  BSP_LED_Off(CONTROL_LED);

  HAL_VTIMER_StopTimer(&statsTimerHandle);

#if AUTO_TOGGLE_LED
#if !DO_NOT_USE_VTIMER_CB
    HAL_VTIMER_StopTimer(&writeTimerHandle);
#endif
#endif
```

rc_client.c

```
}

void aci_l2cap_connection_update_resp_event(uint16_t Connection_Handle,
                                    uint16_t Result)
{
  if(Result) {
    PRINTF("> Connection parameters rejected.\n");
  } else  {
    PRINTF("> Connection parameters accepted.\n");
  }
}

void hci_le_phy_update_complete_event(uint8_t Status,
                                uint16_t Connection_Handle,
                                uint8_t TX_PHY,
                                uint8_t RX_PHY)
{
  PRINTF("PHY changed: %d %d\n", TX_PHY, RX_PHY);
  if(TX_PHY == LE_CODED_PHY && RX_PHY == LE_CODED_PHY){
    BSP_LED_On(LONG_RANGE_LED);
    phy = LE_CODED_PHY;
  }
  else if(TX_PHY == LE_1M_PHY && RX_PHY == LE_1M_PHY){
    BSP_LED_Off(LONG_RANGE_LED);
    phy = LE_1M_PHY;
  }
  else {
    PRINTF("Unexpected\n");
    BSP_LED_Off(LONG_RANGE_LED);
  }

  HAL_VTIMER_StopTimer(&statsTimerHandle);

  llc_conn_per_statistic(connection_handle, &per_statistic_ptr);
  HAL_VTIMER_StartTimerMs(&statsTimerHandle, STATS_INTERVAL_MS);

}
```

gatt_db.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

gatt_db.c

```
#include "ble_const.h"
#include "bluenrg_lp_stack.h"
#include "app_state.h"
#include "gatt_db.h"
#include "osal.h"
#include "bluenrg_lp_evb_config.h"
#include "rc.h"
#include "clock.h"
#include "gp_timer.h"
#include "osal.h"
#include "SensorDemo_config.h"
#include "gatt_profile.h"
#include "gap_profile.h"
#include "OTA_btl.h"
#include "rf_driver_hal_gpio.h"
#include "rf_driver_hal_gpio_ex.h"
#include "rf_driver_hal.h"
#include "rf_driver_hal_vtimer.h"

#ifndef DEBUG
#define DEBUG 1
#endif

#define ENABLE_SECURITY        0

#if DEBUG
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#else
#define PRINTF(...)
#endif

#define LED_NUMBER 3

#define LED2_PIN                          LL_GPIO_PIN_6
#define LED2_GPIO_PORT                         GPIOB

#define INT(x)    ((int)(x))
#define FRACTIONAL(x)  (x>0)? ((int) (((x) - INT(x)) * 10)) : ((int) ((INT(x) -
(x)) * 10))
```

gatt_db.c

```
#define RC_SRVC_UUID
0xba,0x5c,0xf7,0x93,0x3b,0x12,0xd3,0x89,0xe4,0x11,0x0d,0x9b,0x2e,0xf6,0x0e,
0xed
#define RC_CONTROL_POINT_UUID
0xba,0x5c,0xf7,0x93,0x3b,0x12,0xd3,0x89,0xe4,0x11,0x0d,0x9b,0x1a,0xfb,0x0e,
0xed

#define ACC_SERVICE_UUID
0x1b,0xc5,0xd5,0xa5,0x02,0x00,0xb4,0x9a,0xe1,0x11,0x3a,0xcf,0x80,0x6e,0x36,
0x02
#define FREE_FALL_UUID
0x1b,0xc5,0xd5,0xa5,0x02,0x00,0xfc,0x8f,0xe1,0x11,0x4a,0xcf,0xa0,0x78,0x3e,0
xe2
#define ACC_UUID
0x1b,0xc5,0xd5,0xa5,0x02,0x00,0x36,0xac,0xe1,0x11,0x4b,0xcf,0x80,0x1b,0x0a,
0x34

#define ENV_SENS_SERVICE_UUID
0x1b,0xc5,0xd5,0xa5,0x02,0x00,0xd0,0x82,0xe2,0x11,0x77,0xe4,0x40,0x1a,0x82
,0x42
#define TEMP_CHAR_UUID
0x1b,0xc5,0xd5,0xa5,0x02,0x00,0xe3,0xa9,0xe2,0x11,0x77,0xe4,0x20,0x55,0x2e
,0xa3
#define PRESS_CHAR_UUID
0x1b,0xc5,0xd5,0xa5,0x02,0x00,0x0b,0x84,0xe2,0x11,0x8b,0xe4,0x80,0xc4,0x20
,0xcd

#define PWR_SRVC_UUID
0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe0,0xf2,0x73,
0xd9
#define TX_CHR_UUID
0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe1,0xf2,0x73,
0xd9

uint16_t accServHandle, freeFallCharHandle, accCharHandle;
uint16_t envSensServHandle, tempCharHandle, pressCharHandle;
uint16_t pwrSensServHandle, txCharHandle;

extern uint16_t connection_handle;
extern BOOL sensor_board;

#ifndef SENSOR_ACCELEROMETER_EMULATION
extern lsm6dsox_ctx_t inertialHandle;
```

gatt_db.c

```
#endif

extern uint32_t start_time;

#ifndef SENSOR_PRESSURE_TEMPERATURE_EMULATION
extern lps22hh_ctx_t pressureHandle;
#endif

static const ble_gatt_chr_def_t rc_chars[] = {

 #if ENABLE_SECURITY
    {
        .properties = BLE_GATT_SRV_CHAR_PROP_READ |
BLE_GATT_SRV_CHAR_PROP_WRITE |
BLE_GATT_SRV_CHAR_PROP_WRITE_NO_RESP |
BLE_GATT_SRV_CHAR_PROP_AUTH_SIGN_WRITE,
        .permissions =
BLE_GATT_SRV_PERM_AUTHEN_READ|BLE_GATT_SRV_PERM_AUTH
EN_WRITE,
        .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
        .uuid = BLE_UUID_INIT_128(RC_CONTROL_POINT_UUID),
    },
#else
    {
        .properties = BLE_GATT_SRV_CHAR_PROP_READ |
BLE_GATT_SRV_CHAR_PROP_WRITE_NO_RESP|
BLE_GATT_SRV_CHAR_PROP_WRITE,
        .permissions = BLE_GATT_SRV_PERM_NONE,
        .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
        .uuid = BLE_UUID_INIT_128(RC_CONTROL_POINT_UUID),
    },
#endif
};

static const ble_gatt_srv_def_t rc_service = {
   .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
   .uuid = BLE_UUID_INIT_128(RC_SRVC_UUID),
   .chrs = {
       .chrs_p = (ble_gatt_chr_def_t *)rc_chars,
       .chr_count = 1U,
   },
};
```

gatt_db.c

```
BLE_GATT_SRV_CCCD_DECLARE(free_fall,
                NUM_LINKS,
                BLE_GATT_SRV_CCCD_PERM_DEFAULT,
                BLE_GATT_SRV_OP_MODIFIED_EVT_ENABLE_FLAG);

BLE_GATT_SRV_CCCD_DECLARE(accell,
                NUM_LINKS,
                BLE_GATT_SRV_CCCD_PERM_DEFAULT,
                BLE_GATT_SRV_OP_MODIFIED_EVT_ENABLE_FLAG);

static ble_gatt_chr_def_t acc_chars[] = {
   {
      .properties = BLE_GATT_SRV_CHAR_PROP_NOTIFY,
      .permissions = BLE_GATT_SRV_PERM_NONE,
      .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
      .uuid = BLE_UUID_INIT_128(FREE_FALL_UUID),
      .descrs = {
         .descrs_p = &BLE_GATT_SRV_CCCD_DEF_NAME(free_fall),
         .descr_count = 1,
      },
   },
   {
      .properties = BLE_GATT_SRV_CHAR_PROP_NOTIFY |
BLE_GATT_SRV_CHAR_PROP_READ,
      .permissions = BLE_GATT_SRV_PERM_NONE,
      .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
      .uuid = BLE_UUID_INIT_128(ACC_UUID),
      .descrs = {
         .descrs_p = &BLE_GATT_SRV_CCCD_DEF_NAME(accell),
         .descr_count = 1,
      },
   }
};

static ble_gatt_srv_def_t acc_service = {
  .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
  .uuid = BLE_UUID_INIT_128(ACC_SERVICE_UUID),
  .chrs = {
     .chrs_p = (ble_gatt_chr_def_t *)acc_chars,
     .chr_count = 2U,
  },
};
```

gatt_db.c

```
static charactFormat temp_char_format = {
    .format = FORMAT_SINT16,
    .exp = -1,
    .unit = UNIT_TEMP_CELSIUS,
    .name_space = 0,
    .desc = 0,
};

static charactFormat press_char_format = {
    .format = FORMAT_SINT24,
    .exp = -5,
    .unit = UNIT_PRESSURE_BAR,
    .name_space = 0,
    .desc = 0,
};

static ble_gatt_val_buffer_def_t env_descr_val_buffers[] =
{
    {
        .buffer_len = 7,
        .buffer_p = (uint8_t *)&temp_char_format,
    },
    {
        .buffer_len = 7,
        .buffer_p = (uint8_t *)&press_char_format,
    }
};

static ble_gatt_descr_def_t env_descrs[] =
{
    {
        .uuid = BLE_UUID_INIT_16(CHAR_FORMAT_DESC_UUID),
        .permissions = BLE_GATT_SRV_PERM_NONE,
        .properties = BLE_GATT_SRV_DESCR_PROP_READ,
        .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
        .val_buffer_p = &env_descr_val_buffers[0],
    },
    {
        .uuid = BLE_UUID_INIT_16(CHAR_FORMAT_DESC_UUID),
        .permissions = BLE_GATT_SRV_PERM_NONE,
        .properties = BLE_GATT_SRV_DESCR_PROP_READ,
        .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
        .val_buffer_p = &env_descr_val_buffers[1],
```

gatt_db.c

```
   },
};

static ble_gatt_chr_def_t env_chars[] = {
   {
      .properties = BLE_GATT_SRV_CHAR_PROP_READ,
      .permissions = BLE_GATT_SRV_PERM_NONE,
      .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
      .uuid = BLE_UUID_INIT_128(TEMP_CHAR_UUID),
      .descrs = {
         .descrs_p = &env_descrs[0],
         .descr_count = 1,
      },
   },
   {
      .properties = BLE_GATT_SRV_CHAR_PROP_READ,
      .permissions = BLE_GATT_SRV_PERM_NONE,
      .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
      .uuid = BLE_UUID_INIT_128(PRESS_CHAR_UUID),
      .descrs = {
         .descrs_p = &env_descrs[1],
         .descr_count = 1,
      },
   },
};

static ble_gatt_srv_def_t env_service = {
  .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
  .uuid = BLE_UUID_INIT_128(ENV_SENS_SERVICE_UUID),
  .chrs = {
     .chrs_p = (ble_gatt_chr_def_t *)env_chars,
     .chr_count = 2U,
  },
};

static const ble_gatt_chr_def_t pwr_chars[] = {
   {
      .properties = BLE_GATT_SRV_CHAR_PROP_READ|
BLE_GATT_SRV_CHAR_PROP_WRITE ,
      .permissions = BLE_GATT_SRV_PERM_NONE,
      .min_key_size = BLE_GATT_SRV_MAX_ENCRY_KEY_SIZE,
      .uuid = BLE_UUID_INIT_128(TX_CHR_UUID),
   },
```

gatt_db.c

```
};

static ble_gatt_srv_def_t pwr_service = {
   .type = BLE_GATT_SRV_PRIMARY_SRV_TYPE,
   .uuid = BLE_UUID_INIT_128(PWR_SRVC_UUID),
   .chrs = {
       .chrs_p = (ble_gatt_chr_def_t *)pwr_chars,
       .chr_count = 1U,
   },
};

uint8_t GetAccAxesRaw(AxesRaw_t * acceleration_data, AxesRaw_t *
gyro_data)
{
#ifndef SENSOR_ACCELEROMETER_EMULATION
  uint8_t tmp = 0;
  (void)tmp;

  axis3bit16_t data_raw_acceleration;
  axis3bit16_t data_raw_angular_rate;

  lsm6dsox_xl_flag_data_ready_get(&inertialHandle, &tmp);
  if(tmp)
  {
    memset(data_raw_acceleration.u8bit, 0x00, 3 * sizeof(int16_t));
    lsm6dsox_acceleration_raw_get(&inertialHandle, data_raw_acceleration.u8bit);
    acceleration_data->AXIS_X =
(int32_t)lsm6dsox_from_fs2_to_mg(data_raw_acceleration.i16bit[0]);
    acceleration_data->AXIS_Y =
(int32_t)lsm6dsox_from_fs2_to_mg(data_raw_acceleration.i16bit[1]);
    acceleration_data->AXIS_Z =
(int32_t)lsm6dsox_from_fs2_to_mg(data_raw_acceleration.i16bit[2]);
  }

  lsm6dsox_gy_flag_data_ready_get(&inertialHandle, &tmp);
  if(tmp) {
    lsm6dsox_angular_rate_raw_get(&inertialHandle,
data_raw_angular_rate.u8bit);
    gyro_data->AXIS_X  =
(int32_t)lsm6dsox_from_fs2000_to_mdps(data_raw_angular_rate.i16bit[0]);
    gyro_data->AXIS_Y  =
(int32_t)lsm6dsox_from_fs2000_to_mdps(data_raw_angular_rate.i16bit[1]);
```

gatt_db.c

```
    gyro_data->AXIS_Z  =
(int32_t)lsm6dsox_from_fs2000_to_mdps(data_raw_angular_rate.i16bit[2]);
  }
#else
  uint8_t tmp = 1;
  acceleration_data->AXIS_X = ((uint64_t)rand()) % X_OFFSET;
  acceleration_data->AXIS_Y = ((uint64_t)rand()) % Y_OFFSET;
  acceleration_data->AXIS_Z = ((uint64_t)rand()) % Z_OFFSET;

#endif

  return(tmp);
}

void GetFreeFallStatus(void)
{
#ifndef SENSOR_ACCELEROMETER_EMULATION

  lsm6dsox_all_sources_t all_source;

  lsm6dsox_all_sources_get(&inertialHandle, &all_source);
  if (all_source.wake_up_src.ff_ia)
  {
    request_free_fall_notify = TRUE;
  }
#endif
}

uint16_t controlPointHandle;
uint16_t controlPointHandle1;
uint8_t GetTemperature(float * temperature_degC);

tBleStatus Add_RC_Service(void)
{
  tBleStatus ret;

  ret = aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&rc_service);
  if (ret != BLE_STATUS_SUCCESS)
  {
    goto fail;
  }
  controlPointHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t
*)&rc_chars[0]);
```

gatt_db.c

```
  PRINTF("RC Service added.\nControl Point Char Handle %04X\n",
controlPointHandle);

  return BLE_STATUS_SUCCESS;

fail:
  PRINTF("Error while adding RC service.\n");
  return BLE_STATUS_ERROR ;
}

static uint8_t leds_value[2] = {0,0};

VTIMER_HandleType led2_blink_timer;

void LED2_BlinkTimerCallback(void *param)
{
    static uint8_t led_on = 0;

    if (led_on)
    {
        LL_GPIO_WriteOutputPin(LED2_GPIO_PORT, LED2_PIN,
LL_GPIO_OUTPUT_LOW);
        led_on = 0;
    }
    else
    {
        LL_GPIO_WriteOutputPin(LED2_GPIO_PORT, LED2_PIN,
LL_GPIO_OUTPUT_HIGH);
        led_on = 1;
    }

    HAL_VTIMER_StartTimerMs(&led2_blink_timer, 500);
}

void Attribute_Modified_CB(uint16_t handle, uint8_t data_length, uint8_t
*att_data)
{
    if (handle == controlPointHandle + 1)
    {
        leds_value[0] = att_data[0];
        leds_value[1] = att_data[1];
```

gatt_db.c

```
      PRINTF("\nAttribute modified: leds_value[0] = 0x%02X, leds_value[1] =
0x%02X\n\n", leds_value[0], leds_value[1]);

      if (att_data[0] & (1 << CONTROL_LED))
      {
         BSP_LED_On(CONTROL_LED);

         led2_blink_timer.callback = LED2_BlinkTimerCallback;
         HAL_VTIMER_StartTimerMs(&led2_blink_timer, 500);
      }
      else
      {
         BSP_LED_Off(CONTROL_LED);
         LL_GPIO_WriteOutputPin(LED2_GPIO_PORT, LED2_PIN,
LL_GPIO_OUTPUT_LOW);

         HAL_VTIMER_StopTimer(&led2_blink_timer);
      }
   }

   else if (handle == txCharHandle + 1)
   {
      if (data_length == 1)
      {
         int8_t new_power_level = (int8_t)att_data[0];

         OUTPUT_POWER_LEVEL = new_power_level;

         TX_Update((uint8_t)OUTPUT_POWER_LEVEL);

         PRINTF("Updated TX Power Level: %d dBm\n",
OUTPUT_POWER_LEVEL);
      }
      else
      {
         PRINTF("Error: Invalid data length for TX power level update.\n");
      }
   }
}

tBleStatus Add_Acc_Service(void)
{
  tBleStatus ret;
```

gatt_db.c

```
  ret = aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&acc_service);
  if (ret != BLE_STATUS_SUCCESS)
  {
    goto fail;
  }

  accServHandle = aci_gatt_srv_get_service_handle(&acc_service);
  freeFallCharHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t
*)&acc_chars[0]);
  accCharHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t
*)&acc_chars[1]);
  PRINTF("Service ACC added. Handle 0x%04X, Free fall Charac handle:
0x%04X, Acc Charac handle: 0x%04X\n",
        accServHandle, freeFallCharHandle, accCharHandle);

  return BLE_STATUS_SUCCESS;

  fail:
     PRINTF("Error while adding ACC service at step: 0x%04X\n", __LINE__);
     return BLE_STATUS_ERROR;
}

tBleStatus Free_Fall_Notify(void)
{
  uint8_t val;
  tBleStatus ret;

  val = 0x01;

  ret = aci_gatt_srv_notify(connection_handle, freeFallCharHandle + 1, 0, 1,
&val);
  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating Free Fall characteristic: 0x%02X\n",ret) ;
    return BLE_STATUS_ERROR ;
  }
  return BLE_STATUS_SUCCESS;
}

tBleStatus Acc_Update(AxesRaw_t *x_axes, AxesRaw_t *g_axes)
{
  uint8_t buff[2+2*6];
  tBleStatus ret;
```

gatt_db.c

