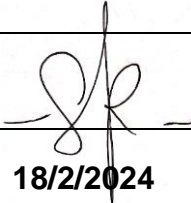




**SCHOOL OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITI TEKNOLOGI MARA**

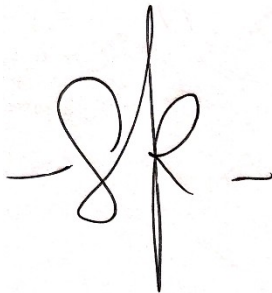
FINAL YEAR PROJECT 2

SUBMISSION OF PROJECT REPORT

STUDENT'S NAME	:	AHMAD AMIR MUHSIN BIN MOHAMAD NASIR
STUDENT'S UiTM ID	:	2021113573
COURSE CODE	:	ELE607
PROJECT TITLE		
SMART SENSOR CUSTOMIZATION USING BLE COMMUNICATION FOR HOME AUTOMATION		
This is to certify that the above student has submitted the project report to the project supervisor (SV).		
SV's NAME	:	MOHD SAUFY ROHMAD (MR)
SV's SIGNATURE	:	
DATE OF SUBMISSION	:	18/2/2024
<p>1) <i>Student needs to <u>fill in and submit this form and report</u> to his/her SV for endorsement/approval.</i></p> <p>2) <i>Student needs to return <u>this endorsed/approved form and report</u> to the FYP2 Coordinator.</i></p> <p>3) <i>SV needs to <u>submit evaluation marks</u> to the FYP2 Coordinator.</i></p>		

Document Information

Analyzed document	SMART SENSOR CUSTOMIZATION using BLE communication for home automation.docx (D185919190)
Submitted	2024-02-15 14:54:00 UTC+01:00
Submitted by	Ahmad Amir Muhsin Bin Mohamad Nasir
Submitter email	2021113573@student.uitm.edu.my
Similarity	3%
Analysis address	saufy.UiTM@analysis.ouriginal.com



Sources included in the report

SA	UNIVERSITI TEKNOLOGI MARA (UiTM) / REPORT_2019229738_EE242_JULY2023.pdf		
	Document REPORT_2019229738_EE242_JULY2023.pdf (D172576914)		1
	Submitted by: wannaifhaikal22@gmail.com		
	Receiver: adib.UiTM@analysis.ouriginal.com		
SA	UNIVERSITI TEKNOLOGI MARA (UiTM) / THESIS IQMAL DRAFT 1.pdf		
	Document THESIS IQMAL DRAFT 1.pdf (D185832643)		5
	Submitted by: 2021125955@student.uitm.edu.my		
	Receiver: mohdn829.UiTM@analysis.ouriginal.com		
SA	UNIVERSITI TEKNOLOGI MARA (UiTM) / technical_report iqram_draft 2.pdf		
	Document technical_report iqram_draft 2.pdf (D184310684)		3
	Submitted by: 2021101343@student.uitm.edu.my		
	Receiver: saufy.UiTM@analysis.ouriginal.com		

Entire Document

SMART SENSOR CUSTOMIZATION using BLE communication
for home automation
ahmad amir muhsin bin mohamad nasir

100%

MATCHING BLOCK 1/9

SA

REPORT_2019229738_EE242_JULY2023.pdf (D172576914)

SCHOOL OF ELECTRICAL ENGINEERING COLLEGE OF ENGINEERING UNIVERSITI TEKNOLOGI MARA MALAYSIA

SMART SENSOR CUSTOMIZATION using BLE communication for home automation
AHMAD amir muhsin bin mohamad nasir

**SMART SENSOR CUSTOMIZATION USING
BLE COMMUNICATION FOR HOME
AUTOMATION**

**AHMAD AMIR MUHSIN BIN MOHAMAD
NASIR**

**SCHOOL OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITI TEKNOLOGI MARA
MALAYSIA**

SMART SENSOR CUSTOMIZATION USING BLE COMMUNICATION FOR HOME AUTOMATION

**AHMAD AMIR MUHSIN BIN MOHAMAD
NASIR**

Final Year Project Report is submitted in partial fulfilment of the
requirements for the degree of
Bachelor of Engineering (Hons) Electronics Engineering

**SCHOOL OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITI TEKNOLOGI MARA
MALAYSIA**

AUTHOR'S DECLARATION

I declare that the work in this final year project report was carried out in accordance with the regulations of Universiti Teknologi MARA. It is original and is the results of my own work, unless otherwise indicated or acknowledged as referenced work. This final year project report has not been submitted to any other academic institution or non-academic institution for any degree or qualification.

I, hereby, acknowledge that I have been supplied with the Academic Rules and Regulations for Undergraduate, Universiti Teknologi MARA, regulating the conduct of my study and research.

Name of Student : Ahmad Amir Muhsin Bin Mohamad Nasir

Student I.D. No. : 2021113573


Programme : Bachelor of Engineering (Hons) Electronics Engineering

School : School of Electrical Engineering

Title of The Final Year Project Report : Smart Sensor Customization Using BLE Communication for Home Automation

Signature of Student : 

Date : February 2024

This Report is Approved by (Project Supervisor): 
(Mohd Saufy Rohmad)

Date : February 2024

ABSTRACT

This study embarks on an innovative effort to redefine home automation through the development of an advanced smart sensor system. Central to the research is the utilization of cutting-edge Bluetooth Low Energy (BLE) communication technology to establish seamless data transmission between sensors and controllers, facilitating real-time monitoring and control of home devices. The primary aim is to optimize energy consumption within the smart sensor system by implementing innovative deep sleep features. These features intelligently regulate sensor activity based on predefined timers, thereby conserving power and extending the operational lifespan of the system. A key aspect of the project involves integrating the Pydroid3 mobile application, which serves as a central interface for users to remotely control lamp switches, monitor environmental data, and enact dynamic automation rules. Methodologically, the project encompasses meticulous system design, hardware implementation, and software development utilizing popular platforms such as Arduino IDE and Pydroid3. Furthermore, the integration of the MQTT protocol facilitates efficient message exchange between system components, ensuring smooth operation and seamless communication. Through rigorous experimentation and analysis, the study demonstrates the successful establishment of BLE communication, the effectiveness of deep sleep-in energy optimization, and the seamless integration of the Pydroid3 app for enhanced user interaction and automation capabilities. These findings hold significant implications for the future of home automation, promising improved energy efficiency, heightened user engagement, and enhanced automation capabilities in modern living environments.

ACKNOWLEDGEMENT

I express my heartfelt gratitude to the Almighty God for His blessings and guidance throughout the journey of completing this thesis. His unwavering support has been a source of strength and inspiration, enabling me to overcome challenges and achieve success in this project.

I am deeply indebted to my supervisor, Mr. Mohd Saufy Rohmad, for his invaluable guidance, mentorship, and unwavering assistance throughout the project development. His expertise, insights, and constructive feedback have been instrumental in shaping this research and enhancing its quality. Additionally, I extend my sincere appreciation to my friends, Siti Sarah Binti Azhan and Muhammad Iqram Raziq bin Roshidi, for their invaluable contributions to the project, particularly in the areas of controller implementation and hardware solutions. Their dedication, collaboration, and support have played a significant role in the successful completion of this project.

Lastly, I would like to thank my family for their unwavering support, encouragement, and understanding throughout this journey. Their love, patience, and belief in my abilities have been a constant source of motivation and strength, driving me to strive for excellence in every part of this project.

TABLE OF CONTENTS

	Page
AUTHOR’S DECLARATION	ii
ABSTRACT	iii
ACKNOWLEDGEMENT	iv
TABLE OF CONTENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF PLATES	xi
LIST OF SYMBOLS	xii
LIST OF ABBREVIATIONS	xiii
LIST OF NOMENCLATURE	xiv
CHAPTER ONE INTRODUCTION	1
1.1 Research Background	1
1.2 Problem Statement	2
1.3 Objectives	2
1.4 Significance of Study	3
1.5 Scope of Work and Limitation	3
1.6 Thesis Organization	4
CHAPTER TWO LITERATURE REVIEW	6
2.1 Introduction	6
2.2 A Multi-Protocol IoT Gateway and WiFi/BLE Sensor Nodes for Smart Home and Building Automation: Design and Implementation	6
2.3 2.4 GHz BLE-based Smart Sensing System for Remote Monitoring of Health, Safety and Comfort at Workplace	7
2.4 A Dual Delay Timer Strategy for Optimizing Server Farm Energy	8
CHAPTER THREE RESEARCH METHODOLOGY	12
3.1 Introduction	12

3.2	System Overview	12
3.3	Flowchart	14
3.4	Block Diagram	15
3.5	Software and Hardware Setup	16
3.5.1	Software	17
3.5.2	Hardware	19
3.6	BLE Communication	26
3.6.1	Time Taken for Data Transmission	27
3.6.2	RSSI Value	29
3.6.3	Packet Loss Rate	31
3.7	Deep Sleep	32
3.7.1	Power Consumption During Active and Deep Sleep Mode	34
3.7.2	ANOVA Test	37
3.8	MQTT Protocol	39
3.9	Link-Up Communication Between All Peripheral Systems	40
3.9.1	Communication Between The Sender and Controller	41
3.9.2	Communication between the controller and MQTT Cloud Broker	42
3.9.3	Communication between MQTT Cloud Broker and Pydroid3	43
3.9.4	Communication between Pydroid3 and Switch	44
3.10	Integration of Pydroid3 Application	45
3.10.1	Data	45
3.10.2	Control	48
3.10.3	Auto	49
CHAPTER FOUR RESULTS AND DISCUSSION		50
4.1	Introduction	50
4.2	Result for Time Speed Data Transmission in BLE	50
4.3	Result for RSSI Value in BLE	51
4.4	Result for Loss Packet Rate in BLE	53
4.5	Result for Power Consumption During Active and Deep Sleep Mode	54
4.6	Results for Power Consumption with Different Sleep Duration	56
4.7	Result from Regression Analysis for Power Consumption in Different Sleep Durations	57

4.8	Pydroid3 Application	59
4.8.1	Data	59
4.8.2	Control	61
4.8.3	Auto	63
CHAPTER FIVE CONCLUSION		65
5.1	Conclusion	65
5.2	Recommendation for Future Work	65
REFERENCES		66
APPENDICES		70

LIST OF TABLES

Tables	Title	Page
Table 2.1	Research Gap	10
Table 3.1	The Measurement for Data Transmission	28
Table 3.2	The Timestamp for the 1 hour with the RSSI value recorded	30
Table 3.3		32
Table 3.4		36
Table 3.5	The Sleep Durations and Their Power Consumption	38
Table 3.6	The Timestamp for the 1 hour with the Temperature value recorded	46
Table 3.7	The Timestamp for the 1 hour with the Humidity value recorded	47
Table 3.8	The Timestamp for the 1 hour with the Motion Data recorded	47
Table 3.9	List of Automation Rules	49