```
  uint32_t time =
HAL_VTIMER_DiffSysTimeMs(HAL_VTIMER_GetCurrentSysTime(),
start_time);

  HOST_TO_LE_16(buff,time);

  HOST_TO_LE_16(buff+2,-x_axes->AXIS_Y);
  HOST_TO_LE_16(buff+4, x_axes->AXIS_X);
  HOST_TO_LE_16(buff+6,-x_axes->AXIS_Z);

  HOST_TO_LE_16(buff+8,g_axes->AXIS_Y);
  HOST_TO_LE_16(buff+10,g_axes->AXIS_X);
  HOST_TO_LE_16(buff+12,g_axes->AXIS_Z);

  ret = aci_gatt_srv_notify(connection_handle, accCharHandle + 1, 0,  2+2*6,
buff);
  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating Acceleration characteristic: 0x%02X\n",ret) ;
    return BLE_STATUS_ERROR ;
  }

  return BLE_STATUS_SUCCESS;
}

tBleStatus Add_Environmental_Sensor_Service(void)
{
  tBleStatus ret;

  ret = aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&env_service);
  if (ret != BLE_STATUS_SUCCESS)
  {
    goto fail;
  }

  envSensServHandle = aci_gatt_srv_get_service_handle(&env_service);
  tempCharHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t
*)&env_chars[0]);
  pressCharHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t
*)&env_chars[1]);

  PRINTF("Service ENV_SENS added. Handle 0x%04X, TEMP Charac handle:
0x%04X, PRESS Charac handle: 0x%04X\n",envSensServHandle,
tempCharHandle, pressCharHandle);
```

gatt_db.c

```
  return BLE_STATUS_SUCCESS;

  fail:
      PRINTF("Error while adding ENV_SENS service at step: 0x%04X\n",
__LINE__);
      return BLE_STATUS_ERROR;
}

tBleStatus Add_Tx_Service(void)
{
  tBleStatus ret;

  ret = aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&pwr_service);
  if (ret != BLE_STATUS_SUCCESS)
  {
    goto fail;
  }

  pwrSensServHandle = aci_gatt_srv_get_service_handle(&pwr_service);
  txCharHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t
*)&pwr_chars[0]);

  PRINTF("Service Tx added. Handle 0x%04X, Tx Charac handle: 0x%04X, Rx
Charac handle: 0x%04X\n",pwrSensServHandle, txCharHandle);
  return BLE_STATUS_SUCCESS;

  fail:
      PRINTF("Error while adding Tx service at step: 0x%04X\n", __LINE__);
      return BLE_STATUS_ERROR;
}

tBleStatus Temp_Update(int16_t temp)
{
  tBleStatus ret;

  ret = aci_gatt_srv_notify(connection_handle, tempCharHandle + 1, 0, 2, (uint8_t
*)&temp);
  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating TEMP characteristic: 0x%02X\n",ret);
    return BLE_STATUS_ERROR ;
  }

  return BLE_STATUS_SUCCESS;
```

gatt_db.c

```
}

tBleStatus Press_Update(int32_t press)
{
  tBleStatus ret;

  ret = aci_gatt_srv_notify(connection_handle, pressCharHandle + 1, 0, 3, (uint8_t
*)&press);
  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating Pressure characteristic: 0x%02X\n",ret);
    return BLE_STATUS_ERROR;
  }

  return BLE_STATUS_SUCCESS;
}

#ifndef SENSOR_PRESSURE_TEMPERATURE_EMULATION

uint8_t GetPressure(float * pressure_hPa)
{
   axis1bit32_t data_raw_pressure;
   lps22hh_reg_t reg;

   data_raw_pressure.i32bit = 0;

   lps22hh_read_reg(&pressureHandle, LPS22HH_STATUS, (uint8_t *)&reg, 1);
   if (reg.status.p_da)
   {
     lps22hh_pressure_raw_get(&pressureHandle, data_raw_pressure.u8bit);
     *pressure_hPa = lps22hh_from_lsb_to_hpa(data_raw_pressure.i32bit);
   }
   return (reg.status.p_da);
}

uint8_t GetTemperature(float * temperature_degC)
{
    axis1bit16_t data_raw_temperature;
    lps22hh_reg_t reg;

    lps22hh_read_reg(&pressureHandle, LPS22HH_STATUS, (uint8_t *)&reg, 1);
    if (reg.status.t_da)
    {
```

gatt_db.c

```
      lps22hh_temperature_raw_get(&pressureHandle,
data_raw_temperature.u8bit);
      *temperature_degC =
lps22hh_from_lsb_to_celsius(data_raw_temperature.i16bit);
    }
    return (reg.status.t_da);
}

#endif

void aci_gatt_srv_write_event(uint16_t Connection_Handle, uint8_t
Resp_Needed, uint16_t Attribute_Handle, uint16_t Data_Length, uint8_t Data[])
{
    uint8_t att_error = BLE_ATT_ERR_NONE;

    Attribute_Modified_CB(Attribute_Handle, Data_Length, Data);

    if (Resp_Needed == 1U)
    {
        aci_gatt_srv_resp(Connection_Handle, 0, att_error, 0, NULL);
    }
}

void aci_gatt_srv_read_event(uint16_t Connection_Handle, uint16_t
Attribute_Handle, uint16_t Data_Offset)
{
  uint8_t att_err;
  uint8_t buff[6], *data_p;
  uint16_t data_len;
  int16_t temp_val;
  int32_t press_val;
  float fdata;

#ifdef SENSOR_PRESSURE_TEMPERATURE_EMULATION
  uint16_t udata;
#endif

  att_err = BLE_ATT_ERR_NONE;
  if(Attribute_Handle == controlPointHandle + 1)
  {
    aci_gatt_srv_resp(Connection_Handle, Attribute_Handle, att_err, 2,
leds_value);
    PRINTF ("RC operation was accessed.\n");
```

gatt_db.c

```
    PRINTF("Read leds_value: 0x%02X\n", leds_value[0]);

    if (leds_value[0] == 0x04)
    {
      PRINTF("Buzzer is on.\n");
    }
    else
    {
      PRINTF("Buzzer is off.\n\n");
    }
  }

  else if(Attribute_Handle == accCharHandle + 1)
  {
    AxesRaw_t x_axes, g_axes;
    if (GetAccAxesRaw(&x_axes, &g_axes))
    {
            uint32_t time =
HAL_VTIMER_DiffSysTimeMs(HAL_VTIMER_GetCurrentSysTime(),
start_time);

            HOST_TO_LE_16(buff, time);

            PRINTF("Accelerometer Data - X: %d, Y: %d, Z: %d\n",
x_axes.AXIS_X, x_axes.AXIS_Y, x_axes.AXIS_Z);

            HOST_TO_LE_16(buff+2, -x_axes.AXIS_Y);
            HOST_TO_LE_16(buff+4, x_axes.AXIS_X);
            HOST_TO_LE_16(buff+6, -x_axes.AXIS_Z);

            PRINTF("Gyroscope Data - X: %d, Y: %d, Z: %d\n", g_axes.AXIS_X,
g_axes.AXIS_Y, g_axes.AXIS_Z);

            HOST_TO_LE_16(buff+8, g_axes.AXIS_Y);
            HOST_TO_LE_16(buff+10, g_axes.AXIS_X);
            HOST_TO_LE_16(buff+12, g_axes.AXIS_Z);

            data_p = buff;
            data_len = 2 + 2 * 6;

            PRINTF("Buffer content: ");
            for (int i = 0; i < data_len; i++) {
```

gatt_db.c

```
          PRINTF("%02X ", buff[i]);
        }
        PRINTF("\n");

    }
    else
    {
      att_err = BLE_ATT_ERR_APPL_MIN;
      PRINTF("Error in reading ACC values\n");
    }
    aci_gatt_srv_resp(Connection_Handle, Attribute_Handle, att_err, data_len,
data_p);
  }

  else if (Attribute_Handle == tempCharHandle + 1)
  {
  #ifdef SENSOR_PRESSURE_TEMPERATURE_EMULATION
    fdata = 27 + ((uint64_t)rand() * 15) / RAND_MAX;
  #else
    if (GetTemperature(&fdata) != 1)
    {
      att_err = BLE_ATT_ERR_APPL_MIN;
    }
  #endif
    data_len = 2;
    temp_val = (int16_t)(fdata * 10);
    data_p = (uint8_t *)&temp_val;

    PRINTF("Temperature Value: %d.%d C deg\n", INT(fdata),
FRACTIONAL(fdata));
    aci_gatt_srv_resp(Connection_Handle, Attribute_Handle, att_err, data_len,
data_p);
  }
  else if (Attribute_Handle == pressCharHandle + 1)
  {
  #ifdef SENSOR_PRESSURE_TEMPERATURE_EMULATION
    fdata = 1000 + ((uint64_t)rand() * 1000) / RAND_MAX;
  #else
    if (GetPressure(&fdata) != 1)
    {
      att_err = BLE_ATT_ERR_APPL_MIN;
    }
  #endif
```

gatt_db.c

```
      data_len = 3;
      press_val = (int32_t)(fdata * 100);
      data_p = (uint8_t *)&press_val;

      PRINTF("Pressure Value: %d.%d hpa\n", INT(fdata), FRACTIONAL(fdata));
      aci_gatt_srv_resp(Connection_Handle, Attribute_Handle, att_err, data_len,
data_p);
  }
  else if (Attribute_Handle == txCharHandle + 1)
  {
      int8_t txPowerLevel = OUTPUT_POWER_LEVEL;
      data_len = 1;
      data_p = (uint8_t *)&txPowerLevel;

      PRINTF("TX Power Level: %d dBm\n", txPowerLevel);
      aci_gatt_srv_resp(Connection_Handle, Attribute_Handle, att_err, data_len,
data_p);
  }
}
```

APPENDIX B: Code needed for BLE Beacons application

BLE_Beacon_main.c

```
#include <stdio.h>
#include <string.h>
#include "rf_device_it.h"
#include "ble_const.h"
#include "bluenrg_lp_stack.h"
#include "Beacon_config.h"
#include "OTA_btl.h"
#include "rf_driver_hal_power_manager.h"
#include "rf_driver_hal_vtimer.h"
#include "bluenrg_lp_evb_com.h"
#include "bleplat.h"
#include "nvm_db.h"
#include "pka_manager.h"
#include "rng_manager.h"
#include "aes_manager.h"
#include "ble_controller.h"
#include "miscutil.h"

#define BLE_BEACON_VERSION_STRING "2.1"

#define LEGACY_ADV_INTERVAL     160

#if PERIODIC_ADV

#define EXT_ADV_INTERVAL        1600
#else
#define EXT_ADV_INTERVAL        160
#endif

#define PERIODIC_ADV_INTERVAL   240

#define DEVICE_NAME_IN_ADV 1

#define EXT_ADV_PHY LE_CODED_PHY


#define CTE_LENGTH              20
```

BLE_Beacon_main.c

```
#define CTE_TYPE                 0
#define CTE_COUNT                  4
#define ANTENNA_IDS                {0,1}

#ifdef DEBUG
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#else
#define PRINTF(...)
#endif

#define SHORT_ADV_DATA_LENGTH    (27 + 3)

static uint8_t adv_data[] = {
  0x02, 0x01, 0x06,
  26,
  AD_TYPE_MANUFACTURER_SPECIFIC_DATA,
  0x4C, 0x00,
  0x02,
  0x15,
  0xE2, 0x0A, 0x39, 0xF4, 0x73, 0xF5, 0x4B, 0xC4,
  0xA1, 0x2F, 0x17, 0xD1, 0xAD, 0x07, 0xA9, 0x61,
  0x00, 0x05,
  0x00, 0x07,
  (uint8_t)-56,
#if DEVICE_NAME_IN_ADV
  15,
  0x09,'E','x','t','e','n','d','e','d','B','e','a','c','o','n'
#endif
};

NO_INIT(uint32_t dyn_alloc_a[DYNAMIC_MEMORY_SIZE>>2]);

void ModulesInit(void)
{
  uint8_t ret;
  BLE_STACK_InitTypeDef BLE_STACK_InitParams = 
BLE_STACK_INIT_PARAMETERS;

  LL_AHB_EnableClock(LL_AHB_PERIPH_PKA|LL_AHB_PERIPH_RNG);

  BLECNTR_InitGlobal();
```

BLE_Beacon_main.c

```
  HAL_VTIMER_InitType VTIMER_InitStruct = {HS_STARTUP_TIME,
INITIAL_CALIBRATION, CALIBRATION_INTERVAL};
  HAL_VTIMER_Init(&VTIMER_InitStruct);

  BLEPLAT_Init();
  if (PKAMGR_Init() == PKAMGR_ERROR)
  {
      while(1);
  }
  if (RNGMGR_Init() != RNGMGR_SUCCESS)
  {
      while(1);
  }

  AESMGR_Init();

  ret = BLE_STACK_Init(&BLE_STACK_InitParams);
  if (ret != BLE_STATUS_SUCCESS) {
    printf("Error in BLE_STACK_Init() 0x%02x\r\n", ret);
    while(1);
  }
}

void ModulesTick(void)
{
  HAL_VTIMER_Tick();
  BLE_STACK_Tick();
  NVMDB_Tick();
}

void Device_Init(void)
{
  uint8_t ret;
  uint16_t service_handle;
  uint16_t dev_name_char_handle;
  uint16_t appearance_char_handle;
  uint8_t address[CONFIG_DATA_PUBADDR_LEN] =
{0x66,0x77,0x88,0xE1,0x80,0x02};

#if (CTE_TYPE == 1) || (CTE_TYPE  == 2)
  aci_hal_set_antenna_switch_parameters(0x06,
                                        1,
```

BLE_Beacon_main.c

```
                                        0x00,
                                        1);
#endif

  aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
CONFIG_DATA_PUBADDR_LEN, address);

  ret = aci_hal_set_tx_power_level(1, 8);
  if(ret != 0) {
    PRINTF ("Error in aci_hal_set_tx_power_level() 0x%04xr\n", ret);
    while(1);
  }

  ret = aci_gap_init(GAP_BROADCASTER_ROLE, 0x00, 0x08,
PUBLIC_ADDR, &service_handle, &dev_name_char_handle,
&appearance_char_handle);
  if (ret != 0)
  {
    PRINTF ("Error in aci_gap_init() 0x%04x\r\n", ret);
  }
  else
  {
    PRINTF ("aci_gap_init() --> SUCCESS\r\n");
  }
}

static void Start_Beaconing(void)
{
  uint8_t ret;
  Advertising_Set_Parameters_t Advertising_Set_Parameters[2];
  uint8_t adv_sets = 0;

#ifdef EXTENDED_ADV
  ret = aci_gap_set_advertising_configuration(1,
GAP_MODE_GENERAL_DISCOVERABLE,
                                              ADV_PROP_NONE,
                                              EXT_ADV_INTERVAL, EXT_ADV_INTERVAL,
                                              ADV_CH_ALL,
                                              0,NULL,
                                              ADV_NO_WHITE_LIST_USE,
                                              0,
                                              LE_CODED_PHY,
                                              0,
```

BLE_Beacon_main.c

```
                                    LE_CODED_PHY,
                                    0,
                                    0);
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("Error in aci_gap_set_advertising_configuration() 0x%02x\r\n", ret);
    return;
  }

  ret = aci_gap_set_advertising_data(1, ADV_COMPLETE_DATA,
sizeof(adv_data), adv_data);
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("Error in aci_gap_set_advertising_data() 0x%02x\r\n", ret);
    return;
  }

  printf("Extended advertising configured\n");

  Advertising_Set_Parameters[adv_sets].Advertising_Handle = 1;
  Advertising_Set_Parameters[adv_sets].Duration = 0;
  Advertising_Set_Parameters[adv_sets].Max_Extended_Advertising_Events = 0;

  adv_sets++;

#endif

#ifdef PERIODIC_ADV
  ret = aci_gap_set_periodic_advertising_configuration(1,
PERIODIC_ADV_INTERVAL, PERIODIC_ADV_INTERVAL,  0 );
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("Error in aci_gap_set_periodic_advertising_configuration()
0x%02x\r\n", ret);
    return;
  }

  ret = aci_gap_set_periodic_advertising_data(1,
SHORT_ADV_DATA_LENGTH, adv_data);
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("Error in aci_gap_set_periodic_advertising_data() 0x%02x\r\n", ret);
    return;
```

BLE_Beacon_main.c

```
  }

#if CTE_TAG
  uint8_t antenna_ids[] = ANTENNA_IDS;

  ret = hci_le_set_connectionless_cte_transmit_parameters(1, CTE_LENGTH,
CTE_TYPE, CTE_COUNT, sizeof(antenna_ids), antenna_ids);
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("Error in hci_le_set_connectionless_cte_transmit_parameters()
0x%02x\r\n", ret);
    return;
  }

  ret = hci_le_set_connectionless_cte_transmit_enable(1, 1);

  printf("CTE configured\n");
#endif

  ret = aci_gap_set_periodic_advertising_enable(ENABLE, 1);
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("Error in aci_gap_set_periodic_advertising_enable() 0x%02x\r\n",
ret);
    return;
  }

  printf("Periodic advertising configured\n");

#endif

  ret = aci_gap_set_advertising_enable(ENABLE, adv_sets,
Advertising_Set_Parameters);
  if (ret != BLE_STATUS_SUCCESS)
  {
    PRINTF ("Error in aci_gap_set_advertising_enable() 0x%02x\r\n", ret);
    return;
  }

  printf("Advertising started\n");

}
```

BLE_Beacon_main.c

```
int main(void)
{
  WakeupSourceConfig_TypeDef wakeupIO;
  PowerSaveLevels stopLevel;

  if (SystemInit(SYSCLK_64M, BLE_SYSCLK_32M) != SUCCESS)
  {
    while(1);
  }
  BSP_IO_Init();
  BSP_COM_Init(NULL);
  ModulesInit();
  Device_Init();
#if CONFIG_OTA_USE_SERVICE_MANAGER
  BSP_PB_Init(USER_BUTTON, BUTTON_MODE_GPIO);
#endif

  printf("BlueNRG-X Bluetooth LE Beacon Application (version: %s)",
BLE_BEACON_VERSION_STRING);
#if defined(CTE_TAG)
  printf(", with CTE tag\r\n");
#elif defined(PERIODIC_ADV)
  printf(", with Periodic advertising\r\n");
#elif defined (EXTENDED_ADV)
  printf(", with Extended advertising (DEVICE_NAME_IN_ADV: %d)\r\n",
DEVICE_NAME_IN_ADV);
#else
  printf("\r\n");
#endif

  Start_Beaconing();

  wakeupIO.IO_Mask_High_polarity = 0;
  wakeupIO.IO_Mask_Low_polarity = 0;
  wakeupIO.RTC_enable = 0;
  wakeupIO.LPU_enable = 0;

  while(1)
  {
    ModulesTick();

    HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_STOP_NOTIMER,
wakeupIO, &stopLevel);
```

BLE_Beacon_main.c

```
#if CONFIG_OTA_USE_SERVICE_MANAGER
    if (BSP_PB_GetState(USER_BUTTON) == SET)
    {
      OTA_Jump_To_Service_Manager_Application();
    }
#endif
  }
}

PowerSaveLevels App_PowerSaveLevel_Check(PowerSaveLevels level)
{
  if(BSP_COM_TxFifoNotEmpty() || BSP_COM_UARTBusy())
    return POWER_SAVE_LEVEL_RUNNING;

  return POWER_SAVE_LEVEL_STOP_NOTIMER;
}

void hci_hardware_error_event(uint8_t Hardware_Code)
{
  if (Hardware_Code <= 0x03)
  {
    NVIC_SystemReset();
  }
}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t* file, uint32_t line)
{
  while (1)
  {
  }
}

#endif
```

APPENDIX C: Codes needed for BLE Android application

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?> <!-- default setting -->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   >

   <uses-permission android:name="android.permission.BLUETOOTH"
      android:maxSdkVersion="30" />
   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
      android:maxSdkVersion="30" />

   <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"
      android:maxSdkVersion="30" />

   <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"
      android:maxSdkVersion="30" />

   <uses-permission android:name="android.permission.BLUETOOTH_SCAN"
      android:usesPermissionFlags="neverForLocation"
      tools:targetApi="s" />

   <uses-permission
android:name="android.permission.BLUETOOTH_CONNECT" />

   <uses-feature
      android:name="android.hardware.bluetooth_le"
      android:required="true" />

   <application
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/AppTheme"
      tools:ignore="GoogleAppIndexingWarning">
```

AndroidManifest.xml

```
        <activity android:name=".BleTracker"
            android:exported="false"
            />
        <activity android:name=".SensorReadings"
            android:exported="false"
            />
        <activity android:name=".BleOperationsActivity"
            android:exported="false"
            />
        <activity android:name=".MainActivity"
            android:exported="true"
            >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

MainActivity

```
package com.punchthrough.blestarterappandroid

import android.Manifest
import android.annotation.SuppressLint
import android.app.Activity
import android.app.AlertDialog
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothManager
import android.bluetooth.le.ScanCallback
import android.bluetooth.le.ScanFilter
import android.bluetooth.le.ScanResult
import android.bluetooth.le.ScanSettings
import android.content.ActivityNotFoundException
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import android.os.Build
```

MainActivity

```
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.provider.Settings
import android.view.View
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.annotation.RequiresApi
import androidx.annotation.UiThread
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.SimpleItemAnimator
import com.punchthrough.blestarterappandroid.ble.ConnectionEventListener
import com.punchthrough.blestarterappandroid.ble.ConnectionManager
import com.punchthrough.blestarterappandroid.databinding.ActivityMainBinding
import timber.log.Timber

private const val PERMISSION_REQUEST_CODE = 1

@RequiresApi(Build.VERSION_CODES.UPSIDE_DOWN_CAKE)
class MainActivity : AppCompatActivity() {

    private val deviceAddress = "02:80:E1:00:00:E1"

    private val scanDuration = 30000
    private var deviceFound = false

    private lateinit var binding: ActivityMainBinding

    private val bluetoothAdapter: BluetoothAdapter by lazy {
        val bluetoothManager =
getSystemService(Context.BLUETOOTH_SERVICE) as BluetoothManager
        bluetoothManager.adapter
    }

    private val bleScanner by lazy {
        bluetoothAdapter.bluetoothLeScanner
    }

    private val scanSettings = ScanSettings.Builder()
        .setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY)
```

MainActivity

```
        .build()

    private val filterSetting =
listOf(ScanFilter.Builder().setDeviceAddress(deviceAddress).build())

    private var isScanning = false
        set(value) {
            field = value
            runOnUiThread {
                binding.scanButton.text = if (value) "STOP SCAN" else "START
SCAN"
                if (value) {
                    binding.scanButton.backgroundTintList =
                        getColorStateList(android.R.color.holo_red_dark)
                } else {
                    binding.scanButton.backgroundTintList =
                        getColorStateList(android.R.color.holo_green_dark)
                }
            }
        }

    private val scanResults = mutableListOf<ScanResult>()
    private val scanResultAdapter: ScanResultAdapter by lazy {
        ScanResultAdapter(scanResults) { result ->
            if (isScanning) {
                stopBleScan()
            }
            with(result.device) {
                Timber.w("Connecting to $address")
                ConnectionManager.connect(this, this@MainActivity)
            }
        }
    }

    private val bluetoothEnablingResult = registerForActivityResult(
        ActivityResultContracts.StartActivityForResult()
    ) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            Timber.i("Bluetooth is enabled, good to go")
        } else {
            Timber.e("User dismissed or denied Bluetooth prompt")
            promptEnableBluetooth()
        }
```

MainActivity

```
}

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   binding = ActivityMainBinding.inflate(layoutInflater)
   setContentView(binding.root)
   if (BuildConfig.DEBUG) {
      Timber.plant(Timber.DebugTree())
   }
   binding.scanButton.setOnClickListener {
      if (isScanning){
         Toast.makeText(this, "Stop Scanning", Toast.LENGTH_SHORT).show()
         stopBleScan()
      }
      else{
         Toast.makeText(this, "Start Scanning", Toast.LENGTH_SHORT).show()
         startBleScan()
      }
   }
   setupRecyclerView()
}


override fun onResume() {
   super.onResume()
   ConnectionManager.registerListener(connectionEventListener)
   if (!bluetoothAdapter.isEnabled) {
      promptEnableBluetooth()
   }
}

override fun onPause() {
   super.onPause()
   if (isScanning) {
      stopBleScan()
   }
   ConnectionManager.unregisterListener(connectionEventListener)
}

override fun onRequestPermissionsResult(
   requestCode: Int,
   permissions: Array<out String>,
   grantResults: IntArray
```

MainActivity

```
   ) {
       super.onRequestPermissionsResult(requestCode, permissions, grantResults)
       if (requestCode != PERMISSION_REQUEST_CODE) {
           return
       }
       if (permissions.isEmpty() && grantResults.isEmpty()) {
           Timber.e("Empty permissions and grantResults array in
onRequestPermissionsResult")
           Timber.w("This is likely a cancellation due to user interaction interrupted")
           return
       }

       // Log permission request outcomes
       val resultsDescriptions = grantResults.map {
           when (it) {
               PackageManager.PERMISSION_DENIED -> "Denied"
               PackageManager.PERMISSION_GRANTED -> "Granted"
               else -> "Unknown"
           }
       }
       Timber.w("Permissions: ${permissions.toList()}, grant results:
$resultsDescriptions")

       val containsPermanentDenial =
permissions.zip(grantResults.toTypedArray()).any {
           it.second == PackageManager.PERMISSION_DENIED &&
               !ActivityCompat.shouldShowRequestPermissionRationale(this, it.first)
       }
       val containsDenial = grantResults.any { it ==
PackageManager.PERMISSION_DENIED }
       val allGranted = grantResults.all { it ==
PackageManager.PERMISSION_GRANTED }
       when {
           containsPermanentDenial -> {
               Timber.e("User permanently denied granting of permissions")
               Timber.e("Requesting for manual granting of permissions from App
Settings")
               promptManualPermissionGranting()
           }
           containsDenial -> {

requestRelevantBluetoothPermissions(PERMISSION_REQUEST_CODE)
           }
```

MainActivity

```
            allGranted && hasRequiredBluetoothPermissions() -> {
                startBleScan()
            }
            else -> {
                Timber.e("Unexpected scenario encountered when handling
permissions")
                recreate()
            }
        }
    }

    private fun promptEnableBluetooth() {
        if (hasRequiredBluetoothPermissions() && !bluetoothAdapter.isEnabled) {
            Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE).apply {
                bluetoothEnablingResult.launch(this)
            }
        }
    }

    @SuppressLint("MissingPermission, NotifyDataSetChanged")
    private fun startBleScan() {
        deviceFound = false
        binding.scanResultsTitle.visibility= View.INVISIBLE
        binding.divider.visibility=View.INVISIBLE

        if (!hasRequiredBluetoothPermissions()) {
            requestRelevantBluetoothPermissions(PERMISSION_REQUEST_CODE)
        } else {
            scanResults.clear()
            scanResultAdapter.notifyDataSetChanged()
            bleScanner.startScan(filterSetting, scanSettings, scanCallback)
            isScanning = true
            Handler(Looper.getMainLooper()).postDelayed({
                if (!deviceFound){
                bleScanner.stopScan(scanCallback)
                Toast.makeText(this, "Device not found",
Toast.LENGTH_SHORT).show()
                isScanning = false
                }
            }, scanDuration.toLong())
        }
    }
```

MainActivity

```
    @SuppressLint("MissingPermission")
    private fun stopBleScan() {
        if (hasRequiredBluetoothPermissions()) {
            bleScanner.stopScan(scanCallback)
            isScanning = false
        }
    }

    @UiThread
    private fun setupRecyclerView() {
        binding.scanResultsRecyclerView.apply {
            adapter = scanResultAdapter
            layoutManager = LinearLayoutManager(
                this@MainActivity,
                RecyclerView.VERTICAL,
                false
            )
            isNestedScrollingEnabled = false
            itemAnimator.let {
                if (it is SimpleItemAnimator) {
                    it.supportsChangeAnimations = false
                }
            }
        }
    }

    @UiThread
    private fun promptManualPermissionGranting() {
        AlertDialog.Builder(this)
            .setTitle(R.string.please_grant_relevant_permissions)
            .setMessage(R.string.app_settings_rationale)
            .setPositiveButton(R.string.app_settings) { _, _ ->
                try {
                    startActivity(

Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS).apply {
                            data = Uri.parse("package:$packageName")
                            addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
                        }
                    )
                } catch (e: ActivityNotFoundException) {
                    if (!isFinishing) {
                        Toast.makeText(
```

MainActivity

```
                    this,
                    R.string.cannot_launch_app_settings,
                    Toast.LENGTH_LONG
                ).show()
            }
        }
        finish()
    }
    .setNegativeButton(R.string.quit) { _, _ -> finishAndRemoveTask() }
    .setCancelable(false)
    .show()
}

@SuppressLint("MissingPermission")
private val scanCallback = object : ScanCallback() {
    override fun onScanResult(callbackType: Int, result: ScanResult) {
        val indexQuery = scanResults.indexOfFirst { it.device.address ==
result.device.address }
        if (indexQuery != -1) {
            scanResults[indexQuery] = result
            scanResultAdapter.notifyItemChanged(indexQuery)
        } else {
            with(result.device) {
                Timber.i("Found BLE device! Name: ${name ?: "Unnamed"},
address: $address")
            }
            scanResults.add(result)
            scanResultAdapter.notifyItemInserted(scanResults.size - 1)
        }

        if (result.device.address == deviceAddress){
            deviceFound = true
            binding.scanResultsTitle.visibility= View.VISIBLE
            binding.divider.visibility=View.VISIBLE
        }
    }

    override fun onScanFailed(errorCode: Int) {
        Timber.e("onScanFailed: code $errorCode")
    }
}

private val connectionEventListener by lazy {
```

MainActivity

```
        ConnectionEventListener().apply {
            onConnectionSetupComplete = { gatt ->
                Intent(this@MainActivity, BleOperationsActivity::class.java).also {
                    it.putExtra(BluetoothDevice.EXTRA_DEVICE, gatt.device)
                    startActivity(it)
                }
            }
            @SuppressLint("MissingPermission")
            onDisconnect = {
                val deviceName = if (hasRequiredBluetoothPermissions()) {
                    it.name
                } else {
                    "device"
                }
                runOnUiThread {
                    AlertDialog.Builder(this@MainActivity)
                        .setTitle(R.string.disconnected)
                        .setMessage(

getString(R.string.disconnected_or_unable_to_connect_to_device, deviceName)
                        )
                        .setPositiveButton(R.string.ok, null)
                        .show()
                }
            }
        }
    }
}
```

ScanResultAdapter

```
package com.punchthrough.blestarterappandroid

import android.annotation.SuppressLint
import android.bluetooth.BluetoothDevice
import android.bluetooth.le.ScanResult
import android.os.Build
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.annotation.RequiresApi
import androidx.recyclerview.widget.RecyclerView
```

ScanResultAdapter

```
class ScanResultAdapter(
    private val items: List<ScanResult>,
    private val onClickListener: ((device: ScanResult) -> Unit)
) : RecyclerView.Adapter<ScanResultAdapter.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(
            R.layout.row_scan_result,
            parent,
            false
        )
        return ViewHolder(view, onClickListener)
    }

    override fun getItemCount() = items.size

    @RequiresApi(Build.VERSION_CODES.UPSIDE_DOWN_CAKE)
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = items[position]
        holder.bind(item)
    }

    class ViewHolder(
        private val view: View,
        private val onClickListener: ((device: ScanResult) -> Unit)
    ) : RecyclerView.ViewHolder(view) {

        @RequiresApi(Build.VERSION_CODES.UPSIDE_DOWN_CAKE)
        @SuppressLint("MissingPermission", "SetTextI18n")
        fun bind(result: ScanResult) {
            view.findViewById<TextView>(R.id.device_name).text =
                if (view.context.hasRequiredBluetoothPermissions()) {
                    result.device.name ?: "Unnamed"
                } else {
                    error("Missing required Bluetooth permissions")
                }
            view.findViewById<TextView>(R.id.mac_address).text =
result.device.address
            view.findViewById<TextView>(R.id.signal_strength).text =
"${result.rssi} dBm"
            view.setOnClickListener { onClickListener.invoke(result) }
```

ScanResultAdapter

```
            val deviceType = getDeviceType(result.device.type)
            view.findViewById<TextView>(R.id.device_type).text = "Device type:
$deviceType"

            view.findViewById<TextView>(R.id.advertising_type).text =
                if (result.isLegacy) "Advertising type: Legacy" else "Advertising type:
Bluetooth 5 Advertising Extension"

            val primaryPhy = getPhy(result.primaryPhy)
            val secondaryPhy = getPhy(result.secondaryPhy)

            if (secondaryPhy != "null") { // if have no secondary advertising channel
                view.findViewById<TextView>(R.id.primary_phy).text = "Primary
PHY: $primaryPhy"
                view.findViewById<TextView>(R.id.secondary_phy).text =
                    "Secondary PHY: $secondaryPhy"
            }

        }

        private fun getDeviceType(deviceType: Int): String {
            return when (deviceType) {
                BluetoothDevice.DEVICE_TYPE_CLASSIC -> "Classic only"
                BluetoothDevice.DEVICE_TYPE_LE -> "LE only"
                BluetoothDevice.DEVICE_TYPE_DUAL -> " Classic and LE"
                BluetoothDevice.DEVICE_TYPE_UNKNOWN -> "Unknown"
                else -> "Unknown"
            }
        }

        private fun getPhy (phy: Int): String {
            return when (phy) {
                BluetoothDevice.PHY_LE_1M -> "LE 1M"
                BluetoothDevice.PHY_LE_2M -> "LE 2M"
                BluetoothDevice.PHY_LE_CODED -> "LE Coded"
                ScanResult.PHY_UNUSED -> "null"
                else -> "Unknown"
            }
        }
    }
}
```

CharacteristicAdapter

```
package com.punchthrough.blestarterappandroid

import android.bluetooth.BluetoothGattCharacteristic
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.punchthrough.blestarterappandroid.ble.printProperties

class CharacteristicAdapter(
    private val items: List<BluetoothGattCharacteristic>,
    private val onClickListener: ((characteristic: BluetoothGattCharacteristic) ->
Unit)
) : RecyclerView.Adapter<CharacteristicAdapter.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(
            R.layout.row_characteristic,
            parent,
            false
        )
        return ViewHolder(view, onClickListener)
    }

    override fun getItemCount() = items.size

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = items[position]
        holder.bind(item)
    }

    class ViewHolder(
        private val view: View,
        private val onClickListener: ((characteristic: BluetoothGattCharacteristic) ->
Unit)
    ) : RecyclerView.ViewHolder(view) {

        fun bind(characteristic: BluetoothGattCharacteristic) {
            view.findViewById<TextView>(R.id.characteristic_uuid).text =
                characteristic.uuid.toString()
            view.findViewById<TextView>(R.id.characteristic_properties).text =
```

CharacteristicAdapter