LIST OF FIGURES

Figures	Title	Page
Figure 3.1:	Research Flowchart	15
Figure 3.2:	Block Diagram	16
Figure 3.3:	Arduino IDE	18
Figure 3.4:	Wokwi Simulator Software	18
Figure 3.5:	Interface of Pydroid3	19
Figure 3.6:	PIR Motion Sensor	20
Figure 3.7:	Temperature Sensor	20
Figure 3.8:	NodeMCU ESP32	21
Figure 3.9:	Switch Toggle 3-pin	21
Figure 3.10:	Green LED	22
Figure 3.11:	9V Battery	22
Figure 3.12:	10k Ohm Resistor	22
Figure 3.13:	Multimeter	23
Figure 3.14:	Overview Prototype for Temperature and Humidity Sensor	24
Figure 3.15:	Overview Prototype for PIR Sensor	24
Figure 3.16:	Schematic Diagram for Temperature and Humidity Sensor	25
Figure 3.17:	Schematic Diagram for PIR Sensor	26
Figure 3.18:	System BLE Communication of The Project	27
Figure 3.19:	Current Value for Active and Deep Sleep Mode for 5 Times Experiment	35
Figure 3.20:	The ANOVA Result	39
Figure 3.21:	Successfully Display Temperature Sensor Data from Sender (Above) to Receiver (Below)	41
Figure 3.22:	Successfully Display Motion Sensor Data from Sender (Above) to Receiver (Below)	42
Figure 3.23:	Establish a connection between the ESP32 controller and MQTT Cloud	42
Figure 3.24:	Temperature and Humidity Data Sensor (Below) Received at MQTT Cloud (ABove)	43
Figure 3.25:	PIR Sensor (Below) in MQTT Cloud (Above) is Received Value from the Controller	43

Figure 3.26: The MQTT Cloud Broker and the Subscriber (Pydroid3)	44
Figure 3.27: Control Button in Pydroid3	49
Figure 4.1: Transmission Duration for BLE Communication	51
Figure 4.2: RSSI Variation Over 1 hour	52
Figure 4.3: Packet Loss Rate Analysis	53
Figure 4.4: Average Power Consumption During Active and Deep Sleep Modes	55
Figure 4.5: Mean Power Consumption for Different Sleep Durations	56
Figure 4.6: Statistical Measures for Plot Analysis	58
Figure 4.7: Regression Analysis Graph	58
Figure 4.8: Temperature and Humidity Graph	60
Figure 4.9: PIR Sensor Reading	61
Figure 4.10: Output Display When Changes Made on the Control Button	62
Figure 4.11: Lamp is Turned On	62
Figure 4.12: When the User Pressed the Turn Off Button on Control Page	62
Figure 4.13: Lamp is Turned Off	63
Figure 4.14: User Select to Turn On Lamp Whenever Motion is Detected	63
Figure 4.15: User Selecting to Turn Off The Lamp Whenever There is No Motion Detected	64
Figure 4.16: Network Error Occurs during Data Transmission.	64

LIST OF PLATES

Plates	Title	Page
Plate 3.1	:System Overview for the Smart Sensor Customization Using BLE Communication For Home Automation	13

LIST OF SYMBOLS

Symbols

I Current

V Voltage

P Power

LIST OF ABBREVIATIONS

Abbreviations

BLE	Bluetooth Low Energy
MQTT	Message Queuing Telemetry Transport
LED	Light-Emitting Diode
DHT22	Digital Temperature and Humidity Sensor
HCSR501	Motion Sensor Module
RSSI	Received Signal Strength Indicator
PIR Sensor	Passive Infrared Sensor
UUID	Universally Unique IDentifier

LIST OF NOMENCLATURE

Nomenclatures

ESP32	low-power system-on-chip microcontrollers for IoT applications.
IoT	Internet of Things,
Jitter	Variability in packet arrival times in a network.

CHAPTER ONE

INTRODUCTION

1.1 Research Background

The project delves into the realm of home automation, leveraging modern technological advancements to enhance convenience and efficiency in household management. As society gravitates towards interconnected devices and smartphones, there's an increasing demand for automating routine tasks within homes. This initiative seeks to address this growing need by developing a smart sensor system that employs Bluetooth technology to establish seamless communication between sensors and a central controller [1]. Through this interconnected network, users can efficiently monitor and control various aspects of their home environment.

One pivotal aspect of this endeavour is the integration of the Pydroid3 mobile application, which acts as a user-friendly interface for managing the smart home system. With Pydroid3, users can remotely control lamp switches, monitor environmental data such as temperature and humidity, and enact dynamic automation rules tailored to their preferences. This integration not only enhances user convenience but also streamlines the process of interacting with the smart home system, making it accessible to individuals of varying technical proficiencies.

Energy conservation is another key focus of this project, achieved through the implementation of a deep sleep feature in the sensor system [2]. When sensors are not actively collecting data, they transition into a low-power mode to minimize energy consumption. This strategic approach not only contributes to environmental sustainability but also prolongs the operational lifespan of the sensor devices, ensuring long-term viability and cost-effectiveness.

Home automation is gaining traction due to its ability to simplify daily routines, enhance energy efficiency, and bolster security. By harnessing Bluetooth technology, the project aims to optimize communication between sensors and the controller, bolstering the responsiveness and reliability of the smart home system. The Pydroid3 app serves as a conduit for users to interact with the system effortlessly, empowering them to tailor their home environment to their preferences with ease. Ultimately, this project endeavours to bridge the gap between cutting-edge technology

and user-centric design, delivering a smart home solution that is intuitive, efficient, and accessible to all.

1.2 Problem Statement

The project addresses the need for a more efficient and user-friendly home automation system by integrating smart sensor technology and mobile application control. Traditional home management methods often lack adaptability and energy efficiency, relying on manual control and outdated systems. This project aims to bridge these gaps by developing a smart sensor system capable of real-time monitoring and remote control through a mobile application interface [3]. By leveraging Bluetooth communication and deep sleep features, the system optimizes energy usage while providing users with comprehensive control over various home devices and systems.

Through the integration of smart sensor technology and mobile application control, the project seeks to enhance user convenience, promote energy conservation, and streamline household management tasks. By creating a seamless and intuitive interface for monitoring and controlling home systems, the project aims to address the limitations of existing home automation solutions and provide users with greater flexibility and efficiency in managing their living spaces. Ultimately, the project aims to contribute to the advancement of smart home technology and improve the overall quality of life for users.

1.3 Objectives

This research aims to revolutionize home automation through the integration of cutting-edge technologies and smart sensor systems. The objectives of this study include:

- a) To implement BLE communication for efficient data transmission between the smart sensor and the controller, while integrating deep sleep features initiated by a timer mechanism to optimize energy consumption in the smart sensor system.

- b) To develop a smart sensor system integrated with the Pydroid3 mobile application, enabling remote control of the lamp switch, real-time monitoring of sensor data through graphical representation, and dynamic automation capabilities for automatic switch activation or deactivation based on predefined sensor-triggered rules.

1.4 Significance of Study

This project holds significant implications for advancing home automation systems to enhance convenience, efficiency, and energy conservation. By integrating Bluetooth Low Energy (BLE) communication, the system facilitates seamless data transmission between sensors and controllers, offering users a streamlined experience with minimal power consumption. The incorporation of deep sleep features further underscores the project's significance, as it optimizes energy usage during periods of inactivity, promoting sustainability and prolonging the operational lifespan of the sensor system.

Moreover, the development of a smart sensor system integrated with the Pydroid3 mobile application signifies a paradigm shift in user interaction with home automation technologies. Through this integration, users gain remote control over lamp switches, real-time access to environmental data, and the ability to implement dynamic automation rules, revolutionizing the way they manage and interact with their living spaces [4]. This project's significance extends beyond individual user experiences, as it lays the groundwork for scalable, adaptable, and responsive smart home solutions that can contribute to energy efficiency initiatives and improve overall quality of life.

1.5 Scope of Work and Limitation

The scope of work for this project encompasses several key components aimed at achieving the research objectives. Firstly, it involves the design and development of a smart sensor system utilizing Bluetooth Low Energy (BLE) communication for efficient data transmission. This system will integrate various sensors, such as temperature and motion sensors, along with a microcontroller to collect environmental data. Additionally, the implementation of deep sleep features initiated by a timer

mechanism will optimize energy consumption, enhancing the system's sustainability and longevity. The development process will include hardware and software integration, sensor calibration, and testing to ensure the system's reliability and functionality.

Furthermore, the scope extends to the integration of the Pydroid3 mobile application, which serves as a central interface for users to interact with the smart sensor system. Through the application, users can remotely control lamp switches, monitor real-time environmental data, and enact dynamic automation rules based on predefined triggers. This integration involves software development, user interface design, and compatibility testing across various mobile platforms. Additionally, the scope encompasses data analysis to evaluate the system's performance, including power consumption, data transmission efficiency, and user interaction.

However, it is important to acknowledge the limitations of this project. These may include constraints related to budget, time, and resources, which could impact the depth and scope of the research. Additionally, technical challenges or unforeseen issues during the development and implementation phases may affect the project's outcomes. Understanding and addressing these limitations will be crucial for managing expectations and ensuring the project's success within the defined scope.

1.6 Thesis Organization

This thesis is structured into five main chapters, each addressing specific aspects of the research project. In Chapter 1, the background, objectives, problem statement, significance of the study, scope of work, and limitations are outlined. This chapter provides a comprehensive overview of the project's context and aims, setting the stage for the subsequent chapters. Chapter 2 delves into an in-depth literature review, where previous works are analyzed, and the need for the study is explained. This chapter involves a thorough literature search using relevant citations and references from various validated resources to establish the research's theoretical framework.

Moving on to Chapter 3, the focus shifts to outlining, explaining, and justifying the complete methodological components used in the project. This includes detailing the system overview, system flowchart, block diagram project, software and hardware approach, data collection for BLE communication, deep sleep analysis,

MQTT broker, and integration of Pydroid3. Chapter 4 presents the significant results based on the project objectives, accompanied by proper discussion and analysis. This includes the time taken for data transmission, RSSI value analysis, loss packet rate in BLE communication, power consumption during active and deep sleep modes, and the analysis of mean power consumption with different sleep durations. Additionally, this chapter integrates the Pydroid3 app with its three interfaces: data, control, and auto.