```
            characteristic.printProperties()
          view.setOnClickListener { onClickListener.invoke(characteristic) }
        }
    }
}
```

ConnectionManager

```
package com.punchthrough.blestarterappandroid.ble

import android.annotation.SuppressLint
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothGatt
import android.bluetooth.BluetoothGattCallback
import android.bluetooth.BluetoothGattCharacteristic
import android.bluetooth.BluetoothGattDescriptor
import android.bluetooth.BluetoothGattService
import android.bluetooth.BluetoothProfile
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.os.Build
import android.os.Handler
import android.os.Looper
import android.os.Parcelable
import androidx.annotation.RequiresApi
import timber.log.Timber
import java.lang.ref.WeakReference
import java.util.Timer
import java.util.TimerTask
import java.util.UUID
import java.util.concurrent.ConcurrentHashMap
import java.util.concurrent.ConcurrentLinkedQueue

/** Maximum BLE MTU size as defined in gatt_api.h. */
private const val GATT_MAX_MTU_SIZE = 517
private const val GATT_MIN_MTU_SIZE = 23

@SuppressLint("MissingPermission")
object ConnectionManager {

    private var listeners: MutableSet<WeakReference<ConnectionEventListener>>
```

ConnectionManager

```
= mutableSetOf()
    private val listenersAsSet
        get() = listeners.toSet()

    val deviceGattMap = ConcurrentHashMap<BluetoothDevice, BluetoothGatt>()
    private val operationQueue = ConcurrentLinkedQueue<BleOperationType>()
    private var pendingOperation: BleOperationType? = null

    private var gatt: BluetoothGatt? = null

    fun getGatt(): BluetoothGatt? {
        return gatt
    }

    fun servicesOnDevice(device: BluetoothDevice): List<BluetoothGattService>?
=
        deviceGattMap[device]?.services

    fun listenToBondStateChanges(context: Context) {
        context.applicationContext.registerReceiver(
            broadcastReceiver,
            IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED)
        )
    }

    fun registerListener(listener: ConnectionEventListener) {
        if (listeners.map { it.get() }.contains(listener)) { return }
        listeners.add(WeakReference(listener))
        listeners = listeners.filter { it.get() != null }.toMutableSet()
        Timber.d("Added listener $listener, ${listeners.size} listeners total")
    }

    fun unregisterListener(listener: ConnectionEventListener) {

        var toRemove: WeakReference<ConnectionEventListener>? = null
        listenersAsSet.forEach {
            if (it.get() == listener) {
                toRemove = it
            }
        }
        toRemove?.let {
            listeners.remove(it)
            Timber.d("Removed listener ${it.get()}, ${listeners.size} listeners total")
```

ConnectionManager

```
        }
    }

    fun connect(device: BluetoothDevice, context: Context) {
        gatt = device.connectGatt(context, false, callback)
        if (device.isConnected()) {
            Timber.e("Already connected to ${device.address}!")
        } else {
            enqueueOperation(Connect(device, context.applicationContext))
        }

    }



    fun disconnect() {
        gatt?.disconnect()
        gatt?.close()
        gatt = null
    }

    fun teardownConnection(device: BluetoothDevice) {
        if (device.isConnected()) {
            enqueueOperation(Disconnect(device))
        } else {
            Timber.e("Not connected to ${device.address}, cannot teardown
connection!")
        }
    }

    fun readCharacteristic(device: BluetoothDevice, characteristic:
BluetoothGattCharacteristic) {
        if (device.isConnected() && characteristic.isReadable()) {
            enqueueOperation(CharacteristicRead(device, characteristic.uuid))
        } else if (!characteristic.isReadable()) {
            Timber.e("Attempting to read ${characteristic.uuid} that isn't readable!")
        } else if (!device.isConnected()) {
            Timber.e("Not connected to ${device.address}, cannot perform
characteristic read")
        }
    }

    fun writeCharacteristic(
        device: BluetoothDevice,
```

ConnectionManager

```
        characteristic: BluetoothGattCharacteristic,
        payload: ByteArray
    ) {
        val writeType = when {
            characteristic.isWritable() ->
BluetoothGattCharacteristic.WRITE_TYPE_DEFAULT
            characteristic.isWritableWithoutResponse() -> {
                BluetoothGattCharacteristic.WRITE_TYPE_NO_RESPONSE
            }
            else -> {
                Timber.e("Characteristic ${characteristic.uuid} cannot be written to")
                return
            }
        }
        if (device.isConnected()) {
            enqueueOperation(CharacteristicWrite(device, characteristic.uuid,
writeType, payload))
        } else {
            Timber.e("Not connected to ${device.address}, cannot perform
characteristic write")
        }
    }

    fun readDescriptor(device: BluetoothDevice, descriptor:
BluetoothGattDescriptor) {
        if (device.isConnected() && descriptor.isReadable()) {
            enqueueOperation(DescriptorRead(device, descriptor.uuid))
        } else if (!descriptor.isReadable()) {
            Timber.e("Attempting to read ${descriptor.uuid} that isn't readable!")
        } else if (!device.isConnected()) {
            Timber.e("Not connected to ${device.address}, cannot perform descriptor
read")
        }
    }

    fun writeDescriptor(
        device: BluetoothDevice,
        descriptor: BluetoothGattDescriptor,
        payload: ByteArray
    ) {
        if (device.isConnected() && (descriptor.isWritable() || descriptor.isCccd())) {
            enqueueOperation(DescriptorWrite(device, descriptor.uuid, payload))
        } else if (!device.isConnected()) {
```

ConnectionManager

```
            Timber.e("Not connected to ${device.address}, cannot perform descriptor
write")
        } else if (!descriptor.isWritable() && !descriptor.isCccd()) {
            Timber.e("Descriptor ${descriptor.uuid} cannot be written to")
        }
    }

    fun enableNotifications(device: BluetoothDevice, characteristic:
BluetoothGattCharacteristic) {
        if (device.isConnected() &&
            (characteristic.isIndicatable() || characteristic.isNotifiable())
        ) {
            enqueueOperation(EnableNotifications(device, characteristic.uuid))
        } else if (!device.isConnected()) {
            Timber.e("Not connected to ${device.address}, cannot enable
notifications")
        } else if (!characteristic.isIndicatable() && !characteristic.isNotifiable()) {
            Timber.e("Characteristic ${characteristic.uuid} doesn't support
notifications/indications")
        }
    }

    fun disableNotifications(device: BluetoothDevice, characteristic:
BluetoothGattCharacteristic) {
        if (device.isConnected() &&
            (characteristic.isIndicatable() || characteristic.isNotifiable())
        ) {
            enqueueOperation(DisableNotifications(device, characteristic.uuid))
        } else if (!device.isConnected()) {
            Timber.e("Not connected to ${device.address}, cannot disable
notifications")
        } else if (!characteristic.isIndicatable() && !characteristic.isNotifiable()) {
            Timber.e("Characteristic ${characteristic.uuid} doesn't support
notifications/indications")
        }
    }

    fun requestMtu(device: BluetoothDevice, mtu: Int) {
        if (device.isConnected()) {
            enqueueOperation(MtuRequest(device,
mtu.coerceIn(GATT_MIN_MTU_SIZE, GATT_MAX_MTU_SIZE)))
        } else {
            Timber.e("Not connected to ${device.address}, cannot request MTU
```

ConnectionManager

```
update!")
        }

    }

    @Synchronized
    private fun enqueueOperation(operation: BleOperationType) {
        operationQueue.add(operation)
        if (pendingOperation == null) {
            doNextOperation()
        }
    }

    @Synchronized
    private fun signalEndOfOperation() {
        Timber.d("End of $pendingOperation")
        pendingOperation = null
        if (operationQueue.isNotEmpty()) {
            doNextOperation()
        }
    }

    /**

    @Synchronized
    private fun doNextOperation() {
        if (pendingOperation != null) {
            Timber.e("doNextOperation() called when an operation is pending!
Aborting.")
            return
        }

        val operation = operationQueue.poll() ?: run {
            Timber.v("Operation queue empty, returning")
            return
        }
        pendingOperation = operation

        if (operation is Connect) {
            with(operation) {
                Timber.w("Connecting to ${device.address}")
                device.connectGatt(context, false, callback)
            }
```

ConnectionManager

```
            return
        }

        val gatt = deviceGattMap[operation.device]
            ?: this@ConnectionManager.run {
                Timber.e("Not connected to ${operation.device.address}! Aborting
$operation operation.")
                signalEndOfOperation()
                return
            }

        when (operation) {
            is Disconnect -> with(operation) {
                Timber.w("Disconnecting from ${device.address}")
                gatt.close()
                deviceGattMap.remove(device)
                listenersAsSet.forEach { it.get()?.onDisconnect?.invoke(device) }
                signalEndOfOperation()
            }
            is CharacteristicWrite -> with(operation) {
                gatt.findCharacteristic(characteristicUuid)?.executeWrite(
                    gatt,
                    payload,
                    writeType
                ) ?: this@ConnectionManager.run {
                    Timber.e("Cannot find $characteristicUuid to write to")
                    signalEndOfOperation()
                }
            }
            is CharacteristicRead -> with(operation) {
                gatt.findCharacteristic(characteristicUuid)?.let { characteristic ->
                    gatt.readCharacteristic(characteristic)
                } ?: this@ConnectionManager.run {
                    Timber.e("Cannot find $characteristicUuid to read from")
                    signalEndOfOperation()
                }
            }
            is DescriptorWrite -> with(operation) {
                gatt.findDescriptor(descriptorUuid)?.executeWrite(
                    gatt,
                    payload
                ) ?: this@ConnectionManager.run {
                    Timber.e("Cannot find $descriptorUuid to write to")
```

ConnectionManager

```
                    signalEndOfOperation()
                }
            }
            is DescriptorRead -> with(operation) {
                gatt.findDescriptor(descriptorUuid)?.let { descriptor ->
                    gatt.readDescriptor(descriptor)
                } ?: this@ConnectionManager.run {
                    Timber.e("Cannot find $descriptorUuid to read from")
                    signalEndOfOperation()
                }
            }
            is EnableNotifications -> with(operation) {
                gatt.findCharacteristic(characteristicUuid)?.let { characteristic ->
                    val cccdUuid = UUID.fromString(CCC_DESCRIPTOR_UUID)
                    val payload = when {
                        characteristic.isIndicatable() ->
                            BluetoothGattDescriptor.ENABLE_INDICATION_VALUE
                        characteristic.isNotifiable() ->
                            BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
                        else ->
                            error("${characteristic.uuid} doesn't support
notifications/indications")
                    }

                    characteristic.getDescriptor(cccdUuid)?.let { cccDescriptor ->
                        if (!gatt.setCharacteristicNotification(characteristic, true)) {
                            Timber.e("setCharacteristicNotification failed for
${characteristic.uuid}")
                            signalEndOfOperation()
                            return
                        }
                        cccDescriptor.executeWrite(gatt, payload)
                    } ?: this@ConnectionManager.run {
                        Timber.e("${characteristic.uuid} doesn't contain the CCC
descriptor!")
                        signalEndOfOperation()
                    }
                } ?: this@ConnectionManager.run {
                    Timber.e("Cannot find $characteristicUuid! Failed to enable
notifications.")
                    signalEndOfOperation()
                }
            }
```

ConnectionManager

```
            is DisableNotifications -> with(operation) {
                gatt.findCharacteristic(characteristicUuid)?.let { characteristic ->
                    val cccdUuid = UUID.fromString(CCC_DESCRIPTOR_UUID)
                    characteristic.getDescriptor(cccdUuid)?.let { cccDescriptor ->
                        if (!gatt.setCharacteristicNotification(characteristic, false)) {
                            Timber.e("setCharacteristicNotification failed for
${characteristic.uuid}")
                            signalEndOfOperation()
                            return
                        }
                        cccDescriptor.executeWrite(
                            gatt,
                            BluetoothGattDescriptor.DISABLE_NOTIFICATION_VALUE
                        )
                    } ?: this@ConnectionManager.run {
                        Timber.e("${characteristic.uuid} doesn't contain the CCC
descriptor!")
                        signalEndOfOperation()
                    }
                } ?: this@ConnectionManager.run {
                    Timber.e("Cannot find $characteristicUuid! Failed to disable
notifications.")
                    signalEndOfOperation()
                }
            }
            is MtuRequest -> with(operation) {
                gatt.requestMtu(mtu)
            }



            else -> error("Unsupported operation: $operation")
        }
    }

    private val rssiTimer = Timer() // Implement timer
    private const val rssi_Interval: Long = 500 // 0.5 second to refresh rssi value

    private fun startRssiRead(gatt: BluetoothGatt) {
        rssiTimer.scheduleAtFixedRate(object : TimerTask() {
            override fun run() {
                gatt.readRemoteRssi() //Ensure the rssi value is accessed
            }
        }, 0, rssi_Interval)
```

ConnectionManager

```
    }

    private fun stopRssiRead() {
        rssiTimer.cancel() // Stop timer
    }

    private fun readPhy(gatt: BluetoothGatt) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            gatt.readPhy()
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    private fun preferredPhy (gatt: BluetoothGatt){
        gatt.setPreferredPhy(
            BluetoothDevice.PHY_LE_CODED_MASK,
            BluetoothDevice.PHY_LE_CODED_MASK,
            BluetoothDevice.PHY_OPTION_S8
        )
    }

    var txphydisplay: Int? = null
    var rxphydisplay: Int? = null

    var deviceName: String? =null

    private val callback = object : BluetoothGattCallback() {
        override fun onConnectionStateChange(gatt: BluetoothGatt, status: Int,
newState: Int) {
            val deviceAddress = gatt.device.address

            if (status == BluetoothGatt.GATT_SUCCESS) {
                if (newState == BluetoothProfile.STATE_CONNECTED) {
                    startRssiRead(gatt)
                    Timber.w("onConnectionStateChange: connected to $deviceAddress")
                    deviceGattMap[gatt.device] = gatt
                    Handler(Looper.getMainLooper()).post {
                        gatt.discoverServices()
                    }
                } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
                    stopRssiRead()
                    Timber.e("onConnectionStateChange: disconnected from
$deviceAddress")
```

ConnectionManager

```
                teardownConnection(gatt.device)
            }
        } else {
            Timber.e("onConnectionStateChange: status $status encountered for
$deviceAddress!")
            if (pendingOperation is Connect) {
                signalEndOfOperation()
            }
            stopTempRead()
            stopRssiRead()
            teardownConnection(gatt.device)
        }
    }

    @RequiresApi(Build.VERSION_CODES.O)
    override fun onServicesDiscovered(gatt: BluetoothGatt, status: Int) {
        with(gatt) {
            if (status == BluetoothGatt.GATT_SUCCESS) {

                gatt.services?.forEach { service ->
                    Timber.d("Service discovered: ${service.uuid}")
                    service.characteristics.forEach { characteristic ->
                        Timber.d("Characteristic discovered: ${characteristic.uuid}")
                    }
                }

                val deviceNameCharacteristic =
gatt.getService(UUID.fromString("00001800-0000-1000-8000-00805f9b34fb"))
                    ?.getCharacteristic(UUID.fromString("00002a00-0000-1000-8000-
00805f9b34fb"))

                if (deviceNameCharacteristic != null) {
                    gatt.readCharacteristic(deviceNameCharacteristic)
                } else {
                    Timber.e("Device Name characteristic not found.")
                }

                readPhy(gatt)
                preferredPhy(gatt)
                Timber.w("Discovered ${services.size} services for
${device.address}.")
                printGattTable()
                requestMtu(device, GATT_MAX_MTU_SIZE)
```

ConnectionManager

```
                    listenersAsSet.forEach
{ it.get()?.onConnectionSetupComplete?.invoke(this) }
                } else {
                    Timber.e("Service discovery failed due to status $status")
                    teardownConnection(gatt.device)
                }
            }

            if (pendingOperation is Connect) {
                signalEndOfOperation()
            }
        }

        override fun onReadRemoteRssi(gatt: BluetoothGatt, rssi: Int, status: Int) {
            super.onReadRemoteRssi(gatt, rssi, status)
            if (status == BluetoothGatt.GATT_SUCCESS){
                listenersAsSet.forEach
{ it.get()?.onReadRemoteRSSI?.invoke(gatt.device, rssi) }
            }
        }

        override fun onPhyRead(gatt: BluetoothGatt, txPhy: Int, rxPhy: Int, status:
Int) {
            super.onPhyRead(gatt, txPhy, rxPhy, status)
            if (status == BluetoothGatt.GATT_SUCCESS) {
                Timber.i("TxPHY: $txPhy, RxPHY: $rxPhy")
                txphydisplay = txPhy
                rxphydisplay = rxPhy
            } else {
                Timber.e("Failed to display PHY")
            }
            }


        override fun onPhyUpdate(gatt: BluetoothGatt, txPhy: Int, rxPhy: Int, status:
Int) {
            super.onPhyUpdate(gatt, txPhy, rxPhy, status)
            if (status == BluetoothGatt.GATT_SUCCESS) {
                listenersAsSet.forEach { it.get()?.onPhyUpdate?.invoke(gatt.device,
txPhy, rxPhy) }
                Timber.i("TxPHY_updated: $txPhy, RxPHY_updated: $rxPhy")
            } else {
                Timber.e("Failed to update PHY")
```

ConnectionManager

```
            }
        }

        override fun onMtuChanged(gatt: BluetoothGatt, mtu: Int, status: Int) {
            Timber.w("ATT MTU changed to $mtu, success: ${status ==
BluetoothGatt.GATT_SUCCESS}")
            listenersAsSet.forEach { it.get()?.onMtuChanged?.invoke(gatt.device,
mtu) }

            if (pendingOperation is MtuRequest) {
                signalEndOfOperation()
            }
        }


        private val tempTimer = Timer() // Implement timer
        private val temp_Interval: Long = 500

        private fun startTempRead(
            gatt: BluetoothGatt,
            characteristic: BluetoothGattCharacteristic,
        ) {
            tempTimer.scheduleAtFixedRate(object : TimerTask() {
                override fun run() {
                    gatt.readCharacteristic(characteristic)
                }
            }, 0, temp_Interval)
        }

        private fun stopTempRead() {
            tempTimer.cancel() // Stop timer
        }