In Chapter 5, the findings from the project are concluded, addressing all objectives and providing recommendations for future work. This chapter synthesizes the research outcomes, highlights key insights, and offers suggestions for further research directions. Finally, the thesis includes a references section, listing all the sources cited throughout the document. Each chapter is meticulously structured to provide a coherent and comprehensive exploration of the research project, from its inception to its conclusions and future implications.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The literature review section of this research project provides a comprehensive examination of existing studies, theories, and findings relevant to the field of smart sensor systems and home automation. It serves as a foundational element in understanding the current state of knowledge, identifying gaps, and informing the research objectives. By synthesizing information from various scholarly sources, including academic journals, conference papers, and reputable publications, this literature review aims to contextualize the research within the broader academic discourse. Furthermore, it offers insights into key concepts, methodologies, and technologies employed in previous studies, laying the groundwork for the subsequent analysis and discussion of the research project.

2.2 A Multi-Protocol IoT Gateway and WiFi/BLE Sensor Nodes for Smart Home and Building Automation: Design and Implementation

The paper, authored by Kanitkorn Khanchuea and Rawat Siripokarpirom from the Department of Electrical and Computer Engineering at the Faculty of Engineering, KMUTNB, Bangkok, Thailand, presents a comprehensive analysis and implementation of a multi-protocol gateway and multi-hop wireless network for smart home and building automation applications. It begins by providing a background on the advantages of wireless networks over wired ones, emphasizing reduced installation costs and increased flexibility.

Additionally, the paper highlights the importance of multi-hop wireless networks in extending connectivity range, addressing issues such as reliability and security. In discussing previous and existing trends, the authors examine various wireless communication technologies like ZigBee, Z-Wave, 6LowPAN, Thread, and Bluetooth Mesh, commonly used in smart home applications [5]. They also acknowledge the popularity of WiFi SoCs like ESP8266 and ESP32 from Espressif Systems in IoT applications due to their low cost and high performance. However, the

authors note limitations in previous research, such as reliability concerns and limited support for multiple protocols. To address these limitations, the paper proposes a multi-protocol gateway that supports both wired and wireless connectivity and can handle multiple protocols, including ESP-NOW and ZigBee. It also explores the use of the ESP-NOW protocol for constructing a low-power multi-hop wireless network.

Furthermore, the authors suggest integrating Bluetooth Low Energy (BLE) advertising beacons for discovering neighbouring devices, enhancing the network formation process. The authors' contribution lies in effectively addressing the limitations of previous research by proposing a comprehensive solution that leverages existing technologies while mitigating their drawbacks. By utilizing the ESP-NOW protocol and BLE for network formation, the proposed system offers a robust and efficient solution for smart home and building automation applications.

Moreover, the experimental results provided by the authors offer quantitative evidence of the system's performance, validating its effectiveness in real-world scenarios. In conclusion, the authors present a significant contribution to the field of smart home and building automation by offering a well-designed and implemented solution. They not only address the limitations of previous research but also demonstrate its practical viability through experimentation. Additionally, the authors open avenues for future research, such as extending the proposed architecture with mesh network routing capabilities, indicating a forward-looking approach.

2.3 2.4 GHz BLE-based Smart Sensing System for Remote Monitoring of Health, Safety and Comfort at Workplace

The paper, authored by R.Colella et al presents a smart sensing system aimed at revolutionizing the monitoring of health, safety, and comfort in workplace settings. This system represents a significant advancement in the field of sensor technology, leveraging the latest innovations to collect and transmit data on workers' physiological states. By integrating a diverse array of sensors into a single board, the system offers a comprehensive approach to monitoring vital parameters such as heart rate, blood oxygen saturation, skin temperature, vibration, and motion levels. This multifaceted data collection enables a more nuanced understanding of workers' well-being and provides valuable insights into potential health and safety risks in the workplace.

Previous trends in workplace monitoring have often relied on disparate sensor

technologies, leading to fragmented data collection and analysis. The limitations of these approaches have underscored the need for a more integrated and efficient system, capable of providing real-time, accurate, and actionable information to improve worker safety and comfort. The proposed smart sensing system addresses these shortcomings by consolidating innovative sensors and leveraging a remote Bluetooth Low Energy (BLE) interface for seamless data transmission to a monitoring mobile app. This streamlined approach not only enhances data accuracy and reliability but also facilitates timely interventions to mitigate potential health and safety hazards in the workplace.

One of the key contributions of this research lies in its ability to overcome the challenges faced by previous monitoring systems, particularly in the areas of temperature monitoring accuracy and fatigue assessment. By incorporating advanced sensors like the MAX30102 PPG sensor for heart rate and blood oxygen saturation measurements and the IMU LSM6DS3 for motion detection and posture assessment, the smart sensing system offers a more holistic and precise monitoring solution [6]. This comprehensive approach not only enhances the quality of data collected but also enables a more nuanced understanding of workers' physiological responses to their work environment, thereby empowering organizations to proactively address health and safety concerns.

In conclusion, the proposed smart sensing system represents a significant advancement in workplace monitoring technology, offering a comprehensive and cost-effective solution to enhance worker health, safety, and comfort. By leveraging state-of-the-art sensors and innovative data transmission methods, this system provides a robust framework for real-time monitoring and analysis of vital physiological parameters. The integration of advanced sensors and remote monitoring capabilities not only improves the accuracy and reliability of data collection but also enables organizations to take proactive measures to ensure the well-being of their workforce. This research sets a new standard in workplace monitoring, paving the way for more effective and responsive strategies to promote a safe and healthy work environment for employees.

2.4 A Dual Delay Timer Strategy for Optimizing Server Farm Energy

The paper "A Dual Delay Timer Strategy for Optimizing Server Farm Energy"

by the author Fan Yao et al [7], presents a comprehensive analysis of energy optimization in server farms, a critical area of research due to the increasing energy consumption of data centres. The background of the research highlights the rapid growth of energy consumption in server farms, emphasizing the need for effective energy management techniques that consider workload characteristics. The authors discuss the limitations of traditional approaches, such as over-provisioning servers and ineffective system-wide energy management, leading to suboptimal energy utilization. This sets the stage for the exploration of novel strategies to address energy inefficiencies in server farms.

Previous research in the field has primarily focused on techniques like Dynamic Voltage Frequency Scaling (DVFS) and processor sleep states to achieve energy savings. While these approaches have shown some effectiveness, the paper identifies the need for more advanced energy optimization methods tailored to server farm environments. The authors highlight the drawbacks of existing strategies, such as high-performance overheads when transitioning processors from low-power states to active states. By introducing the concept of dual delay timers in conjunction with processor sleep states, the proposed work aims to overcome these limitations and maximize energy savings across various workloads.

The proposed Dual Delay Timer strategy offers a significant advancement in energy optimization for server farms. By orchestrating the entry and exit from low-power states more intelligently, the authors demonstrate substantial energy savings compared to traditional approaches. The research findings indicate that the new technique can achieve up to 71% energy savings over naive energy management methods, showcasing its potential to address the energy inefficiencies prevalent in server farm environments. Through this innovative approach, the authors contribute to bridging the gap between existing energy management practices and the growing demand for more efficient and sustainable server farm operations.

In conclusion, the paper by the authors presents a compelling argument for the adoption of the Dual Delay Timer strategy to optimize server farm energy consumption. By building upon the limitations of previous research and proposing a novel approach that leverages dual delay timers and processor sleep states, the study offers a promising solution to the energy challenges faced by modern data centres.

Table 2.1
Research Gap

No	Title	Parameter measured/System Platform	Input/Output	Area focused	Limitations
1	A Multi-Protocol IoT Gateway and WiFi/BLE Sensor Nodes for Smart Home and Building Automation: Design and Implementation	The average round trip time, throughput, and jitter/ a single-board computer (SBC) running an embedded Linux operating system	Data from the sensor nodes, control commands from the gateway, and network configuration settings/ communication between the sensor nodes, gateway, and external devices	Smart home and building automation applications,	The restriction to a simple linear network typology with a limited number of nodes (up to 6 nodes). Another limitation of the study was the use of a tree-based network model, which restricted the network topology to a specific structure with the coordinator as the root node and routers as intermediate and end nodes
2	2.4 GHz BLE-based Smart Sensing System for Remote Monitoring of Health, Safety and Comfort at Workplace	Heart Rate (HR) and Blood Oxygen Saturation (SpO2), Skin Temperature, Motion Levels and Fatigue Assessment/MSP430 and mobile app	Physiological Parameters, Motion Data, Command Messages/ Processed Physiological Data, Alerts and Notifications, Data Transmission	Worker Health Monitoring, Worker Safety, Worker Comfort	The effectiveness of the system relies on user compliance and proper sensor placement. If users do not wear or position the sensors correctly, it may lead to inaccurate readings and affect the overall reliability of the monitoring data. While the system collects a wide range of physiological and motion data, the interpretation and analysis of this data may require expertise or specialized knowledge. Ensuring that the data is accurately interpreted and actionable insights are derived from it could be a challenge.

No	Title	Parameter measured/System Platform	Input/Output	Area focused	Limitations
3	A Dual Delay Timer Strategy for Optimizing Server Farm Energy	Analyze the energy savings achieved through the proposed Dual Delay Timer strategy in server farm environments/workload generator, a server power state/performance manager, and a server farm job handler/load balancer	various workloads Google search, Apache, Mail, and DNS jobs, both synthetic and real/ the measurement of energy savings achieved, job latencies, and system performance metrics	The primary focus of the research was to explore techniques that leverage processor sleep states augmented with dual delay timer	While simulations provide a controlled environment for testing hypotheses, the results may not fully capture the complexities and nuances of real-world server farm operations. While the research investigated the scalability of the Dual Delay Timer strategy by varying the size of the server farm from 20 to 100 servers, the scalability challenges in larger server farms with hundreds or thousands of servers were not explicitly addressed.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

The methodology adopted for this project encapsulates a systematic approach aimed at realizing the project's objectives effectively. At its core, the methodology revolves around comprehensively outlining the operation methods necessary to achieve the desired outcomes. It encompasses various key components, each contributing to the overall success of the endeavour. The methodology encompasses a system overview, system flowchart, block diagram project, software and hardware approach, data collection for ble communication, deep sleep analysis, MQTT broker, and integration of Pydroid3.

3.2 System Overview

The system overview serves as a comprehensive guide outlining the operational framework and functionality of the smart sensor system. It provides a holistic view of the system's components, interactions, and objectives, offering invaluable insights into its design and operation. By delineating the sequence of actions and communication pathways within the system, the overview facilitates a clear understanding of its purpose and functionality.

The system overview goes into the complexities of the smart sensor customisation project, which uses BLE communication for home automation. It provides a detailed description of the steps involved in converting sensor-detected environmental data into actionable commands for home devices. The plate below visually encapsulates the process for data transmission and interpretation within the project's framework.

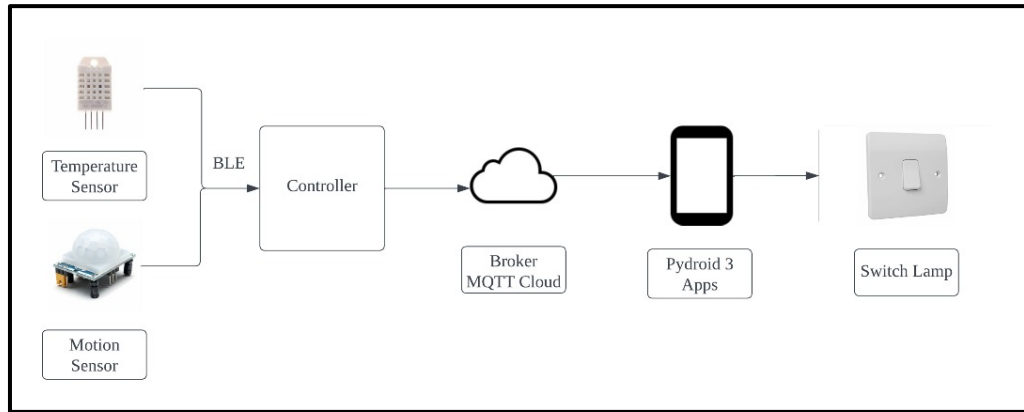


Plate 3.1: System Overview for the Smart Sensor Customization Using BLE Communication For Home Automation

The smart sensor system operates through a coordinated sequence of actions designed to monitor and manage environmental conditions effectively. The system initiates when the user activates the sensor by toggling a switch on the sensor box, signalling the beginning of data collection. This data encompasses temperature, humidity, and motion detection, facilitated by sensors embedded within the device. The sensor's active state is indicated by the illumination of an LED, providing a visual cue of its operational status. Once data collection is underway, the sensor wirelessly transmits this information to the controller using Bluetooth Low Energy (BLE) technology. This transmission involves two ESP32 devices, with one serving as the sender and the other as the receiver [8]. The receiver, located within the controller, processes and displays the sensor data, providing users with real-time insights into environmental conditions.

Additionally, the controller is equipped with another ESP32 module responsible for converting BLE-formatted data into WiFi-compatible data. This transformation enables communication with the MQTT (Message Queuing Telemetry Transport) broker, a central hub that facilitates the exchange of data between the sensor system and other network components. The MQTT broker acts as a conduit, ensuring seamless communication and data transmission across the network.

Furthermore, the sensor data received by the MQTT broker is published to the Pydroid3 application, serving as the end-user client. Within the Pydroid3 interface, users have access to three distinct interfaces, each fulfilling a specific function while interconnected to provide a cohesive user experience. The Control interface allows users to remotely operate devices such as lamps, providing convenient control over the smart home environment. The Auto interface enables users to define automation

rules based on predefined conditions, such as motion detection, streamlining device management and enhancing efficiency [9]. Finally, the Data interface presents graphical representations of temperature, humidity, and motion sensor data over time, empowering users with valuable insights into environmental trends and fluctuations.

In summary, the smart sensor system encompasses a comprehensive operational framework, leveraging BLE technology, MQTT communication, and the Pydroid3 application to monitor and manage environmental conditions efficiently. Through seamless integration and user-friendly interfaces, the system provides users with real-time data visualization and control capabilities, enhancing convenience, comfort, and energy efficiency within the smart home environment.

3.3 Flowchart

The system flowchart serves as a visual representation of the sequential processes underlying the smart sensor system for home automation. It outlines the journey of data, starting from the sensor's capture of environmental parameters such as temperature, humidity, and motion. Subsequently, this data is transmitted to the receiver housed within the controller unit. The controller, acting as an intermediary, publishes the received data to the MQTT cloud broker, facilitating its dissemination to the Pydroid3 application. Finally, the Pydroid3 application interprets the received data to trigger the activation or deactivation of the lamp switch, thereby affecting control over the home automation system.

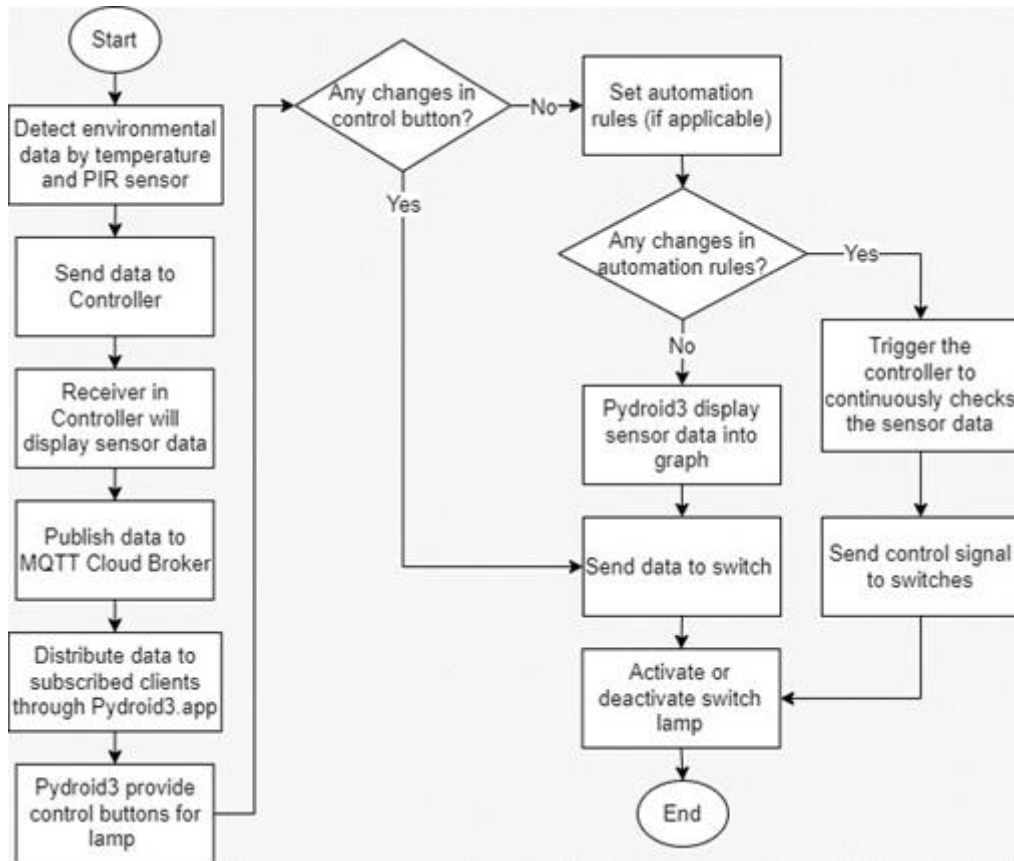


Figure 3.1: Research Flowchart

3.4 Block Diagram

The block diagram is a schematic representation of the Smart Sensor Customisation Using Ble Communication For Home Automation project, providing a detailed breakdown of its components and their linkages. At its core, the block diagram depicts the system's architecture, including the flow of information and control signals between hardware and software components. The figure below represents the block diagram for the entire system of smart sensors.

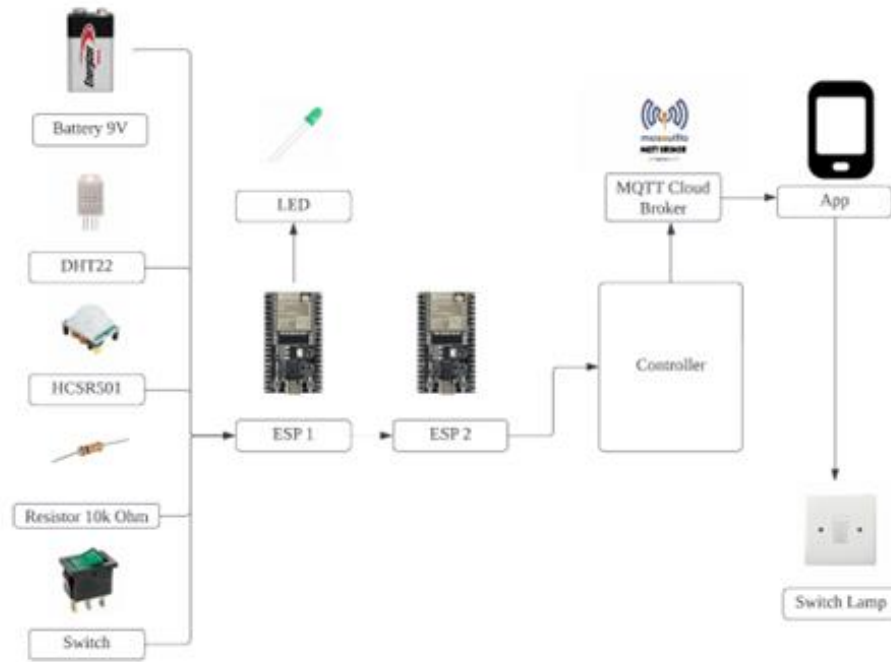


Figure 3.2: Block Diagram

The block diagram depicts the overall system design, which includes input components such as the DHT22 temperature sensor, the HCSR501 PIR sensor, a 9V battery, a 10k resistor, and a switch pin connected to the microcontroller. These input components work together to collect environmental data and human inputs, making the smart sensor system more efficient.

Furthermore, the diagram shows the communication paths between the sensor, controller, MQTT Cloud broker, and Pydroid3 application. Once acquired, sensor data is transmitted from ESP 1 to ESP 2, which serves as the receiver, using BLE communication. The controller unit's receiver then connects to the MQTT Cloud broker, allowing sensor data to be distributed to subscribing applications like Pydroid3.

Moreover, the block diagram displays the control flow from the application interface to the lamp switch, which is enabled by remote control features built into the Pydroid3 program. This interface allows users to remotely activate or deactivate the lamp switch, allowing them control over the home automation system.

3.5 Software and Hardware Setup

The software and hardware components of the smart sensor project represent the foundational elements that enable the system's functionality and operation. In

essence, the software comprises the programs, algorithms, and firmware that govern the behaviour and interaction of these hardware components, while the hardware encompasses the physical devices and circuitry, including sensors, microcontrollers, and peripheral components. Together, the software and hardware components form an integrated system that facilitates the collection, processing, and transmission of environmental data, as well as the control and automation of home devices.

3.5.1 Software

The software that was used in this project includes Arduino IDE, Wokwi, and Pydroid3. Arduino IDE was utilized for programming the microcontrollers and managing the firmware of the smart sensor system. Wokwi facilitated the simulation and testing of the system's functionality in a virtual environment. Pydroid3, on the other hand, served as the platform for developing and executing Python code to control and interface with the smart sensor system on a mobile device.