        @Deprecated("Deprecated for Android 13+")
        @Suppress("DEPRECATION")
        override fun onCharacteristicRead(
            gatt: BluetoothGatt,
            characteristic: BluetoothGattCharacteristic,
            status: Int
        ) {
            with(characteristic) {
                when (status) {
                    BluetoothGatt.GATT_SUCCESS -> {
```

ConnectionManager

```
            Timber.i("Read characteristic $uuid | value:
${value.toHexString()}")

            listenersAsSet.forEach {
              it.get()?.onCharacteristicRead?.invoke(
                gatt.device,
                this,
                value
              )
            }

            val characteristicUUID = characteristic.uuid
            Timber.d("Characteristic UUID: $characteristicUUID read
successfully.")
            Timber.d("Characteristic value:
${characteristic.value?.toHexString()}")

            when (characteristicUUID) {
              UUID.fromString("00002a00-0000-1000-8000-00805f9b34fb")
-> {
                deviceName = characteristic.value?.toString(Charsets.UTF_8)
                if (deviceName != null) {
                  Timber.i("Device Name: $deviceName")
                } else {
                  Timber.e("Failed to read device name or value is null.")
                }

                // After reading the Device Name, read the Temperature
characteristic
                readTemperatureCharacteristic(gatt)
              }

              UUID.fromString("a32e5520-e477-11e2-a9e3-00002a5d5c51b")
-> {
                Timber.d("Temperature characteristic detected.")
                handleTemperatureCharacteristic(gatt, characteristic)
                // Read the Acceleration+Gyroscope characteristic
                readAccGyroCharacteristic(gatt)
              }

              UUID.fromString("340a1b80-cf4b-11e1-ac36-0002a5d5c51b")
-> {
                Timber.d("Acceleration+Gyroscope characteristic detected.")
```

ConnectionManager

```
                handleAccGyroCharacteristic(gatt, characteristic)
                readTxCharacteristic(gatt)
              }

              UUID.fromString("d973f2e1-b19e-11e2-9e96-0800200c9a66")
-> {
                Timber.d("TX characteristic detected.")
                handleTxCharacteristic(gatt, characteristic)
              }

              else -> {
                Timber.d("Unknown characteristic UUID:
$characteristicUUID")
              }
            }
          }

          BluetoothGatt.GATT_READ_NOT_PERMITTED -> {
            Timber.e("Read not permitted for $uuid!")
          }

          else -> {
            Timber.e("Characteristic read failed for $uuid, error: $status")

          }
        }
      }

      if (pendingOperation is CharacteristicRead) {
        signalEndOfOperation()
      }
    }

    private fun readTemperatureCharacteristic(gatt: BluetoothGatt) {
      val temperatureCharacteristic =
        gatt.getService(UUID.fromString("42821a40-e477-11e2-82d0-
0002a5d5c51b"))
          ?.getCharacteristic(UUID.fromString("a32e5520-e477-11e2-a9e3-
00002a5d5c51b"))

      if (temperatureCharacteristic != null) {
        startTempRead(gatt, temperatureCharacteristic)
      } else {
```

ConnectionManager

```
            Timber.e("Temperature characteristic not found.")
        }
    }

    private fun handleTemperatureCharacteristic(gatt: BluetoothGatt,
characteristic: BluetoothGattCharacteristic) {
        val value = characteristic.value
        if (value != null && value.size >= 2) {
            Timber.d("Temperature value received: ${value.joinToString("-")
{ String.format("%02X", it) }}")

            val msb = value[1].toInt() and 0xFF
            val lsb = value[0].toInt() and 0xFF
            val temperatureValue = ((msb shl 8) or lsb) / 10.0

            Timber.i("Converted Temperature Value: $temperatureValue °C")

            listenersAsSet.forEach { it.get()?.onTempUpdate?.invoke(gatt.device,
temperatureValue) }
        } else {
            Timber.e("Invalid temperature value received or value array size is less
than 2.")
        }
    }

    private fun readAccGyroCharacteristic(gatt: BluetoothGatt) {
        val accCharacteristic =
            gatt.getService(UUID.fromString("02366e80-cf3a-11e1-9ab4-
0002a5d5c51b"))
                ?.getCharacteristic(UUID.fromString("340a1b80-cf4b-11e1-ac36-
0002a5d5c51b"))

        if (accCharacteristic != null) {
            gatt.readCharacteristic(accCharacteristic)
        } else {
            Timber.e("Acceleration+Gyroscope characteristic not found.")
        }
    }

    private fun handleAccGyroCharacteristic(gatt: BluetoothGatt, characteristic:
BluetoothGattCharacteristic) {
        val value = characteristic.value
        if (value != null && value.size >= 14) {
```

ConnectionManager

```
            Timber.d("Characteristic value received: ${value.joinToString("-")
{ String.format("%02X", it) }}")

            val time = (value[1].toInt() and 0xFF shl 8) or (value[0].toInt() and
0xFF)

            val yAxisAccelerationInverted = (value[2].toInt() and 0xFF) or
((value[3].toInt() and 0xFF) shl 8)
            val xAxisAccelerationInverted = (value[4].toInt() and 0xFF) or
((value[5].toInt() and 0xFF) shl 8)
            val zAxisAccelerationInverted = (value[6].toInt() and 0xFF) or
((value[7].toInt() and 0xFF) shl 8)
            val yAxisGyroscope = (value[8].toInt() and 0xFF) or ((value[9].toInt()
and 0xFF) shl 8)
            val xAxisGyroscope = (value[10].toInt() and 0xFF) or ((value[11].toInt()
and 0xFF) shl 8)
            val zAxisGyroscope = (value[12].toInt() and 0xFF) or ((value[13].toInt()
and 0xFF) shl 8)

            val yAxisAcceleration = if (yAxisAccelerationInverted > 32767)
yAxisAccelerationInverted - 65536 else yAxisAccelerationInverted
            val xAxisAcceleration = if (xAxisAccelerationInverted > 32767)
xAxisAccelerationInverted - 65536 else xAxisAccelerationInverted
            val zAxisAcceleration = if (zAxisAccelerationInverted > 32767)
zAxisAccelerationInverted - 65536 else zAxisAccelerationInverted
            val yAxisGyro = if (yAxisGyroscope > 32767) yAxisGyroscope - 65536
else yAxisGyroscope
            val xAxisGyro = if (xAxisGyroscope > 32767) xAxisGyroscope - 65536
else xAxisGyroscope
            val zAxisGyro = if (zAxisGyroscope > 32767) zAxisGyroscope - 65536
else zAxisGyroscope

            Timber.i("Time: $time ms")
            Timber.i("Y-Axis Acceleration: $yAxisAcceleration")
            Timber.i("X-Axis Acceleration: $xAxisAcceleration")
            Timber.i("Z-Axis Acceleration: $zAxisAcceleration")
            Timber.i("Y-Axis Gyroscope: $yAxisGyro")
            Timber.i("X-Axis Gyroscope: $xAxisGyro")
            Timber.i("Z-Axis Gyroscope: $zAxisGyro")

            listenersAsSet.forEach {
                it.get()?.onSensorDataUpdate?.invoke(
                    gatt.device,
```

ConnectionManager

```
                time,
                yAxisAcceleration,
                xAxisAcceleration,
                zAxisAcceleration,
                yAxisGyro,
                xAxisGyro,
                zAxisGyro
            )
        }
    } else {
        Timber.e("Invalid characteristic value received or value array size is less than expected.")
    }
}

private fun readTxCharacteristic(gatt: BluetoothGatt) {
    val txCharUuid = UUID.fromString("d973f2e1-b19e-11e2-9e96-0800200c9a66")
    val serviceUuid = UUID.fromString("d973f2e0-b19e-11e2-9e96-0800200c9a66")
    val service = gatt.getService(serviceUuid)
    val txCharacteristic = service?.getCharacteristic(txCharUuid)

    if (txCharacteristic != null) {
        gatt.readCharacteristic(txCharacteristic)
        Timber.i("Requested read for TX characteristic.")
    } else {
        Timber.e("TX characteristic not found.")
    }
}

private fun handleTxCharacteristic(gatt: BluetoothGatt, characteristic: BluetoothGattCharacteristic) {
    val txValueBytes = characteristic.value
    val txValue = if (txValueBytes != null && txValueBytes.isNotEmpty()) {
        txValueBytes[0].toInt()
    } else {
        0
    }
    listenersAsSet.forEach { it.get()?.onTxUpdate?.invoke(gatt.device, txValue) }

    Timber.i("TX Value: $txValue")
```

ConnectionManager

```
    }


    override fun onCharacteristicRead(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic,
        value: ByteArray,
        status: Int
    ) {
        val uuid = characteristic.uuid
        when (status) {
            BluetoothGatt.GATT_SUCCESS -> {
                Timber.i("Read characteristic $uuid | value: ${value.toHexString()}")
                listenersAsSet.forEach {
                    it.get()?.onCharacteristicRead?.invoke(gatt.device, characteristic,
value)
                }
            }
            BluetoothGatt.GATT_READ_NOT_PERMITTED -> {
                Timber.e("Read not permitted for $uuid!")
            }
            else -> {
                Timber.e("Characteristic read failed for $uuid, error: $status")
            }
        }

        if (pendingOperation is CharacteristicRead) {
            signalEndOfOperation()
        }
    }

    override fun onCharacteristicWrite(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic,
        status: Int
    ) {
        val writtenValue = (pendingOperation as? CharacteristicWrite)?.payload
        with(characteristic) {
            when (status) {
                BluetoothGatt.GATT_SUCCESS -> {
                    Timber.i("Wrote to characteristic $uuid | value:
${writtenValue?.toHexString()}")
                    listenersAsSet.forEach
```

ConnectionManager

```
{ it.get()?.onCharacteristicWrite?.invoke(gatt.device, this) };
                }
                BluetoothGatt.GATT_WRITE_NOT_PERMITTED -> {
                    Timber.e("Write not permitted for $uuid!")
                }
                else -> {
                    Timber.e("Characteristic write failed for $uuid, error: $status")
                }
            }
        }

        if (pendingOperation is CharacteristicWrite) {
            signalEndOfOperation()
        }
    }



    @Deprecated("Deprecated for Android 13+")
    @Suppress("DEPRECATION")
    override fun onCharacteristicChanged(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic
    ) {
        with(characteristic) {
            Timber.i("Characteristic $uuid changed | value:
${value.toHexString()}")
            listenersAsSet.forEach {
                it.get()?.onCharacteristicChanged?.invoke(gatt.device, this, value)
            }
        }
    }

    override fun onCharacteristicChanged(
        gatt: BluetoothGatt,
        characteristic: BluetoothGattCharacteristic,
        value: ByteArray
    ) {
        Timber.i("Characteristic ${characteristic.uuid} changed | value:
${value.toHexString()}")
        listenersAsSet.forEach {
            it.get()?.onCharacteristicChanged?.invoke(gatt.device, characteristic,
value)
```

ConnectionManager

```
        }
    }

    @Deprecated("Deprecated for Android 13+")
    @Suppress("DEPRECATION")
    override fun onDescriptorRead(
        gatt: BluetoothGatt,
        descriptor: BluetoothGattDescriptor,
        status: Int
    ) {
        with(descriptor) {
            when (status) {
                BluetoothGatt.GATT_SUCCESS -> {
                    Timber.i("Read descriptor $uuid | value: ${value.toHexString()}")
                    listenersAsSet.forEach {
                        it.get()?.onDescriptorRead?.invoke(gatt.device, this, value)
                    }
                }
                BluetoothGatt.GATT_READ_NOT_PERMITTED -> {
                    Timber.e("Read not permitted for $uuid!")
                }
                else -> {
                    Timber.e("Descriptor read failed for $uuid, error: $status")
                }
            }
        }

        if (pendingOperation is DescriptorRead) {
            signalEndOfOperation()
        }
    }

    override fun onDescriptorRead(
        gatt: BluetoothGatt,
        descriptor: BluetoothGattDescriptor,
        status: Int,
        value: ByteArray
    ) {
        val uuid = descriptor.uuid
        when (status) {
            BluetoothGatt.GATT_SUCCESS -> {
                Timber.i("Read descriptor $uuid | value: ${value.toHexString()}")
                listenersAsSet.forEach {
```

ConnectionManager

```
                    it.get()?.onDescriptorRead?.invoke(gatt.device, descriptor, value)
                }
            }
            BluetoothGatt.GATT_READ_NOT_PERMITTED -> {
                Timber.e("Read not permitted for $uuid!")
            }
            else -> {
                Timber.e("Descriptor read failed for $uuid, error: $status")
            }
        }

        if (pendingOperation is DescriptorRead) {
            signalEndOfOperation()
        }
    }

    override fun onDescriptorWrite(
        gatt: BluetoothGatt,
        descriptor: BluetoothGattDescriptor,
        status: Int
    ) {
        val operationType = pendingOperation
        with(descriptor) {
            when (status) {
                BluetoothGatt.GATT_SUCCESS -> {
                    Timber.i("Wrote to descriptor $uuid | operation type:
$operationType")

                    if (isCccd() &&
                        (operationType is EnableNotifications || operationType is
DisableNotifications)
                    ) {
                        onCccdWrite(gatt, characteristic, operationType)
                    } else {
                        listenersAsSet.forEach {
                            it.get()?.onDescriptorWrite?.invoke(gatt.device, this)
                        }
                    }
                }
                BluetoothGatt.GATT_WRITE_NOT_PERMITTED -> {
                    Timber.e("Write not permitted for $uuid!")
                }
                else -> {
```

ConnectionManager

```
                Timber.e("Descriptor write failed for $uuid, error: $status")
            }
        }
    }

    val isNotificationsOperation = descriptor.isCccd() &&
        (operationType is EnableNotifications || operationType is
DisableNotifications)
    val isManualWriteOperation = !descriptor.isCccd() && operationType is
DescriptorWrite
    if (isNotificationsOperation || isManualWriteOperation) {
        signalEndOfOperation()
    }
}

private fun onCccdWrite(
    gatt: BluetoothGatt,
    characteristic: BluetoothGattCharacteristic,
    operationType: BleOperationType
) {
    val charUuid = characteristic.uuid

    when (operationType) {
        is EnableNotifications -> {
            Timber.w("Notifications or indications ENABLED on $charUuid")
            listenersAsSet.forEach {
                it.get()?.onNotificationsEnabled?.invoke(
                    gatt.device,
                    characteristic
                )
            }
        }
        is DisableNotifications -> {
            Timber.w("Notifications or indications DISABLED on $charUuid")
            listenersAsSet.forEach {
                it.get()?.onNotificationsDisabled?.invoke(
                    gatt.device,
                    characteristic
                )
            }
        }
        else -> {
            Timber.e("Unexpected operation type of $operationType on CCCD of
```

ConnectionManager

```
$charUuid")
            }
        }
      }
  }

  private val broadcastReceiver = object : BroadcastReceiver() {
      override fun onReceive(context: Context, intent: Intent) {
          with(intent) {
              if (action == BluetoothDevice.ACTION_BOND_STATE_CHANGED) {
                  val device =
parcelableExtraCompat<BluetoothDevice>(BluetoothDevice.EXTRA_DEVICE)
                  val previousBondState =
getIntExtra(BluetoothDevice.EXTRA_PREVIOUS_BOND_STATE, -1)
                  val bondState = getIntExtra(BluetoothDevice.EXTRA_BOND_STATE,
-1)
                  val bondTransition =
"${previousBondState.toBondStateDescription()} to " +
                      bondState.toBondStateDescription()
                  Timber.w("${device?.address} bond state changed |
$bondTransition")
              }
          }
      }

      private fun Int.toBondStateDescription() = when (this) {
          BluetoothDevice.BOND_BONDED -> "BONDED"
          BluetoothDevice.BOND_BONDING -> "BONDING"
          BluetoothDevice.BOND_NONE -> "NOT BONDED"
          else -> "ERROR: $this"
      }
  }

  private fun BluetoothDevice.isConnected() = deviceGattMap.containsKey(this)

  internal inline fun <reified T : Parcelable> Intent.parcelableExtraCompat(key:
String): T? = when {
      Build.VERSION.SDK_INT > Build.VERSION_CODES.TIRAMISU ->
getParcelableExtra(key, T::class.java)
      else -> @Suppress("DEPRECATION") getParcelableExtra(key) as? T
  }
}
```

BleOperationsActivity

```
package com.punchthrough.blestarterappandroid

import android.annotation.SuppressLint
import android.app.Activity
import android.app.AlertDialog
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothGattCharacteristic
import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.view.MenuItem
import android.view.View
import android.view.WindowManager
import android.view.inputmethod.InputMethodManager
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import androidx.recyclerview.widget.SimpleItemAnimator
import com.punchthrough.blestarterappandroid.ble.ConnectionEventListener
import com.punchthrough.blestarterappandroid.ble.ConnectionManager
import
com.punchthrough.blestarterappandroid.ble.ConnectionManager.parcelableExtraC
ompat
import com.punchthrough.blestarterappandroid.ble.isIndicatable
import com.punchthrough.blestarterappandroid.ble.isNotifiable
import com.punchthrough.blestarterappandroid.ble.isReadable
import com.punchthrough.blestarterappandroid.ble.isWritable
import com.punchthrough.blestarterappandroid.ble.isWritableWithoutResponse
import com.punchthrough.blestarterappandroid.ble.toHexString
import
com.punchthrough.blestarterappandroid.databinding.ActivityBleOperationsBindin
g
import timber.log.Timber
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale
import java.util.UUID
import kotlin.math.pow

class BleOperationsActivity : AppCompatActivity() {
```

BleOperationsActivity

```
    private lateinit var binding: ActivityBleOperationsBinding
    private val device: BluetoothDevice by lazy {
        intent.parcelableExtraCompat(BluetoothDevice.EXTRA_DEVICE)
            ?: error("Missing BluetoothDevice from MainActivity!")
    }
    private val dateFormatter = SimpleDateFormat("MMM d, HH:mm:ss",
Locale.US)
    private val characteristics by lazy {
        ConnectionManager.servicesOnDevice(device)?.flatMap { service ->
            service.characteristics ?: listOf()
        } ?: listOf()
    }
    private val characteristicProperties by lazy {
        characteristics.associateWith { characteristic ->
            mutableListOf<CharacteristicProperty>().apply {
                if (characteristic.isNotifiable()) add(CharacteristicProperty.Notifiable)
                if (characteristic.isIndicatable()) add(CharacteristicProperty.Indicatable)
                if (characteristic.isReadable()) add(CharacteristicProperty.Readable)
                if (characteristic.isWritable()) add(CharacteristicProperty.Writable)
                if (characteristic.isWritableWithoutResponse()) {
                    add(CharacteristicProperty.WritableWithoutResponse)
                }
            }.toList()
        }
    }
    private val characteristicAdapter: CharacteristicAdapter by lazy {
        CharacteristicAdapter(characteristics) { characteristic ->
            showCharacteristicOptions(characteristic)
        }
    }
    private val notifyingCharacteristics = mutableListOf<UUID>()