3.5.1.1 Arduino IDE

The Arduino IDE was essential in the development and programming of the microcontrollers that form the backbone of the smart sensor system. With the Arduino IDE, developers could write, compile, and upload code to microcontrollers, allowing them to control various hardware components and perform predetermined tasks [10]. The seamless integration of libraries like the DHT22 Adafruit Unified Sensor Library and the BLE Device Library sped up the development process by offering pre-written code modules for connecting with sensors and implementing Bluetooth Low Energy (BLE) protocols.

Additionally, the use of libraries like the Adafruit NeoPixel Library and the PubSubClient Library enhanced the functionality of the system by enabling control of RGB LEDs and facilitating communication with the MQTT broker, respectively. Arduino IDE's user-friendly interface and extensive library support accelerated development efforts and contributed to the successful implementation of the smart sensor system. Below is the figure of Arduino IDE interface software tools.



Figure 3.3: Arduino IDE

3.5.1.2 Wokwi

Wokwi was an effective tool for modelling and testing the functionality of the smart sensor system in a virtual environment. Using Wokwi's simulation capabilities, developers may evaluate system behaviour, detect potential flaws, and fine-tune code before delivering it to physical hardware. Wokwi's user-friendly interface and real-time feedback systems made quick troubleshooting and optimisation of the system possible, ultimately improving its stability and performance. Furthermore, the Wokwi Simulator is used in this project rather than Proteus Software since the NodeMCU ESP32 component lacks a running file code to mimic the progress project. The file code only supports Arduino components. Below is the figure showing the Wokwi Simulator interface.



Figure 3.4: Wokwi Simulator Software

3.5.1.3 Pydroid3

Pydroid3, a Python-integrated development environment (IDE) tailored for mobile platforms, played a crucial role in developing the control interface for the smart sensor system. This versatile IDE enabled developers to write, execute, and debug Python code directly on mobile devices, facilitating rapid prototyping and testing. Pydroid3's support for external libraries such as Matplotlib and NumPy expanded its capabilities, allowing developers to visualize data in real-time graphs and

perform complex mathematical computations seamlessly.

Additionally, Pydroid3's integration with the Paho MQTT library facilitated communication with the MQTT broker, enabling the seamless exchange of data between the sensor system and the Pydroid3 application. By leveraging these libraries and Pydroid3's intuitive interface, developers could create a user-friendly control interface that enhanced the functionality and usability of the smart sensor system. The diagram below represents the Pydroid3 Interface system.



Figure 3.5: Interface of Pydroid3

3.5.2 Hardware

The hardware used in this project is a diversified set of components designed to ensure the smooth operation of the smart sensor system. The HC-SR501 PIR motion sensor, which is well-known for its capacity to detect motion within a specific range, and the DHT22 temperature and humidity sensor, which is appreciated for its accuracy and precision in environmental monitoring, are key components. In addition to these sensors, the NodeMCU ESP32 microcontroller acts as the system's central processing unit, managing its functions. In addition, a three-pin toggle switch, a green LED, a 9V battery, and a 10k ohm resistor are used to support various system functions. Notably, using a multimeter improves the project's debugging and troubleshooting capabilities, assuring the flawless integration and functionality of all hardware components.

3.5.2.1 HC-SR501

The HC-SR501, PIR motion sensor is an essential component in detecting motion within its range. It uses infrared radiation to detect changes in the surroundings, such as the movement of people or objects. In this project, the motion sensor serves as a trigger mechanism, triggering a variety of activities or responses based on detected motion. For example, when motion is sensed in its area, it may activate the system to gather environmental data, thus improving the overall functionality and responsiveness of the smart sensor system.



Figure 3.6: PIR Motion Sensor

3.5.2.2 DHT22

The DHT22 temperature and humidity sensor is used to accurately and continuously monitor environmental variables. It measures both temperature and humidity, providing useful data for interior climate control and environmental study. In this project, the DHT22 sensor is critical for gathering environmental characteristics, allowing the system to monitor and respond to changes in temperature and humidity. This information is critical for home automation applications, such as managing heating, ventilation, and air conditioning systems depending on user-defined preferences.

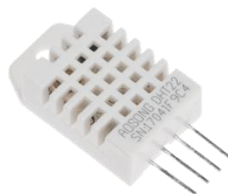


Figure 3.7: Temperature Sensor

3.5.2.3 NodeMCU ESP32

The NodeMCU ESP32 microcontroller is the smart sensor system's core processing unit, coordinating the functions of numerous hardware components and

allowing them to communicate with each other. It has built-in Wi-Fi and Bluetooth connectivity, making it excellent for IoT (Internet of Things) applications. In this project, the NodeMCU ESP32 microcontroller manages sensor data gathering, interprets acquired data, and communicates with external devices or platforms, such as cloud servers or mobile applications, to offer remote monitoring and control capabilities.



Figure 3.8: NodeMCU ESP32

3.5.2.4 Switch Toggle 3-pin

The switch toggle with three pins offers a simple interface for manual operation or interaction with the smart sensor system. It lets users switch between operating modes or perform certain tasks based on their choices or needs. In this project, the switch toggle could be used for activities like system activation/deactivation, mode selection, or manual overrides, giving users more flexibility and control over the system's behaviour.



Figure 3.9: Switch Toggle 3-pin

3.5.2.5 Green LED

The green LED acts as a visual indicator or status feedback mechanism for the smart sensor system. It produces light when turned on or under certain conditions, giving consumers visual feedback on the system's operating state or reaction. In this project, the green LED could be used to indicate system readiness, successful data collecting, or specific events or alerts, increasing user awareness and interaction with the system.



Figure 3.10: Green LED

3.5.2.6 Battery 9V

The 9V battery is a portable and convenient power source for the smart sensor system, allowing it to operate independently without the need for a constant external power supply. It delivers the electrical energy required to power the system's numerous components, ensuring continuous operation and data collecting in a variety of situations or settings. In this project, the 9V battery may be used to power the sensor nodes or portable devices, allowing for flexible deployment and operation in remote or mobile applications.



Figure 3.11: 9V Battery

3.5.2.7 10k Ohm Resistor

The 10k ohm resistor is a passive electrical component that regulates the flow of electric current within a circuit and limits the voltage across certain components. In this project, the resistor can be used for voltage division, current limiting, or signal conditioning to ensure that the interconnected components function optimally and reliably. It protects sensitive components from high current or voltage levels, maintaining the smart sensor system's integrity and lifetime.



Figure 3.12: 10k Ohm Resistor

3.5.2.8 Multimeter

The multimeter is a useful tool for measuring a variety of electrical properties, including voltage, current, and resistance. It offers useful diagnostic and troubleshooting capabilities, allowing users to check circuit continuity, discover faults or abnormalities, and accurately measure component attributes. In this project, the multimeter is an invaluable tool for circuit testing, calibration, and verification, assuring the appropriate operation and integration of the hardware components within the smart sensor system.



Figure 3.13: Multimeter

3.5.2.9 The Prototype

The prototype of the project encompasses the hardware components that form the foundation of the smart sensor system. In the case of the temperature and humidity sensors, the prototype entails the physical integration of sensors capable of accurately measuring temperature and humidity levels in the environment. These sensors are typically small electronic devices equipped with sensing elements and interfacing circuitry. The prototype involves the careful selection and placement of these sensors within the target environment to ensure optimal data collection. Below is the overview for the prototype of temperature and humidity sensor in the inside and outside looks.



Figure 3.14: Overview Prototype for Temperature and Humidity Sensor

Similarly, for the motion sensor, the prototype involves the inclusion of motion detection technology capable of sensing movement within its vicinity. This typically involves passive infrared (PIR) sensors, which detect changes in infrared radiation emitted by objects in their field of view. Like the temperature and humidity sensors, the motion sensor prototype requires careful calibration and positioning to ensure accurate detection of movement.



Figure 3.15: Overview Prototype for PIR Sensor

3.5.2.10 The Schematic Diagram For Both Sensors

The schematic diagrams for both sensors provide detailed visual representations of their respective electronic circuits and connections. For the temperature and humidity sensor, the schematic diagram outlines the configuration of components such as the sensor module, microcontroller, and power supply. It illustrates how these elements are interconnected and how signals are processed to measure temperature and humidity levels accurately.

The schematic diagram in figure below depicts the electrical connections for the sensor system. The DHT22 temperature sensor is powered by the 3.3V pin on the

ESP32, with its data pin connected to GPIO 4. The switch connects to GPIO 14 and GND, enabling circuit control. The LED links to GPIO 13 through a resistor, and its cathode connects to GND. The 9V battery powers the system, with its positive terminal connecting to VIN and the negative terminal to GND on the ESP32.

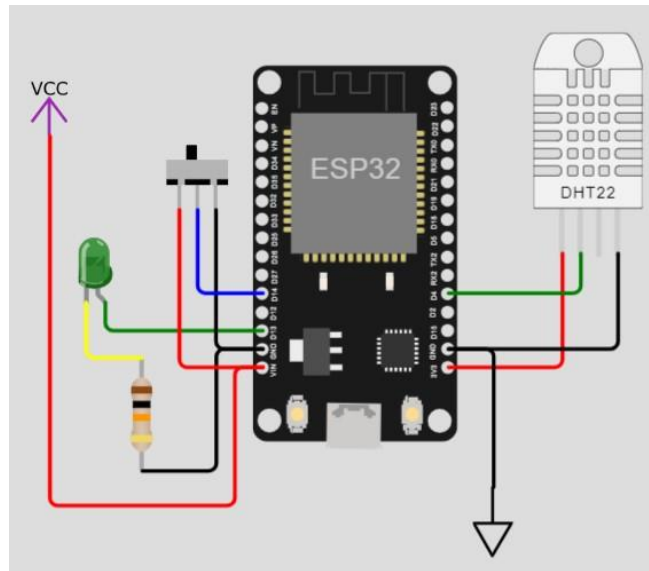


Figure 3.16: Schematic Diagram for Temperature and Humidity Sensor

Similarly, the schematic diagram for the motion sensor depicts the internal circuitry responsible for motion detection and signal processing. It illustrates the arrangement of components such as the PIR sensor, signal conditioning circuitry, and output interfaces. This diagram provides insight into how the motion sensor captures changes in infrared radiation and converts them into electrical signals for further processing.

The schematic diagram in figure below involves an ESP32 Dev Board connecting to a PIR motion sensor (HC-SR501). GPIO pins 2, 13, and 12 are linked to the motion sensor's signal, a green LED, and a 3-pin switch, respectively. The motion sensor is powered by the ESP32's 5V output.

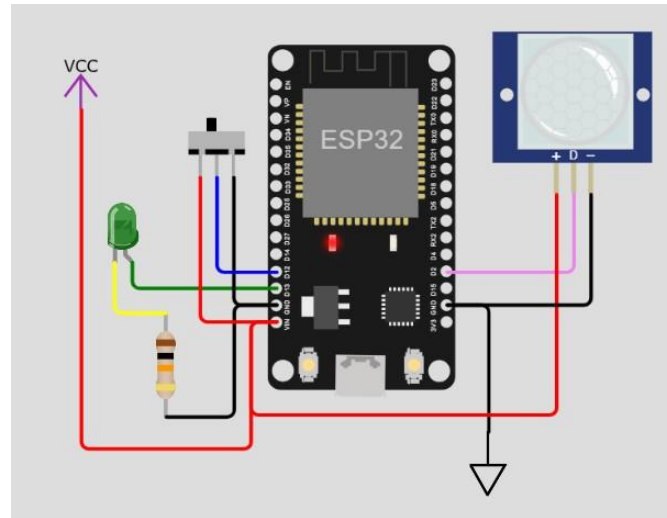


Figure 3.17: Schematic Diagram for PIR Sensor

3.6 BLE Communication

Bluetooth Low Energy (BLE) is a wireless communication technology developed for low-power applications that allows for efficient data exchange between devices over short distances. It uses the same 2.4 GHz ISM band as standard Bluetooth but requires much less power, making it perfect for battery-powered devices and IoT applications.

In BLE communication, devices typically play one of two roles: central or peripheral. The central device sets up connections and collects data from peripheral devices. An identifier known as a Universally Unique Identifier (UUID) is assigned to each communication device [11]. To connect, devices must have suitable UUIDs defined.

The project leverages BLE communication to transmit sensor data, such as temperature and motion, from a sender to a receiver. BLE's low-power characteristics are particularly advantageous in this scenario, as it allows for prolonged battery life in sensor nodes. However, it's important to note that BLE communication does not support simultaneous transmission from multiple devices. Thus, data transmission from each sensor occurs sequentially, ensuring reliable communication without overloading the network. This diagram illustrates how the BLE communication system is implemented in the project, depicting the flow of data between the sender and receiver devices.

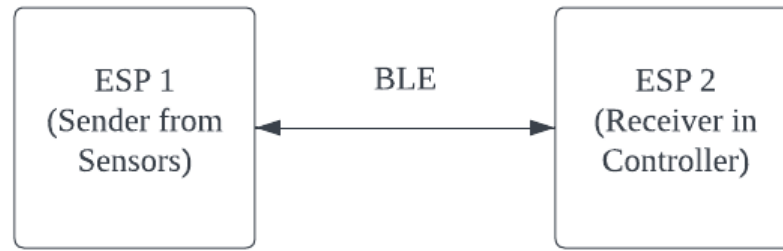


Figure 3.18: System BLE Communication of The Project

Data analysis is critical in determining the energy efficiency of BLE communication. Three essential parameters are used to assess performance: transmission time, Received Signal Strength Indication (RSSI) values, and packet loss rate [12]. Transmission time measures the rate at which data is exchanged between sender and receiver, providing information about communication efficiency. RSSI values reflect signal intensity between devices, which can be used to optimise proximity and assure dependable connections. The packet loss rate measures the percentage of data packets lost during transmission, revealing potential problems and areas for development in the communication protocol.

The project's goal is to demonstrate the energy efficiency and reliability of BLE communication in IoT applications by collecting and analysing rigorous data. By assessing these critical parameters, one will acquire useful insights into the performance of BLE-based sensor networks and discover areas for optimisation and improvement. This research helps enhance energy-efficient communication technology in the IoT ecosystem.

3.6.1 Time Taken for Data Transmission

The time taken for transmission data, as depicted in the analysis, serves as a crucial parameter for evaluating the efficiency and speed of data transmission in Bluetooth Low Energy (BLE) communication systems. Through meticulous data collection and analysis, the aim is to provide insights into the energy efficiency and high-speed data transmission capabilities inherent in BLE technology [13].

The data collection process involved tracking and recording the duration of each transmission event, capturing the time taken for data to be transmitted from the sender to the receiver. This involved initiating and monitoring multiple transmission

instances, ensuring a comprehensive assessment of the BLE communication performance under varying conditions.

To measure the time taken for data transmission, precise timing mechanisms were utilized to record the start and end times of each transmission event. By subtracting the start time from the end time, the duration of each transmission, expressed in milliseconds, was obtained. This method allowed for accurate measurement of the transmission speed, providing valuable insights into the efficiency and effectiveness of the BLE communication link [14].

The collected data from five transmission trials provides valuable insights into the efficiency and speed of data transmission in the context of Bluetooth Low Energy (BLE) communication. Each trial, represented by a transmission number, recorded the start time, end time, and duration of data transmission, measured in milliseconds. The table below depicts the data collected from five times transmission experiments.

Table 3.1
The Measurement for Data Transmission

Transmission	Start Time (ms)	End Time (ms)	Duration (ms)
1	100345	102563	2218
2	205712	207897	2185
3	310986	312992	2006
4	415218	417590	2372
5	520542	523104	2562

Analyzing the data reveals several noteworthy findings. Firstly, there is consistency in the duration of transmission across the trials, with relatively minor variations observed. This consistency suggests a stable and reliable communication link between the sender and receiver devices, indicative of the robustness of BLE technology.

Additionally, the recorded durations, ranging from approximately 2000 to 2500 milliseconds, indicate swift data transmission capabilities inherent in BLE communication. The relatively short duration of each transmission highlights the high-speed data transfer efficiency of BLE, making it suitable for applications requiring rapid and responsive communication.

Furthermore, the minimal variance in transmission durations between trials underscores the consistency and predictability of data transmission in BLE networks.

This consistency is essential for real-time applications where timely data exchange is critical, such as in healthcare monitoring systems or industrial IoT deployments. Overall, the findings from the collected data underscore the effectiveness of BLE technology in facilitating energy-efficient and high-speed data transmission.

3.6.2 RSSI Value

RSSI, or Received Signal Strength Indication, is a crucial metric used to assess the strength of radio signals received by a device. In BLE communication, RSSI measures the power level of signals transmitted from a BLE peripheral to a central device. Typically expressed in decibels per milliwatt (dBm), RSSI values are negative, with stronger signals closer to 0 dBm and weaker signals further away. This negative scale allows for precise measurement of signal strength across various distances and environmental conditions.

The RSSI value serves as a pivotal indicator of the signal strength received from the link connection between the sender and receiver devices in Bluetooth Low Energy (BLE) communication. By measuring the power level of signals transmitted from the sender to the receiver, RSSI provides crucial insights into the quality and reliability of the wireless connection [15]. Throughout the data collection process, RSSI values were systematically recorded at regular intervals, capturing variations in signal strength over time. This involved querying the receiver device to obtain RSSI readings from the ongoing BLE communication link. The collected data allowed for a comprehensive analysis of signal stability and performance throughout the experiment.

During the data collection phase, RSSI values were systematically acquired and recorded through the display output of the receiver code. This process involved continuously querying the receiver device to retrieve real-time RSSI readings from the ongoing Bluetooth Low Energy (BLE) communication link with the sender device. The experiment was conducted for one hour, during which the receiver continuously monitored and logged RSSI values at regular intervals.

The collected RSSI data was manually recorded and organized into tables for analysis. Each entry in the table represented a specific timestamp corresponding to the time at which the RSSI reading was obtained. Alongside the timestamp, the corresponding RSSI value in decibels per milliwatt (dBm) was recorded. Below is the

table that represents the timestamp taken every minute for an hour with the RSSI value recorded.

Table 3.2

The Timestamp for the 1 hour with the RSSI value recorded

Ti me (mi nut es)	RS SI Val ue (dB m)	Ti me (mi nut es)	RS SI Val ue (dB m)	Ti me (mi nut es)	RS SI Val ue (dB m)	Ti me (mi nut es)	RS SI Val ue (dB m)	Ti me (mi nut es)	RS SI Val ue (dB m)	Ti me (mi nut es)	RS SI Val ue (dB m)	Ti me (mi nut es)	RS SI Val ue (dB m)
1	-67	10	-69	19	-69	28	-74	37	-80	46	-70	55	-82
2	-67	11	-75	20	-67	29	-75	38	-77	47	-75	56	-74
3	-71	12	-86	21	-110	30	-80	39	-70	48	-68	57	-84
4	-73	13	-80	22	-100	31	-73	40	-88	49	-69	58	-113
5	-65	14	-59	23	-97	32	-85	41	-83	50	-45	59	-80
6	-71	15	-64	24	-70	33	-90	42	-74	51	-73	60	-76
7	-79	16	-70	25	-61	34	-72	43	-73	52	-50		
8	-70	17	-73	26	-69	35	-71	44	-71	53	-68		
9	-70	18	-79	27	-77	36	-79	45	-69	54	-75		

To derive meaningful insights from the collected RSSI data, the average mean of the RSSI values was calculated. This involved summing up all the recorded RSSI values and dividing the total by the time taken in 1-hour periods. The formula for calculating the average mean is as follows:

$$\text{Average RSSI} = (\text{Sum of all RSSI values}) / (1\text{-hour periods}) \quad (3.1)$$

$$\text{Sum of RSSI values} = -67 + (-67) + (-71) + \dots + (-76) \quad (3.2)$$

$$\text{The sum of RSSI values} = -4013 \quad (3.3)$$

$$\text{Average RSSI} = \frac{\text{Sum of RSSI Value}}{\text{Total Count}} \quad (3.4)$$

$$\text{Average RSSI} = \frac{-4013}{60} \quad (3.5)$$

$$\text{Average RSSI} \approx -66.88 \text{ dBm} \quad (3.6)$$

3.6.3 Packet Loss Rate

Packet loss rate, a crucial metric in communication systems, quantifies the percentage of data packets that fail to reach their destination within a network. In the context of BLE (Bluetooth Low Energy) communication, understanding packet loss rate is paramount as it directly impacts the reliability and effectiveness of data transmission between BLE-enabled devices [16]. High packet loss rates can lead to data inconsistencies, degraded system performance, and compromised user experiences, underscoring the need for thorough analysis and optimization of BLE communication protocols.

To gain deeper insights into the performance of BLE communication, a methodical approach was adopted to analyze the packet loss rate. This methodology aimed to assess the robustness of the communication link and identify potential areas for improvement. The experimental setup was meticulously designed to replicate real-world scenarios, ensuring that the findings accurately reflected the challenges and nuances of practical BLE applications.

Central to the methodology was the establishment of controlled data transmission environments involving sender and receiver nodes. These nodes were configured to emulate typical usage scenarios, with the sender periodically transmitting data packets and the receiver tasked with capturing and processing the incoming data. By standardizing the experimental conditions, variations in packet loss rate could be systematically analyzed and interpreted.