    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        ConnectionManager.registerListener(connectionEventListener)

        binding = ActivityBleOperationsBinding.inflate(layoutInflater)

        setContentView(binding.root)
        supportActionBar?.apply {
            setDisplayHomeAsUpEnabled(true)
            setDisplayShowTitleEnabled(true)
```

BleOperationsActivity

```
            setDisplayShowTitleEnabled(true)
            title = getString(R.string.ble_playground)
        }

        binding.findDeviceButton.setOnClickListener{
            Intent(this, BleTracker::class.java).also {
                startActivity(it)
            }
        }

        binding.calibrationButton.setOnClickListener{
            Intent(this, Calibration::class.java).also {
                startActivity(it)
            }
        }

        binding.sensorReadingButton.setOnClickListener {
            Intent(this, SensorReadings::class.java).also {
                startActivity(it)
            }
        }



        setupRecyclerView()
        binding.requestMtuButton.setOnClickListener {
            val userInput = binding.mtuField.text
            if (userInput.isNotEmpty() && userInput.isNotBlank()) {
                userInput.toString().toIntOrNull()?.let { mtu ->
                    log("Requesting for MTU value of $mtu")
                    ConnectionManager.requestMtu(device, mtu)
                } ?: log("Invalid MTU value: $userInput")
            } else {
                log("Please specify a numeric value for desired ATT MTU (23-517)")
            }
            hideKeyboard()
        }
    }

    override fun onDestroy() {
        ConnectionManager.unregisterListener(connectionEventListener)
        ConnectionManager.teardownConnection(device)
        super.onDestroy()
    }
```

BleOperationsActivity

```
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            android.R.id.home -> {
                onBackPressed()
                return true
            }
        }
        return super.onOptionsItemSelected(item)
    }

    private fun setupRecyclerView() {
        binding.characteristicsRecyclerView.apply {
            adapter = characteristicAdapter
            layoutManager = LinearLayoutManager(
                this@BleOperationsActivity,
                RecyclerView.VERTICAL,
                false
            )
            isNestedScrollingEnabled = false

            itemAnimator.let {
                if (it is SimpleItemAnimator) {
                    it.supportsChangeAnimations = false
                }
            }
        }
    }

    @SuppressLint("SetTextI18n")
    private fun log(message: String) {
        val formattedMessage = "${dateFormatter.format(Date())}: $message"
        runOnUiThread {
            val uiText = binding.logTextView.text
            val currentLogText = uiText.ifEmpty { "Beginning of log." }
            binding.logTextView.text = "$currentLogText\n$formattedMessage"
            binding.logScrollView.post
{ binding.logScrollView.fullScroll(View.FOCUS_DOWN) }
        }
    }

    // Moving Average Filter - start
    val windowSizeMA = 5
```

BleOperationsActivity

```
val rssiValuesMA = mutableListOf<Int>()

fun updateRssiMovingAverage(rssi: Int): Double {
    if (rssiValuesMA.size >= windowSizeMA) {
        rssiValuesMA.removeAt(0)
    }
    rssiValuesMA.add(rssi)
    return rssiValuesMA.average()
}
// Moving Average Filter - end


// EMA filter - start
private var emaRssi: Double? = null
private val alphaE = 0.3  // Smoothing factor

private fun updateRssiEMA(newRssi: Int): Double {
    if (emaRssi == null) {
        emaRssi = newRssi.toDouble()
    } else {
        emaRssi = alphaE * newRssi + (1 - alphaE) * emaRssi!!
    }
    return emaRssi!!
}
// EMA filter - end

// Kalman Filter - start
val processNoise = 1.0
val measurementNoise = 1.0
var estimatedError = 1.0
var estimate = 0.0

fun updateRssiKalman(rssi: Int): Double {
    // Prediction step
    estimatedError += processNoise

    // Update step
    val kalmanGain = estimatedError / (estimatedError + measurementNoise)
    estimate += kalmanGain * (rssi - estimate)
    estimatedError *= (1 - kalmanGain)

    return estimate
}
```

BleOperationsActivity

```
    // Kalman Filter - end

    // Median Filter - start
    val windowSizeM = 5
    val rssiValuesM = mutableListOf<Int>()

    fun updateRssiMedian(rssi: Int): Double {
        if (rssiValuesM.size >= windowSizeM) {
            rssiValuesM.removeAt(0)
        }
        rssiValuesM.add(rssi)
        return rssiValuesM.sorted()[rssiValuesM.size / 2].toDouble()
    }
    //Median Filter - end

    @SuppressLint("SetTextI18n", "DefaultLocale")
    private fun onRssiUpdate(rssi: Int) {
        val d = 1
        val filteredRssiMedian = updateRssiMedian(rssi)
        Timber.i("Median Filter, at $d m, RSSI: $filteredRssiMedian")
        val filteredRssiKalman= updateRssiKalman(rssi)
        Timber.i("Kalman Filter, at $d m, RSSI: $filteredRssiKalman")
        val filteredRssiEMA= updateRssiEMA(rssi)
        Timber.i("EMA Filter, at $d m, RSSI: $filteredRssiEMA")
        val filteredRssiMovingAverage = updateRssiMovingAverage(rssi)
        Timber.i("Moving Average Filter, at $d m, RSSI:
$filteredRssiMovingAverage")

        Timber.i("No Filter (Raw value), at $d m, RSSI: $rssi")

        val rawRssi = rssi.toDouble()
        val distance = getDistanceFromRssi(filteredRssiEMA) // change the suitable
filtered rssi and calculate the distance (in this case, EMA filter is chosen)

        // Maintain a list of distances for averaging
        updateDistanceList(distance)

        // Calculate the mean of the recent distance values
        val averageDistance = getAverageDistance()

        binding.rssiValue.text = "${String.format("%.3f", filteredRssiEMA)} dBm" //
display the rssi value of a connected ble device
        binding.distanceValue.text = "${String.format("%.2f", averageDistance)}
```

BleOperationsActivity

```
meters"  // display the distance based on filtered rssi value
        Timber.i ("Distance estimated: $averageDistance")
    }

    // List to store recent distances for averaging
    private val distanceList = mutableListOf<Double>()
    private val MAX_DISTANCE_SAMPLES = 10  // Maximum number of
distance samples to store for averaging

    // Function to update the distance list with a new value
    private fun updateDistanceList(newDistance: Double) {
        if (distanceList.size >= MAX_DISTANCE_SAMPLES) {
            distanceList.removeAt(0)  // Remove the oldest distance value if the list is
full
        }
        distanceList.add(newDistance)  // Add the new distance value to the list
    }

    // Function to calculate the average of the stored distances
    private fun getAverageDistance(): Double {
        return if (distanceList.isNotEmpty()) {
            distanceList.average()  // Calculate and return the mean distance
        } else {
            0.0  // Return 0 if no distances are available
        }
    }

    object RssiData {
        // Shared RSSI values
        val rssiValues = mutableListOf(-66.5, -76.2, -88.7, -93.3) // dBm
        var rssi0 = rssiValues[0] // Reference RSSI at 1 meter distance
    }

    fun calculateDistanceConstant(): Double { // Calibration on the distance
estimation
        val distances = arrayOf(1.0, 2.0, 5.0, 10.0) // meters

        // Calculate log distances
        val logDistances = distances.map { Math.log10(it) }

        val adjustedRssi = RssiData.rssiValues.map { it - RssiData.rssi0 }

        // Calculate means
```

BleOperationsActivity

```
    val meanLogDist = logDistances.average()
    val meanRssi = adjustedRssi.average()

    // Calculate numerator and denominator for the slope
    var numerator = 0.0
    var denominator = 0.0

    for (i in logDistances.indices) {
        val logDistDiff = logDistances[i] - meanLogDist
        val rssiDiff = adjustedRssi[i] - meanRssi
        numerator += logDistDiff * rssiDiff
        denominator += logDistDiff * logDistDiff
    }

    // Calculate slope (b) and path loss exponent (n)
    val slope = -numerator / denominator
    val pathLossExponent = slope / 10.0

    Timber.i("Path Loss Exponent is calculated: $pathLossExponent")
    return pathLossExponent
}

private fun getDistanceFromRssi(rssi: Double): Double {
    val txPower = RssiData.rssi0  // RSSI value at 1 meter distance
    val n = calculateDistanceConstant()  // Path loss exponent
    return 10.0.pow((txPower - rssi) / (10 * n))
}

private fun phyToString(phy: Int?): String {
    return when (phy) {
        BluetoothDevice.PHY_LE_1M -> "LE 1M"
        BluetoothDevice.PHY_LE_2M -> "LE 2M"
        BluetoothDevice.PHY_LE_CODED -> "LE Coded"
        else -> "Unknown"
    }
}

@SuppressLint("SetTextI18n")
fun updatePHY (txPhy:Int?, rxPhy: Int?) {
    val txPhyString = phyToString(txPhy)
    val rxPhyString = phyToString(rxPhy)
    binding.txPhyValue.text = txPhyString
    binding.rxPhyValue.text = rxPhyString
```

BleOperationsActivity

```
    }

    private fun updateTemp (temp: Double)
    {
        val tempValue = temp.toString()
        Timber.i("Temperature value displayed: $tempValue")
    }

    private fun updateAcc(yAxisAcceleration: Int, xAxisAcceleration: Int,
zAxisAcceleration: Int)
    {
        val xAccValue = xAxisAcceleration.toString()
        val yAccValue = yAxisAcceleration.toString()
        val zAccValue = zAxisAcceleration.toString()

    private fun updateGyro (yAxisGyro: Int, xAxisGyro: Int, zAxisGyro: Int)
    {
        val xGyroValue = xAxisGyro.toString()
        val yGyroValue = yAxisGyro.toString()
        val zGyroValue = zAxisGyro.toString()
    }

    private fun showCharacteristicOptions(
        characteristic: BluetoothGattCharacteristic
    ) = runOnUiThread {
        characteristicProperties[characteristic]?.let { properties ->
            AlertDialog.Builder(this)
                .setTitle("Select an action to perform")
                .setItems(properties.map { it.action }.toTypedArray()) { _, i ->
                    when (properties[i]) {
                        CharacteristicProperty.Readable -> {
                            log("Reading from ${characteristic.uuid}")
                            ConnectionManager.readCharacteristic(device, characteristic)
                        }
                        CharacteristicProperty.Writable,
CharacteristicProperty.WritableWithoutResponse -> {
                            showWritePayloadDialog(characteristic)
                        }
                        CharacteristicProperty.Notifiable,
CharacteristicProperty.Indicatable -> {
                            if (notifyingCharacteristics.contains(characteristic.uuid)) {
                                log("Disabling notifications on ${characteristic.uuid}")
                                ConnectionManager.disableNotifications(device,
```

BleOperationsActivity

```
characteristic)
                        } else {
                            log("Enabling notifications on ${characteristic.uuid}")
                            ConnectionManager.enableNotifications(device,
characteristic)
                        }
                    }
                }
            }
            .show()
    }
}

    @SuppressLint("InflateParams")
    private fun showWritePayloadDialog(characteristic:
BluetoothGattCharacteristic) {
        val hexField = layoutInflater.inflate(R.layout.edittext_hex_payload, null) as
EditText
        AlertDialog.Builder(this)
            .setView(hexField)
            .setPositiveButton("Write") { _, _ ->
                with(hexField.text.toString()) {
                    if (isNotBlank() && isNotEmpty()) {
                        val bytes = hexToBytes()
                        log("Writing to ${characteristic.uuid}: ${bytes.toHexString()}")
                        ConnectionManager.writeCharacteristic(device, characteristic,
bytes)
                    } else {
                        log("Please enter a hex payload to write to ${characteristic.uuid}")
                    }
                }
            }
            .setNegativeButton("Cancel", null)
            .create()
            .apply {
                window?.setSoftInputMode(
                    WindowManager.LayoutParams.SOFT_INPUT_STATE_VISIBLE
                )
                hexField.showKeyboard()
                show()
            }
    }
```

BleOperationsActivity

```
private val connectionEventListener by lazy {
    ConnectionEventListener().apply {
        onDisconnect = {
            runOnUiThread {
                AlertDialog.Builder(this@BleOperationsActivity)
                    .setTitle("Disconnected")
                    .setMessage("Disconnected from device.")
                    .setPositiveButton("OK") { _, _ -> onBackPressed() }
                    .show()
            }
        }

        onCharacteristicRead = { _, characteristic, value ->
            log("Read from ${characteristic.uuid}: ${value.toHexString()}")
        }

        onCharacteristicWrite = { _, characteristic ->
            log("Wrote to ${characteristic.uuid}")
        }

        onMtuChanged = { _, mtu ->
            log("MTU updated to $mtu")
        }

        onCharacteristicChanged = { _, characteristic, value ->
            log("Value changed on ${characteristic.uuid}: ${value.toHexString()}")
        }

        onNotificationsEnabled = { _, characteristic ->
            log("Enabled notifications on ${characteristic.uuid}")
            notifyingCharacteristics.add(characteristic.uuid)
        }

        onNotificationsDisabled = { _, characteristic ->
            log("Disabled notifications on ${characteristic.uuid}")
            notifyingCharacteristics.remove(characteristic.uuid)
        }

        onReadRemoteRSSI = { _, rssi ->
            runOnUiThread {
                onRssiUpdate(rssi)
            }
        }
```

BleOperationsActivity

```
            onPhyUpdate = { _, txPhy, rxPhy ->
                runOnUiThread {
                    updatePHY(txPhy, rxPhy)
                }
            }
            onTempUpdate = { _, temp ->
                runOnUiThread {
                    updateTemp(temp)
                }
            }

            onSensorDataUpdate = { _, time, yAxisAcceleration, xAxisAcceleration,
zAxisAcceleration, yAxisGyro, xAxisGyro, zAxisGyro ->
                runOnUiThread {
                    updateAcc(yAxisAcceleration, xAxisAcceleration,
zAxisAcceleration)
                    Timber.i("Acceleration data is updated.")
                        updateGyro(yAxisGyro, xAxisGyro, zAxisGyro)
                        Timber.i("Gyroscope data is updated.")
                }
            }

        }
    }

    private enum class CharacteristicProperty {
        Readable,
        Writable,
        WritableWithoutResponse,
        Notifiable,
        Indicatable;

        val action
            get() = when (this) {
                Readable -> "Read"
                Writable -> "Write"
                WritableWithoutResponse -> "Write Without Response"
                Notifiable -> "Toggle Notifications"
                Indicatable -> "Toggle Indications"
            }
    }
```

BleOperationsActivity

```
   private fun Activity.hideKeyboard() {
      hideKeyboard(currentFocus ?: View(this))
   }

   private fun Context.hideKeyboard(view: View) {
      val inputMethodManager =
getSystemService(Activity.INPUT_METHOD_SERVICE) as
InputMethodManager
      inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
   }

   private fun EditText.showKeyboard() {
      val inputMethodManager =
getSystemService(Context.INPUT_METHOD_SERVICE) as
InputMethodManager
      requestFocus()
      inputMethodManager.showSoftInput(this,
InputMethodManager.SHOW_IMPLICIT)
   }

   private fun String.hexToBytes() =
      this.chunked(2).map
{ it.uppercase(Locale.US).toInt(16).toByte() }.toByteArray()

}
```

BleTracker

```
package com.punchthrough.blestarterappandroid

import android.annotation.SuppressLint
import android.bluetooth.BluetoothGatt
import android.bluetooth.BluetoothGattCharacteristic
import android.content.Context
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.os.Build
import android.os.Bundle
import android.widget.Toast
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
```

BleTracker

```
import com.punchthrough.blestarterappandroid.ble.ConnectionEventListener
import com.punchthrough.blestarterappandroid.ble.ConnectionManager
import com.punchthrough.blestarterappandroid.ble.DirectionFinder
import
com.punchthrough.blestarterappandroid.databinding.DistanceTrackerBinding
import timber.log.Timber
import java.util.UUID
import kotlin.math.pow

class BleTracker : AppCompatActivity() {

    private lateinit var sensorManager: SensorManager
    private lateinit var accelerometer: Sensor
    private lateinit var magnetometer: Sensor
    private var sensorEventListener: SensorEventListener? = null

    private var currentAzimuth: Float = 0f
    private val targetAngles = listOf(0, 45, 90, 135, 180, 225, 270, 315)
    private var currentTargetIndex = 0
    private var isRecording = false
    private var isWaitingForAngle = false
    private val rssiValuesByAngle = mutableMapOf<Int, MutableList<Int>>()

    private var latestRssi: Int = 0

    private var isDirectionButtonPressed = false

    private lateinit var binding: DistanceTrackerBinding

    private lateinit var directionFinder: DirectionFinder

    private var isPlayingSound = false
        set(value) {
            field = value
            runOnUiThread {
                binding.playSoundButtonText.text = if (value) "STOP" else "PLAY
SOUND"
                binding.playSoundButtonIcon.setImageResource(if (value)
R.drawable.stop_button else R.drawable.play_button)
            }
        }