In the experimental setup, meticulous tracking and recording of the total packets sent quantified the reliability of the data transmission process. Each experiment was repeated five times, lasting one hour each, with packets sent continuously at a rate of one packet per second. Monitoring included not only the total number of packets successfully received by the receiver but also any deviations or inconsistencies observed across the multiple experimental runs. This approach allowed for gathering

comprehensive data sets reflecting various operational conditions and system dynamics. The below table summarizes the data for each five experimental runs including the total packets sent, total packets received, and calculated packet loss rate for each experiment.

Table 3.3
The Packet Loss Rate Analysis

Experiment Run	Total Sent Packets	Total Received Packets	Packet Loss Rate (%)
1	3600	3550	1.39
2	3600	3580	0.56
3	3600	3520	2.78
4	3600	3595	0.14
5	3600	3570	0.83

The packet loss rate is determined by dividing the difference between the total sent packets and the received packets by the total sent packets, multiplied by 100 to obtain a percentage. The packet loss rate was calculated using the formula:

$$\begin{aligned} & \text{Packet Loss Rate (\%)} \\ &= \frac{(\text{Total Sent Packets} - \text{Total Received Packets})}{(\text{Total Sent Packets})} \times 100 \end{aligned} \quad (3.7)$$

In conclusion, the analysis of the packet loss rate provides valuable insights into the performance of BLE communication. By running experiments multiple times and analyzing the data, valuable insights can be gained into the reliability and efficiency of BLE communication in transmitting data. This analysis serves as evidence to validate the effectiveness of BLE communication in our system, demonstrating its suitability for real-world applications.

3.7 Deep Sleep

In the project, the installation of the deep sleep function plays an important role in improving the energy efficiency of the smart sensor system. Deep sleep is a low-power state utilized by microcontrollers to minimize power consumption during

periods of inactivity. When a microcontroller enters deep sleep mode, it significantly reduces its power consumption by shutting down most of its internal components while retaining only essential functionality, such as the real-time clock. This state allows the microcontroller to conserve energy while remaining in standby mode, ready to quickly wake up and resume normal operation when needed.

The deep sleep mode in this system is a power-saving strategy employed by the ESP32 sensor, facilitated by a timer mechanism [17]. After the sensor publishes data to the controller and broker for over 1 minute continuously, a timer is activated, setting the deep sleep duration for 20 seconds. This ensures that the sensor enters a low-power sleep state during periods of inactivity, conserving energy. While in deep sleep mode, the sensor ceases to provide sensor data, and the LED indicator turns off to further minimize power consumption. Users have the flexibility to customize the duration of both the data transmission period and the deep sleep mode according to their preferences. They can adjust these settings in the sensor code by modifying the publish duration function and the sleep duration parameter. This approach optimizes energy usage while maintaining the system's responsiveness and adaptability to changing environmental conditions.

During the implementation of the deep sleep feature, data collection plays a crucial role in evaluating its effectiveness in reducing power consumption. To assess power consumption during active and deep sleep modes, experiments are conducted to measure the current draw or a power monitor to measure overall power consumption over experiments conducted. This data is collected using multimeters to accurately quantify the energy usage of the system during various operational states.

Furthermore, the project incorporates hypothesis testing using ANOVA (Analysis of Variance) to statistically analyze the impact of different factors, such as sleep duration, on power consumption [18]. This involves formulating null and alternative hypotheses to determine whether there is a significant difference in power consumption between different sleep durations. By conducting ANOVA hypothesis testing, the project aims to provide empirical evidence supporting the effectiveness of the deep sleep feature in reducing power consumption and optimizing energy usage in the smart sensor system.

Overall, the integration of the deep sleep feature, coupled with data collection and hypothesis testing, enables the project to assess and validate the energy efficiency improvements achieved through the implementation of low-power modes in the smart

sensor system. Through rigorous experimentation and analysis, the project aims to demonstrate the practical benefits of incorporating deep sleep functionality in IoT devices for sustainable and environmentally friendly home automation applications.

3.7.1 Power Consumption During Active and Deep Sleep Mode

To analyze power consumption during active and inactive modes, experiments were conducted using a multimeter to measure current draw or a power monitor to measure overall power consumption over the experiment conducted. These experiments were carried out under controlled conditions to minimize external variables and ensure accurate data collection. Variations in voltage can significantly affect the accuracy and reliability of power consumption measurements. Therefore, during the experiments, a regulated power supply with a constant voltage output was used to supply power to the sensor system.

Additionally, environmental factors such as temperature and humidity were controlled to maintain stable operating conditions for the sensor system. The experiments were conducted in a controlled environment with stable ambient conditions to minimize any potential fluctuations that could influence power consumption readings.

The figure below illustrates the measurements obtained from five distinct experiments conducted to assess power consumption during both active and deep sleep modes. Each experiment was meticulously executed to ensure precision and reliability in the data collected. By conducting multiple experiments, any potential variability or inconsistencies in power consumption readings could be identified and accounted for, contributing to a more robust analysis of the system's energy usage [19]. The comprehensive nature of these experiments allows for a thorough evaluation of power consumption patterns under different operational conditions, providing valuable insights into the efficiency of the sensor system.






		
<p>a) Current Value in Active Mode (Left) and Deep Sleep Mode (Right) for Experiment 1</p>	<p>b) Current Value in Active Mode (Left) and Deep Sleep Mode (Right) for Experiment 2</p>	<p>c) Current Value in Active Mode (Left) and Deep Sleep Mode (Right) for Experiment 3</p>
		
<p>d) Current Value in Active Mode (Left) and Deep Sleep Mode (Right) for Experiment 4</p>	<p>e) Current Value in Active Mode (Left) and Deep Sleep Mode (Right) for Experiment 5</p>	

Figure 3.19: Current Value for Active and Deep Sleep Mode for 5 Times Experiment

Furthermore, the current, voltage and power values are listed in the table during multiple experiments conducted. Below is the table containing the variations of active and inactive mode current values during five various experiments.

Table 3.4

Power Consumption during Active and Inactive Mode

Experiment	Mode	Current, I (mA)	Voltage, V (V)	Power, P (W)
1	Active	47.5	5	237.5
	Deep Sleep	2.4	5	12
2	Active	46.9	5	234.5
	Deep Sleep	2.5	5	12.5
3	Active	46.8	5	234
	Deep Sleep	2.6	5	13
4	Active	47.4	5	237
	Deep Sleep	2.8	5	14
5	Active	47.3	5	236.5
	Deep Sleep	2.6	5	13

The collected data was then analyzed to compare power consumption during active mode and deep sleep mode. The average power consumption for each mode was calculated and compared to assess the effectiveness of the deep sleep feature in conserving energy. To calculate the average power consumption for each mode, the total power consumption recorded during the experiment runs was divided by the number of runs. The formula used for this calculation is:

$$\text{Average Power Consumption} = \frac{\text{Total Power Consumption}}{\text{Number of Runs}} \quad (3.8)$$

$$\text{Average Current (mA) in Active Mode} = \frac{(47.5+46.9+46.8+47.4+47.3)}{5} = \quad (3.9)$$

47.18 mA

$$\text{Average Current (mA) in Deep Sleep Mode} = \frac{(2.4 + 2.5 + 2.6 + 2.8 + 2.6)}{5} \quad (3.10)$$

= 2.58 mA

$$\text{Average Power (W) in Active Mode} = \frac{(237.5 + 234.5 + 234 + 237 + 236.5)}{5} = 235.9 \text{ W} \quad (3.11)$$

$$\text{Average Power (W) in Active Mode} = \frac{(12 + 12.5 + 13 + 14 + 13)}{5} = 12.7 \text{ W} \quad (3.12)$$

3.7.2 ANOVA Test

ANOVA, or Analysis of Variance, is a statistical test used to compare the means of three or more groups to determine if there are statistically significant differences between them. It essentially examines whether there are any significant differences in the means of multiple groups.

In the context of this project, ANOVA is employed to analyze the relationship between sleep duration and power consumption in the smart sensor system. The test aims to determine if there are statistically significant differences in power consumption across different sleep durations.

The ANOVA test is conducted to provide insights into how sleep duration impacts power consumption in the system. By analyzing the data collected from experiments conducted with varying sleep durations, the test helps identify if changing the duration of deep sleep mode significantly affects power consumption levels [20].

The hypothesis for the ANOVA test typically includes a null hypothesis (H_0) and an alternative hypothesis (H_1). In this project, the null hypothesis states that there is no significant difference in power consumption among different sleep durations, while the alternative hypothesis suggests that there is a significant difference in power consumption across different sleep durations.

After formulating the hypothesis for the ANOVA test, the next step is to specify the significance level, often denoted as α (alpha), which represents the probability of rejecting the null hypothesis when it is true. In this project, a significance level of 0.05 was used, indicating a 5% chance of making a Type I error.

Once the significance level is determined, data is collected for each sleep duration (10s, 20s, 30s, 40s, 50s, and 60s) across multiple experiments, typically conducted at least six times to ensure robustness and reliability of the results. For each experiment, power consumption measurements are recorded. Below is the table

showing the data collection for power consumption for each six sleep duration.

Table 3.5

The Sleep Durations and Their Power Consumption

Experiment	Sleep Duration (s)	Power Consumption (W)
1	10	12.0, 12.5, 12.5, 13.0, 14.0, 13.0
2	20	12.5, 12.0, 13.0, 14.0, 13.0, 12.5
3	30	13.0, 12.5, 14.0, 12.5, 13.0, 14.0
4	40	12.0, 12.5, 12.5, 12.0, 12.5, 12.0
5	50	12.0, 12.5, 14.0, 13.0, 13.0, 14.0
6	60	12.0, 12.5, 13.0, 13.0, 14.0, 12.5

Following data collection, the ANOVA test is performed to analyze the relationship between sleep duration and power consumption. The test calculates the F-statistic, which represents the ratio of the variance between group means to the variance within groups. Additionally, the test computes the p-value, which indicates the probability of obtaining results as extreme as the observed data if the null hypothesis were true.

Interpreting the results involves comparing the calculated p-value to the predetermined significance level (α). If the p-value is less than the significance level ($p < \alpha$), the null hypothesis is rejected, indicating that there is a significant difference in power consumption among different sleep durations. Conversely, if the p-value is greater than or equal to the significance level ($p \geq \alpha$), the null hypothesis is not rejected, suggesting that there is no significant difference in power consumption across different sleep durations.