    @RequiresApi(Build.VERSION_CODES.M)
```

BleTracker

```
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      ConnectionManager.registerListener(connectionEventListener)
      binding = DistanceTrackerBinding.inflate(layoutInflater)
      setContentView(binding.root)

      supportActionBar?.hide() // hide action bar

      val deviceName = ConnectionManager.deviceName
      Timber.i("Device Name is sent: $deviceName")
      binding.deviceNameText.text = deviceName

      binding.playSoundButton.setOnClickListener {
         if (isPlayingSound) {
            Toast.makeText(this, "Stop Ringing", Toast.LENGTH_SHORT).show()
            actionPlaySoundEnd()
         } else {
            Toast.makeText(this, "Start Ringing", Toast.LENGTH_SHORT).show()
            actionPlaySoundStart()
         }
      }

      binding.directionButton.setOnClickListener {
         startMeasurement()
      }
      setupCompass()
   }

   private fun setupCompass() {
      sensorManager = getSystemService(Context.SENSOR_SERVICE) as
SensorManager
      val accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
      val magnetometer =
sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)

      sensorEventListener = object : SensorEventListener {
         private val gravity = FloatArray(3)
         private val geomagnetic = FloatArray(3)

         @SuppressLint("SetTextI18n")
         override fun onSensorChanged(event: SensorEvent) {
            if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
```

BleTracker

```
                gravity[0] = event.values[0]
                gravity[1] = event.values[1]
                gravity[2] = event.values[2]
            } else if (event.sensor.type == Sensor.TYPE_MAGNETIC_FIELD) {
                geomagnetic[0] = event.values[0]
                geomagnetic[1] = event.values[1]
                geomagnetic[2] = event.values[2]
            }

            val R = FloatArray(9)
            val I = FloatArray(9)
            if (SensorManager.getRotationMatrix(R, I, gravity, geomagnetic)) {
                val orientation = FloatArray(3)
                SensorManager.getOrientation(R, orientation)
                currentAzimuth =
Math.toDegrees(orientation[0].toDouble()).toFloat()
                if (!isDirectionButtonPressed) {
                    // Update message when button is not pressed
                    binding.directionText.text = "Current angle:
${currentAzimuth.toInt()}°.\n" + "Tap DIRECTION to start."
                } else {
                    handleOrientationUpdate(currentAzimuth)
                }
            }
        }

        override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}
    }

    sensorEventListener?.let {
        sensorManager.registerListener(it, accelerometer,
SensorManager.SENSOR_DELAY_UI)
        sensorManager.registerListener(it, magnetometer,
SensorManager.SENSOR_DELAY_UI)
    }
}

@SuppressLint("SetTextI18n")
private fun handleOrientationUpdate(azimuth: Float) {
    // Normalize azimuth to be in range 0° to 360°
    currentAzimuth = (azimuth + 360) % 360

    if (currentTargetIndex < targetAngles.size) {
```

BleTracker

```
        val targetAngle = targetAngles[currentTargetIndex]

        // Display current angle and target angle
        binding.directionText.text = "Current Angle: ${currentAzimuth.toInt()}°\n"
+
            "Rotate to ${targetAngle}°."

        if (isWaitingForAngle) {
            if (isAngleCloseEnough(currentAzimuth, targetAngle)) {
                if (!isRecording) {
                    binding.directionText.text = "Reached $targetAngle°. Hold still,
recording RSSI..."
                    recordRssiForAngle(targetAngle)
                }
            } else {
                binding.directionText.text = "Current Angle:
${currentAzimuth.toInt()}°\n" + "Rotate to ${targetAngle}°."
            }
        }
    }
}

private fun isAngleCloseEnough(currentAngle: Float, targetAngle: Int):
Boolean {
    val tolerance = 5f // Tolerance in degrees
    val angleDifference = Math.abs(currentAngle - targetAngle)
    return angleDifference <= tolerance || angleDifference >= (360 - tolerance)
}

@SuppressLint("SetTextI18n")
private fun startMeasurement() {
    isDirectionButtonPressed = true
    currentTargetIndex = 0
    rssiValuesByAngle.clear()
    isWaitingForAngle = true
    binding.directionText.text = "Rotate to
${targetAngles[currentTargetIndex]}°."
}

private fun recordRssiForAngle(angle: Int) {
    isRecording = true
    val rssiValues = mutableListOf<Int>()
```

BleTracker

```
        for (i in 1..20) {
            val rssi = getCurrentRssi()
            rssiValues.add(rssi)
            Thread.sleep(100)
        }

        rssiValuesByAngle[angle] = rssiValues
        Timber.i("RSSI values for $angle°: $rssiValues")

        moveToNextTargetAngle()
    }

    @SuppressLint("SetTextI18n")
    private fun moveToNextTargetAngle() {
        currentTargetIndex++
        if (currentTargetIndex < targetAngles.size) {
            binding.directionText.text = "Rotate to
${targetAngles[currentTargetIndex]}°."
        } else {
            computeDirection()
        }
        isRecording = false
        isWaitingForAngle = true
    }

    @SuppressLint("SetTextI18n")
    private fun computeDirection() {
        var bestAngle = -1
        var highestMeanRssi = Double.NEGATIVE_INFINITY

        for ((angle, rssiList) in rssiValuesByAngle) {
            val meanRssi = rssiList.average()
            Timber.i("Mean RSSI at $angle°: $meanRssi")

            if (meanRssi > highestMeanRssi) {
                highestMeanRssi = meanRssi
                bestAngle = angle
            }
        }

        binding.directionText.text = "Estimated direction: $bestAngle°"
    }
```

BleTracker

```
   private fun getCurrentRssi(): Int {
      return latestRssi
   }

   override fun onPause() {
      super.onPause()
      sensorEventListener?.let {
         sensorManager.unregisterListener(it)
      }
   }

   override fun onResume() {
      super.onResume()
      setupCompass()
   }

   override fun onDestroy() {
      ConnectionManager.unregisterListener(connectionEventListener)
      super.onDestroy()
   }

   private fun actionPlaySoundStart() {
      val payload = byteArrayOf(0x04) // The value to start
      val characteristicUUID = UUID.fromString("ed0efb1a-9b0d-11e4-89d3-
123b93f75cba")
      writeToCharacteristic(payload, characteristicUUID)
      Timber.i("Attempting to write to characteristic to start playing sound")
      isPlayingSound = true
   }

   @RequiresApi(Build.VERSION_CODES.M)
   private fun actionPlaySoundEnd() {
      val payload = byteArrayOf(0x00) // The value to stop
      val characteristicUUID = UUID.fromString("ed0efb1a-9b0d-11e4-89d3-
123b93f75cba")
      writeToCharacteristic(payload, characteristicUUID)
      Timber.i("Attempting to write to characteristic to stop playing sound")
      isPlayingSound = false
   }

   @SuppressLint("MissingPermission")
   private fun writeToCharacteristic(payload: ByteArray, characteristicUUID:
UUID) {
```

BleTracker

```
        val gatt = ConnectionManager.getGatt()
        val device = gatt?.device
        val serviceUUID = UUID.fromString("ed0ef62e-9b0d-11e4-89d3-
123b93f75cba")
        val rCCharacteristic =
gatt?.getService(serviceUUID)?.getCharacteristic(characteristicUUID)

        Timber.i("Attempting to write to characteristic.")

        if (rCCharacteristic != null) {
            // Check if the characteristic is writable
            if ((rCCharacteristic.properties and
BluetoothGattCharacteristic.PROPERTY_WRITE) != 0) {
                rCCharacteristic.value = payload
                rCCharacteristic.writeType =
BluetoothGattCharacteristic.WRITE_TYPE_DEFAULT // Ensure write with
response

                val success = gatt?.writeCharacteristic(rCCharacteristic)
                if (success == true) {
                    Timber.i("Write operation successfully initiated for characteristic:
$characteristicUUID")
                } else {
                    Timber.e("Failed to initiate write operation. Retrying...")
                    retryWrite(gatt, rCCharacteristic, payload)
                }
            } else {
                Timber.e("Characteristic is not writable.")
            }
        } else {
            Timber.e("Characteristic not found. Service UUID: $serviceUUID,
Characteristic UUID: $characteristicUUID")
        }
    }

    @SuppressLint("MissingPermission")
    private fun retryWrite(gatt: BluetoothGatt?, characteristic:
BluetoothGattCharacteristic, payload: ByteArray, retryCount: Int = 3) {
        var attempt = 1
        while (attempt <= retryCount) {
            Timber.i("Retrying write operation (attempt $attempt)...")
            val success = gatt?.writeCharacteristic(characteristic)
            if (success == true) {
```

BleTracker

```
            Timber.i("Write operation succeeded on attempt $attempt.")
            break
        } else {
            Timber.e("Write operation failed on attempt $attempt.")
            Thread.sleep((1000 * (attempt * attempt)).toLong()) // Exponential
backoff
            attempt++
        }
    }
}



private val connectionEventListener by lazy {
    ConnectionEventListener().apply {
        onReadRemoteRSSI = { _, rssi ->
            runOnUiThread {
                onRssiUpdate(rssi)
                updateSignalStrengthUI(rssi)
                updateProximityIndicator(rssi)
            }
        }
    }
}



private var emaRssi: Double? = null
private val alphaE = 0.3  // Smoothing factor

private fun updateRssiEMA(newRssi: Int): Double {
    if (emaRssi == null) {
        emaRssi = newRssi.toDouble()
    } else {
        emaRssi = alphaE * newRssi + (1 - alphaE) * emaRssi!!
    }
    return emaRssi!!
}




@SuppressLint("SetTextI18n", "DefaultLocale")
private fun onRssiUpdate(rssi: Int) {
    val filteredRssiEMA= updateRssiEMA(rssi)
```

BleTracker

```
        latestRssi = filteredRssiEMA.toInt()
        val distance = getDistanceFromRssi(filteredRssiEMA) // change the suitable
filtered rssi and calculate the distance (in this case, EMA filter is chosen)

        // Maintain a list of distances for averaging
        updateDistanceList(distance)

        // Calculate the mean of the recent distance values
        val averageDistance = getAverageDistance()

        binding.distanceText.text = "${String.format("%.2f", averageDistance)}
meters"  // display the distance based on filtered rssi value
        Timber.i ("Distance estimated: $averageDistance")
    }

    // List to store recent distances for averaging
    private val distanceList = mutableListOf<Double>()
    private val MAX_DISTANCE_SAMPLES = 10  // Maximum number of
distance samples to store for averaging

    // Function to update the distance list with a new value
    private fun updateDistanceList(newDistance: Double) {
        if (distanceList.size >= MAX_DISTANCE_SAMPLES) {
            distanceList.removeAt(0)  // Remove the oldest distance value if the list is
full
        }
        distanceList.add(newDistance)  // Add the new distance value to the list
    }

    // Function to calculate the average of the stored distances
    private fun getAverageDistance(): Double {
        return if (distanceList.isNotEmpty()) {
            distanceList.average()  // Calculate and return the mean distance
        } else {
            0.0  // Return 0 if no distances are available
        }
    }

    private fun calculateDistanceConstant(): Double { // Calibration on the distance
estimation
        val distances = arrayOf(1.0, 2.0, 5.0, 10.0) // meters

        // Calculate log distances
```

BleTracker

```
        val logDistances = distances.map { Math.log10(it) }

        val adjustedRssi = BleOperationsActivity.RssiData.rssiValues.map { it -
BleOperationsActivity.RssiData.rssi0 }

        // Calculate means
        val meanLogDist = logDistances.average()
        val meanRssi = adjustedRssi.average()

        // Calculate numerator and denominator for the slope
        var numerator = 0.0
        var denominator = 0.0

        for (i in logDistances.indices) {
            val logDistDiff = logDistances[i] - meanLogDist
            val rssiDiff = adjustedRssi[i] - meanRssi
            numerator += logDistDiff * rssiDiff
            denominator += logDistDiff * logDistDiff
        }

        // Calculate slope (b) and path loss exponent (n)
        val slope = -numerator / denominator
        val pathLossExponent = slope / 10.0

        Timber.i("Path Loss Exponent is calculated: $pathLossExponent")
        return pathLossExponent
    }

    private fun getDistanceFromRssi(rssi: Double): Double {
        val txPower = BleOperationsActivity.RssiData.rssi0  // RSSI value at 1 meter
distance
        val n = calculateDistanceConstant()  // Path loss exponent
        return 10.0.pow((txPower - rssi) / (10 * n))
    }

    private fun getSignalStrength(rssi: Int): Pair<Int, String> {
        val (percentage, message) = when {
            rssi >= -20 -> Pair(100, "Excellent signal - The device is extremely close to
you.")
            rssi >= -30 -> Pair(95, "Very strong signal - The device is very close.")
            rssi >= -40 -> Pair(90, "Strong signal - The device is nearby.")
            rssi >= -50 -> Pair(85, "Good signal - The device is within a reasonable
range.")
```

BleTracker

```
        rssi >= -60 -> Pair(80, "Moderate signal - The device is within range.")
        rssi >= -70 -> Pair(75, "Fair signal - The device is at a moderate distance.")
        rssi >= -80 -> Pair(65, "Weak signal - The device is somewhat far.")
        rssi >= -85 -> Pair(55, "Poor signal - The device is quite distant.")
        rssi >= -90 -> Pair(45, "Very poor signal - The device is far away.")
        rssi >= -95 -> Pair(35, "Extremely weak signal - Try to move closer.")
        rssi >= -100 -> Pair(25, "Almost no signal - The device is very far.")
        rssi >= -105 -> Pair(15, "No signal - The device is likely out of range.")
        rssi >= -110 -> Pair(10, "No signal - Device not reachable.")
        rssi >= -115 -> Pair(5, "No signal - Device definitely out of range.")
        rssi >= -120 -> Pair(2, "No signal - Too far to detect.")
        rssi >= -125 -> Pair(0, "No signal - Device completely out of range.")
        else -> Pair(0, "No signal - Device completely out of range.")
    }

    return Pair(percentage, message)
}



private fun updateSignalStrengthUI(rssi: Int) {
    val (percentage, message) = getSignalStrength(rssi)
    binding.signalStrengthText.text = "$percentage%"
    binding.statusText.text = message
}

private fun updateProximityIndicator(rssi: Int) {
    // Define the range for RSSI
    val minRssi = -125
    val maxRssi = -25

    val normalizedRssi = maxOf(minRssi, minOf(maxRssi, rssi))
    val alpha = (normalizedRssi + 125) / 100f // Convert RSSI to a range of 0 to 1

    if (rssi < minRssi || rssi > maxRssi) {
        // Out of range: make all circles transparent
        binding.outerCircle.alpha = 0f
        binding.middleCircle.alpha = 0f
        binding.innerCircle.alpha = 0f
    } else {
        // Update circle visibility based on normalized alpha value
        binding.outerCircle.alpha = alpha
        binding.middleCircle.alpha = if (alpha > 0.33f) 1f else 0f
```

BleTracker

```
            binding.innerCircle.alpha = if (alpha > 0.66f) 1f else 0f
        }
    }
}
```

SensorReadings

```
package com.punchthrough.blestarterappandroid

import android.annotation.SuppressLint
import android.bluetooth.BluetoothDevice
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.os.Build
import android.os.Bundle
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
import com.punchthrough.blestarterappandroid.ble.ConnectionEventListener
import com.punchthrough.blestarterappandroid.ble.ConnectionManager
import com.punchthrough.blestarterappandroid.databinding.SensorDataBinding
import timber.log.Timber

class SensorReadings : AppCompatActivity(), SensorEventListener {

    private lateinit var binding: SensorDataBinding
    private lateinit var sensorManager: SensorManager
    private var accelerometer: Sensor? = null
    private var gyroscope: Sensor? = null
    private var isUsing1MPhy = true
        set(value) {
            field = value
            runOnUiThread {
                binding.phyButton.text = if (value) "1M PHY" else "CODED PHY"
            }
        }

    @RequiresApi(Build.VERSION_CODES.O)
    @SuppressLint("MissingPermission")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        ConnectionManager.registerListener(connectionEventListener)
```

SensorReadings

```
        binding = SensorDataBinding.inflate(layoutInflater)
        setContentView(binding.root)

        supportActionBar?.hide() // hide action bar

        // Initialize SensorManager and sensors
        sensorManager = getSystemService(SENSOR_SERVICE) as SensorManager
        accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
        gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)

        val deviceName = ConnectionManager.deviceName
        Timber.i("Device Name is sent: $deviceName")
        binding.deviceNameValue.text = deviceName

        binding.phyButton.setOnClickListener {
            val gatt = ConnectionManager.getGatt()
            if (isUsing1MPhy) {
                // Switch back to 1M PHY (PHY_LE_1M)
                gatt?.setPreferredPhy(
                    BluetoothDevice.PHY_LE_1M_MASK, // TX PHY
                    BluetoothDevice.PHY_LE_1M_MASK, // RX PHY
                    BluetoothDevice.PHY_OPTION_NO_PREFERRED // PHY options
                )
            } else {
                // Switch to coded PHY (PHY_LE_CODED)
                gatt?.setPreferredPhy(
                    BluetoothDevice.PHY_LE_CODED_MASK, // TX PHY
                    BluetoothDevice.PHY_LE_CODED_MASK, // RX PHY
                    BluetoothDevice.PHY_OPTION_S8 // PHY options
                )
            }
            isUsing1MPhy = !isUsing1MPhy
        }
    }

    override fun onResume() {
        super.onResume()
        // Register sensor listeners
        accelerometer?.also { sensor ->
            sensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL)
        }
```

SensorReadings

```
        gyroscope?.also { sensor ->
            sensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL)
        }
    }

    override fun onPause() {
        super.onPause()
        // Unregister sensor listeners to save battery
        sensorManager.unregisterListener(this)
    }

    override fun onSensorChanged(event: SensorEvent) {
        when (event.sensor.type) {
            Sensor.TYPE_ACCELEROMETER -> {
                val xAccValue = event.values[0].toString()
                val yAccValue = event.values[1].toString()
                val zAccValue = event.values[2].toString()

                binding.phoneAccXValue.text = xAccValue
                binding.phoneAccYValue.text = yAccValue
                binding.phoneAccZValue.text = zAccValue

                Timber.i("Phone Accelerometer - X: $xAccValue, Y: $yAccValue, Z:
$zAccValue")
            }
            Sensor.TYPE_GYROSCOPE -> {
                val xGyroValue = event.values[0].toString()
                val yGyroValue = event.values[1].toString()
                val zGyroValue = event.values[2].toString()

                binding.phoneGyroXValue.text = xGyroValue
                binding.phoneGyroYValue.text = yGyroValue
                binding.phoneGyroZValue.text = zGyroValue

                Timber.i("Phone Gyroscope - X: $xGyroValue, Y: $yGyroValue, Z:
$zGyroValue")
            }
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    }
```

SensorReadings

```
    private val connectionEventListener by lazy {
        ConnectionEventListener().apply {
            onTempUpdate = { _, temp ->
                runOnUiThread {
                    updateTemp(temp)
                }
            }

            onSensorDataUpdate =
                { _, timestamp, yAxisAcceleration, xAxisAcceleration,
zAxisAcceleration, yAxisGyro, xAxisGyro, zAxisGyro ->
                    runOnUiThread {
                        updateAcc(yAxisAcceleration, xAxisAcceleration,
zAxisAcceleration)
                        Timber.i("Board Acceleration data is updated.")
                        updateGyro(yAxisGyro, xAxisGyro, zAxisGyro)
                        Timber.i("Board Gyroscope data is updated.")
                    }
                }
            onTxUpdate = { _, txValue ->
                runOnUiThread {
                    updateTx(txValue)
                }
            }
        }
    }

    private fun updateTemp(temp: Double) {
        val tempValue = temp.toString()
        Timber.i("Temperature value displayed: $tempValue")
        binding.temperatureValue.text = tempValue
    }

    @SuppressLint("SetTextI18n")
    private fun updateTx(tx: Int) {
        val txValue = tx.toString()
        Timber.i("Transmit power (tx) value displayed: $txValue")
        binding.txValue.text = txValue
    }

    private fun updateAcc(yAxisAcceleration: Int, xAxisAcceleration: Int,
zAxisAcceleration: Int) {
```

SensorReadings

```
        val xAccValue = xAxisAcceleration.toString()
        val yAccValue = yAxisAcceleration.toString()
        val zAccValue = zAxisAcceleration.toString()
        binding.boardAccXValue.text = xAccValue
        binding.boardAccYValue.text = yAccValue
        binding.boardAccZValue.text = zAccValue
    }

    private fun updateGyro(yAxisGyro: Int, xAxisGyro: Int, zAxisGyro: Int) {
        val xGyroValue = xAxisGyro.toString()
        val yGyroValue = yAxisGyro.toString()
        val zGyroValue = zAxisGyro.toString()
        binding.boardGyroXValue.text = xGyroValue
        binding.boardGyroYValue.text = yGyroValue
        binding.boardGyroZValue.text = zGyroValue
    }

    override fun onDestroy() {
        ConnectionManager.unregisterListener(connectionEventListener)
        super.onDestroy()
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    >

    <Button
        android:id="@+id/scan_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="120dp"
        android:paddingRight="120dp"
        android:layout_marginTop="16dp"
        android:textStyle="bold"
        android:text="Start Scan"
        android:background="@drawable/button_background"
        android:backgroundTint="@android:color/holo_green_dark"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

    <TextView
        android:id="@+id/scan_results_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:paddingLeft="8dp"
        android:text="Scan results (tap to connect)"
        android:textAlignment="center"
        android:textSize="16sp"
        android:textStyle="bold"
        android:visibility="invisible"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/scan_button"
        />

    <View
        android:id="@+id/divider"
        android:layout_width="0dp"
        android:layout_height="2dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:background="?android:attr/listDivider"
        android:visibility="invisible"
        app:layout_constraintBottom_toTopOf="@id/scan_results_recycler_view"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/scan_results_title"
        />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/scan_results_recycler_view"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginTop="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/divider"
        tools:listitem="@layout/row_scan_result"
        />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_ble_operations.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".BleOperationsActivity">

    <LinearLayout
        android:id="@+id/mtu_container"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:orientation="horizontal"
        android:layout_marginTop="8dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <EditText
            android:id="@+id/mtu_field"
            android:hint="MTU value"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:importantForAutofill="no"
            android:inputType="number"
            android:digits="01234567890" />

        <Button
            android:id="@+id/request_mtu_button"
            android:text="Request MTU"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
```

activity_ble_operations.xml

```
</LinearLayout>

<TextView
    android:id="@+id/characteristics_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="Characteristics (tap to interact)"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/mtu_container"
    />

<LinearLayout
    app:layout_constraintTop_toBottomOf="@id/characteristics_title"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="0dp">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/characteristics_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:scrollbarFadeDuration="0"
        android:scrollbars="vertical"
        tools:listitem="@layout/row_characteristic" />

    <TextView
        android:id="@+id/log_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="12dp"
        android:layout_marginTop="16dp"
        android:text="Log"
        android:textSize="16sp"
        android:textStyle="bold"
        />
```

activity_ble_operations.xml

```
<ScrollView
    android:id="@+id/log_scroll_view"
    android:layout_width="match_parent"
    android:layout_height="80dp"
    android:padding="4dp">

    <TextView
        android:id="@+id/log_text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="bottom"
        android:scrollbars="vertical"
        android:layout_marginStart="16dp"
        />

</ScrollView>

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginStart="10dp"
    android:layout_marginTop="10dp">

    <TextView
        android:text="RSSI: "
        android:textSize="16sp"
        android:textStyle="bold"
        android:layout_height="20dp"
        android:layout_width="45dp"
        />

    <TextView
        android:id="@+id/rssi_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="XX dBm"
        android:textSize="14sp" />

</LinearLayout>

<LinearLayout
```

activity_ble_operations.xml

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp">

        <TextView
            android:text="Distance: "
            android:textSize="16sp"
            android:textStyle="bold"
            android:layout_height="20dp"
            android:layout_width="75dp"
            />

        <TextView
            android:id="@+id/distance_value"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="XX m"
            android:textSize="14sp" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp">

        <TextView
            android:text="TX PHY: "
            android:textSize="16sp"
            android:textStyle="bold"
            android:layout_height="20dp"
            android:layout_width="65dp"
            />

        <TextView
            android:id="@+id/tx_phy_value"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1M / Coded"
```

activity_ble_operations.xml

```
        android:textSize="14sp" />

</LinearLayout>

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginStart="10dp"
    android:layout_marginTop="10dp">

    <TextView
        android:text="RX PHY: "
        android:textSize="16sp"
        android:textStyle="bold"
        android:layout_height="20dp"
        android:layout_width="65dp"
        />

    <TextView
        android:id="@+id/rx_phy_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1M / Coded"
        android:textSize="14sp" />

</LinearLayout>

<Button
    android:id="@+id/sensor_reading_button"
    android:text="Sensor Reading"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:textColor="@android:color/white"
    android:textStyle="bold"
    android:background="@drawable/button_background"
    android:backgroundTint="@android:color/holo_orange_dark"
    android:paddingLeft="123dp"
    android:paddingRight="123dp"
    android:layout_gravity="center"
    app:layout_constraintTop_toBottomOf="@id/rx_phy_value"
    app:layout_constraintStart_toStartOf="parent"
```

activity_ble_operations.xml

```
        app:layout_constraintEnd_toEndOf="parent"/>

      <Button
        android:id="@+id/find_device_button"
        android:text="Find Device"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:background="@drawable/button_background"
        android:backgroundTint="@color/colorPrimary"
        android:paddingLeft="140dp"
        android:paddingRight="140dp"
        android:layout_gravity="center"
        app:layout_constraintTop_toBottomOf="@id/sensor_reading_button"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginBottom="16dp"/>
  </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

sensor_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="16dp"
  tools:context=".SensorReadings">

  <TextView
    android:id="@+id/device_name_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Device Name"
    android:textSize="35sp"
    android:textStyle="bold"
    android:textColor="@android:color/holo_green_light"
```

sensor_data.xml

```
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/temperature_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Temperature:"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintTop_toBottomOf="@id/device_name_value"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="16dp"/>

<TextView
    android:id="@+id/temperature_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Temperature Value"
    android:textSize="16sp"
    app:layout_constraintTop_toBottomOf="@id/device_name_value"
    app:layout_constraintStart_toEndOf="@id/temperature_title"
    android:layout_marginTop="16dp"
    android:layout_marginStart="8dp"/>

<TextView
    android:id="@+id/tx_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TX Power:"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintTop_toBottomOf="@id/temperature_value"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="16dp"/>

<TextView
    android:id="@+id/tx_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TX Power Value"
```

sensor_data.xml

```
        android:textSize="16sp"
        app:layout_constraintTop_toBottomOf="@id/temperature_value"
        app:layout_constraintStart_toEndOf="@id/tx_title"
        android:layout_marginTop="16dp"
        android:layout_marginStart="8dp"/>

    <TextView
        android:id="@+id/board_accelerometer_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Board Accelerometer"
        android:textSize="16sp"
        android:textStyle="bold"
        app:layout_constraintTop_toBottomOf="@id/tx_value"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="16dp"/>

    <TextView
        android:id="@+id/board_acc_x_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACCx:"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/board_accelerometer_title"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="8dp"/>

    <TextView
        android:id="@+id/board_acc_x_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="X Value"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/board_accelerometer_title"
        app:layout_constraintStart_toEndOf="@id/board_acc_x_title"
        android:layout_marginTop="8dp"
        android:layout_marginStart="8dp"/>

    <TextView
        android:id="@+id/board_acc_y_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACCy:"
```

sensor_data.xml

```
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/board_acc_x_title"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="8dp"/>

    <TextView
        android:id="@+id/board_acc_y_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Y Value"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/board_acc_x_title"
        app:layout_constraintStart_toEndOf="@id/board_acc_y_title"
        android:layout_marginTop="8dp"
        android:layout_marginStart="8dp"/>

    <TextView
        android:id="@+id/board_acc_z_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACCz:"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/board_acc_y_title"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="8dp"/>

    <TextView
        android:id="@+id/board_acc_z_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Z Value"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/board_acc_y_title"
        app:layout_constraintStart_toEndOf="@id/board_acc_z_title"
        android:layout_marginTop="8dp"
        android:layout_marginStart="8dp"/>

    <TextView
        android:id="@+id/board_gyroscope_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Board Gyroscope"
        android:textSize="16sp"
```

sensor_data.xml

```
    android:textStyle="bold"
    app:layout_constraintTop_toBottomOf="@id/board_acc_z_value"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="16dp"/>

<TextView
    android:id="@+id/board_gyro_x_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GYROx:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/board_gyroscope_title"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/board_gyro_x_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="X Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/board_gyroscope_title"
    app:layout_constraintStart_toEndOf="@id/board_gyro_x_title"
    android:layout_marginTop="8dp"
    android:layout_marginStart="8dp"/>

<TextView
    android:id="@+id/board_gyro_y_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GYROy:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/board_gyro_x_title"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/board_gyro_y_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Y Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/board_gyro_x_title"
```

sensor_data.xml

```
    app:layout_constraintStart_toEndOf="@id/board_gyro_y_title"
    android:layout_marginTop="8dp"
    android:layout_marginStart="8dp"/>

  <TextView
    android:id="@+id/board_gyro_z_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GYROz:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/board_gyro_y_title"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="8dp"/>

  <TextView
    android:id="@+id/board_gyro_z_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Z Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/board_gyro_y_title"
    app:layout_constraintStart_toEndOf="@id/board_gyro_z_title"
    android:layout_marginTop="8dp"
    android:layout_marginStart="8dp"/>

  <TextView
    android:id="@+id/phone_accelerometer_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Phone Accelerometer"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintTop_toBottomOf="@id/board_gyro_z_value"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="16dp"/>

  <TextView
    android:id="@+id/phone_acc_x_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ACCx:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_accelerometer_title"
```

sensor_data.xml

```
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="8dp"/>

    <TextView
        android:id="@+id/phone_acc_x_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="X Value"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/phone_accelerometer_title"
        app:layout_constraintStart_toEndOf="@id/phone_acc_x_title"
        android:layout_marginTop="8dp"
        android:layout_marginStart="8dp"/>

    <TextView
        android:id="@+id/phone_acc_y_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACCy:"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/phone_acc_x_title"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginTop="8dp"/>

    <TextView
        android:id="@+id/phone_acc_y_value"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Y Value"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/phone_acc_x_title"
        app:layout_constraintStart_toEndOf="@id/phone_acc_y_title"
        android:layout_marginTop="8dp"
        android:layout_marginStart="8dp"/>

    <TextView
        android:id="@+id/phone_acc_z_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ACCz:"
        android:textSize="14sp"
        app:layout_constraintTop_toBottomOf="@id/phone_acc_y_title"
        app:layout_constraintStart_toStartOf="parent"
```

sensor_data.xml

```
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/phone_acc_z_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Z Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_acc_y_title"
    app:layout_constraintStart_toEndOf="@id/phone_acc_z_title"
    android:layout_marginTop="8dp"
    android:layout_marginStart="8dp"/>

<TextView
    android:id="@+id/phone_gyroscope_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Phone Gyroscope"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintTop_toBottomOf="@id/phone_acc_z_value"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="16dp"/>

<TextView
    android:id="@+id/phone_gyro_x_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GYROx:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_gyroscope_title"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/phone_gyro_x_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="X Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_gyroscope_title"
    app:layout_constraintStart_toEndOf="@id/phone_gyro_x_title"
    android:layout_marginTop="8dp"
```

sensor_data.xml

```
    android:layout_marginStart="8dp"/>

<TextView
    android:id="@+id/phone_gyro_y_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GYROy:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_gyro_x_title"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/phone_gyro_y_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Y Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_gyro_x_title"
    app:layout_constraintStart_toEndOf="@id/phone_gyro_y_title"
    android:layout_marginTop="8dp"
    android:layout_marginStart="8dp"/>

<TextView
    android:id="@+id/phone_gyro_z_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="GYROz:"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_gyro_y_title"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="8dp"/>

<TextView
    android:id="@+id/phone_gyro_z_value"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Z Value"
    android:textSize="14sp"
    app:layout_constraintTop_toBottomOf="@id/phone_gyro_y_title"
    app:layout_constraintStart_toEndOf="@id/phone_gyro_z_title"
    android:layout_marginTop="8dp"
    android:layout_marginStart="8dp"/>
```

sensor_data.xml

```

    <Button
        android:id="@+id/phy_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1M PHY/ Coded PHY"
        android:textColor="@android:color/white"
        android:textStyle="bold"
        android:background="@drawable/button_background"
        android:backgroundTint="@android:color/holo_purple"
        android:paddingLeft="80dp"
        android:paddingRight="80dp"
        android:textSize="16dp"
        android:layout_gravity="center"
        app:layout_constraintTop_toBottomOf="@id/phone_gyro_z_value"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="20dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

distance_tracker.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".BleTracker">

    <TextView
        android:id="@+id/deviceNameText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="MyBLEDevice"
        android:textStyle="bold"
        android:textColor="@android:color/holo_green_light"
        android:textSize="25sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

distance_tracker.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/playSoundButton"
    android:layout_width="350dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:background="@drawable/button_background"
    android:clickable="true"
    android:focusable="true"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/deviceNameText">

    <ImageView
        android:id="@+id/playSoundButtonIcon"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:src="@drawable/play_button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/playSoundButtonText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="PLAY SOUND"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_marginTop="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/playSoundButtonIcon" />

</androidx.constraintlayout.widget.ConstraintLayout>

<Button
    android:id="@+id/directionButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:layout_marginBottom="10dp"
    android:background="@drawable/button_background"
```

distance_tracker.xml

```
        android:backgroundTint="@android:color/holo_purple"
        android:textColor="@android:color/white"
        android:text="Direction"
        android:paddingLeft="130dp"
        android:paddingRight="130dp"
        android:textSize="18sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/proximityIndicator"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.503"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/playSoundButton"
        app:layout_constraintVertical_bias="1.0"
        />

    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/proximityIndicator"
        android:layout_width="240dp"
        android:layout_height="240dp"
        android:layout_marginTop="50dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <View
            android:id="@+id/outerCircle"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@drawable/circle_background_outer"
            tools:layout_editor_absoluteX="0dp"
            tools:layout_editor_absoluteY="26dp"
            />

        <View
            android:id="@+id/middleCircle"
            android:layout_width="160dp"
            android:layout_height="160dp"
            android:background="@drawable/circle_background_middle"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
```

distance_tracker.xml

```
    <View
        android:id="@+id/innerCircle"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:background="@drawable/circle_background_inner"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <View
        android:id="@+id/centerDot"
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:background="@drawable/center_dot"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

<TextView
    android:id="@+id/distanceText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:text="2.5 m"
    android:textSize="35sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/proximityIndicator" />

<LinearLayout
    android:id="@+id/signalStrengthLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginTop="16dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
```

distance_tracker.xml

```
    app:layout_constraintTop_toBottomOf="@id/distanceText">

    <TextView
        android:id="@+id/signalStrengthTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Signal Strength:"
        android:textColor="@android:color/holo_green_light"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/signalStrengthText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="XX %"
        android:textColor="@android:color/holo_green_light"
        android:textSize="18sp"
        android:layout_marginStart="8dp" />

</LinearLayout>

<TextView
    android:id="@+id/statusText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="50dp"
    android:text="Message"
    android:textSize="16sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/signalStrengthLayout" />

<TextView
    android:id="@+id/directionText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="Direction Information"
    android:textSize="18sp"
    android:textColor="@android:color/holo_red_light"
    android:textStyle="bold|italic"
```

distance_tracker.xml

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/statusText" />

</androidx.constraintlayout.widget.ConstraintLayout>
```