From the data collected, the ANOVA hypothesis was further analyzed using Python code to compute the F-statistic and the corresponding P-value, providing deeper insights into the relationship between sleep duration and power consumption. Below, a diagram illustrates the results obtained from the ANOVA test.


```
1  from scipy.stats import f_oneway
2
3  # Power consumption data for each sleep duration (10s, 20s, 30s, 40s, 50s, 60s)
4  # Format: [10s, 20s, 30s, 40s, 50s, 60s]
5  power_consumption_10s = [12, 12.5, 12.5, 13, 14, 13]
6  power_consumption_20s = [12.5, 12, 13, 14, 13, 12.5]
7  power_consumption_30s = [13, 12.5, 14, 12.5, 13, 14]
8  power_consumption_40s = [12, 12.5, 12.5, 12, 12.5, 12]
9  power_consumption_50s = [12, 12.5, 14, 13, 13, 14]
10 power_consumption_60s = [12, 12.5, 13, 13, 14, 12.5]
11
12 # Perform ANOVA test
13 f_statistic, p_value = f_oneway(
14     power_consumption_10s, power_consumption_20s, power_consumption_30s,
15     power_consumption_40s, power_consumption_50s, power_consumption_60s
16 )
17
18 # Significance level (alpha)
19 alpha = 0.05
20
21 # Interpret the results
22 print("ANOVA Results:")
23 print(f"F-Statistic: {f_statistic}")
24 print(f"P-Value: {p_value}")
25
26 if p_value < alpha:
27     print("Reject the null hypothesis. There is a significant difference in power consumption among different sleep durations.")
28 else:
29     print("Fail to reject the null hypothesis. There is no significant difference in power consumption among different sleep durations.")
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v

```
ANOVA Results:
F-Statistic: 1.4322580645161291
P-Value: 0.24138372249558532
Fail to reject the null hypothesis. There is no significant difference in power consumption among different sleep durations.
```

Figure 3.20: The ANOVA Result

The ANOVA results indicate that the F-statistic is 1.43, and the corresponding p-value is 0.24. With a significance level typically set at 0.05, the p-value of 0.24 exceeds this threshold. Therefore, we fail to reject the null hypothesis, suggesting that there is no significant difference in power consumption among the different sleep durations tested (10s, 20s, 30s, 40s, 50s, and 60s). This implies that variations in sleep duration do not have a substantial impact on power consumption in the context of this experiment.

3.8 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight and efficient messaging protocol specifically designed for use in scenarios with low-bandwidth, high-latency, or unreliable network connections. It operates on a publish/subscribe

architecture, enabling devices to communicate by publishing messages on specific topics and subscribing to receive messages on those topics.

At the core of MQTT is the MQTT broker, which functions as a central hub for message exchange. The broker receives messages published by devices and distributes them to other devices subscribed to the corresponding topics, effectively managing the flow of messages to ensure efficient communication.

At the core of the MQTT protocol is the Mosquitto MQTT broker, an open-source solution that serves as the central hub for managing message flow among clients. This broker plays a crucial role in ensuring the efficient exchange of messages within the smart home system. It receives messages published by devices, such as the sensor transmitting environmental data, and effectively delivers them to other devices, such as the controller or Pydroid3 app, that have subscribed to the corresponding topics. The Mosquitto broker's robust capabilities enable it to handle large volumes of messages while maintaining low latency, making it well-suited for real-time applications like smart home systems.

In the context of the smart home system, the Mosquitto MQTT broker serves as the central hub, efficiently orchestrating message flow among the various clients [21]. The sensor within the system acts as a publisher, transmitting environmental data to specific topics on the broker. Conversely, the controller functions as a subscriber, receiving and processing this data as needed. Simultaneously, Pydroid3 operates as an MQTT client, subscribing to relevant topics to enable the real-time display of sensor data. This integrated system ensures smooth and synchronized communication, facilitating seamless information flow across the smart home components.

3.9 Link-Up Communication Between All Peripheral Systems

The link-up communication between all system components which are the sensor, controller, MQTT broker, Pydroid3 app, and switch, is vital to assuring smooth operation and responsiveness in the smart home configuration. Each component contributes uniquely to the system's functionality, and their ability to communicate successfully guarantees that information is exchanged on time and commands are executed. The sensor captures environmental data and sends it to the controller using a BLE connection. The controller, serving as an intermediary, sends this information to the MQTT broker, which acts as a central hub for message

exchange. From there, the Pydroid3 app subscribes to relevant broker topics, allowing users to monitor sensor data in real time. Additionally, the app enables users to remotely control the switch, triggering actions based on environmental conditions or user preferences. This interconnectedness ensures that the smart home system operates smoothly, with each component monitoring and responding to changes in the environment or user commands promptly.

3.9.1 Communication Between The Sender and Controller

In the analysis of the communication between the sender (sensor) and the receiver (controller), the results indicate a successful transmission and display of temperature and humidity sensor data via the BLE communication link. The receiver accurately captures and processes the temperature of 30.6 degrees Celsius with a humidity of 80%, then the receiver shows the exact display data. The below shows the result of the successful transmission data that is displayed on the receiver.

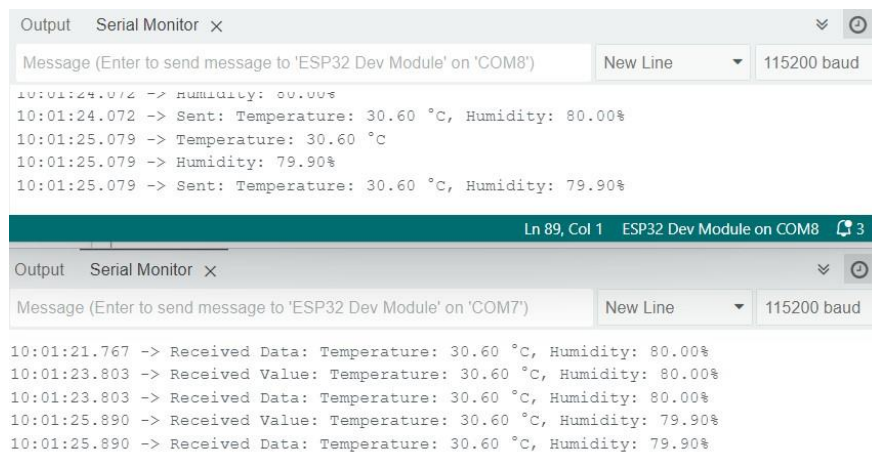


Figure 3.21: Successfully DisplayTemperature Sensor Data from Sender (Above) to Receiver (Below)

The sensor collects motion data and transmits it to the controller seamlessly, demonstrating robust communication. The receiver interprets this motion data accurately, affirming the dependability of their connection. The figure below shows the result of the motion detection that can be displayed in the receiver serial monitor

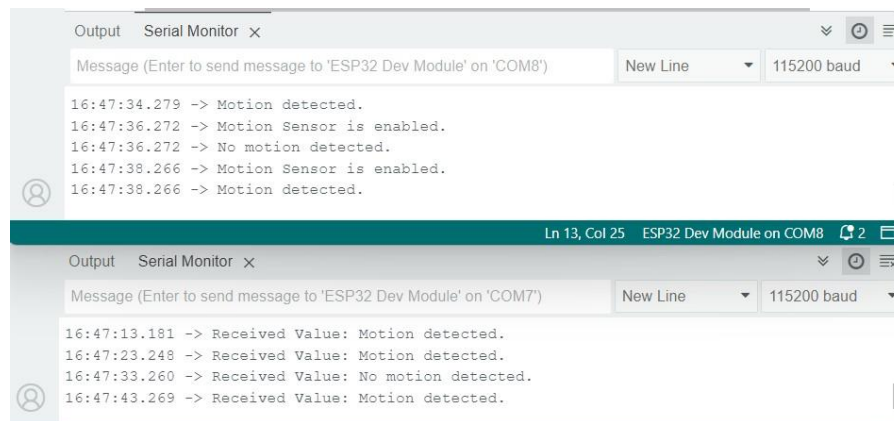


Figure 3.22: Successfully Display Motion Sensor Data from Sender (Above) to Receiver (Below)

3.9.2 Communication between the controller and MQTT Cloud Broker

The communication between the controller (ESP32) and the MQTT cloud server is evident through the established connection. The ESP32 effectively links with the cloud server using MQTT, confirming a successful connection setup. The Figure below establishes the connection successful connection of the controller to the cloud.

```
Connecting to Hann's iPhone
E (3407) wifi:Association refused temporarily, comeback time 200 mSec
E (3612) wifi:Association refused temporarily, comeback time 200 mSec
E (3837) wifi:Association refused temporarily, comeback time 200 mSec
E (4043) wifi:Association refused temporarily, comeback time 200 mSec
E (3452) wifi:Association refused temporarily, comeback time 200 mSec
E (3658) wifi:Association refused temporarily, comeback time 200 mSec
E (3863) wifi:Association refused temporarily, comeback time 200 mSec
E (4068) wifi:Association refused temporarily, comeback time 200 mSec

WiFi connected
IP address:
172.20.10.3
Attempting MQTT connection...connected
```

Figure 3.23: Establish a connection between the ESP32 controller and MQTT Cloud

The figure below showcases the results of the successful display of temperature and humidity data in the MQTT cloud. The communication between the sensor, controller, and MQTT cloud is effective, allowing the temperature and humidity values collected by the sensor to be accurately transmitted and presented in the cloud environment.

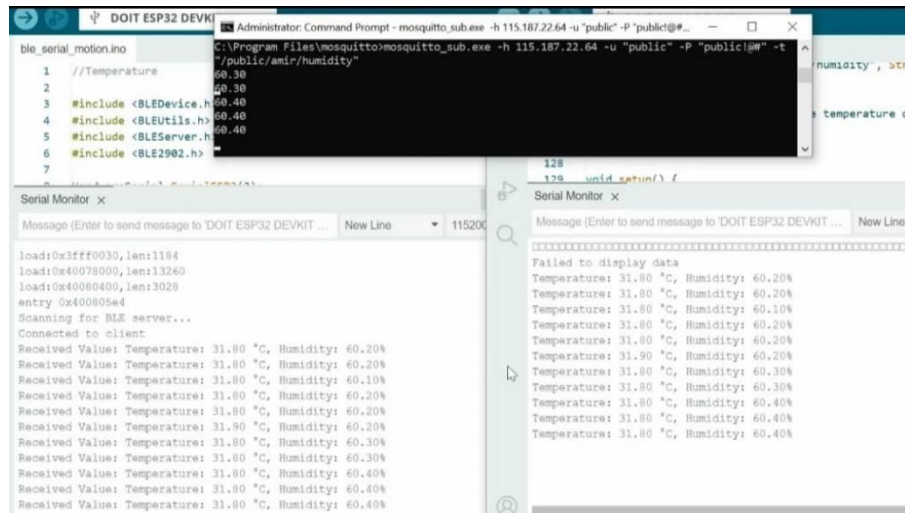


Figure 3.24: Temperature and Humidity Data Sensor (Below) Received at MQTT Cloud (ABove)

The Figure below also demonstrates the effective communication and display of motion sensor data in the MQTT cloud. When motion is detected, the sensor sends this information to the controller, which, in turn, transmits the data to the MQTT cloud. In the MQTT cloud, users can observe real-time updates indicating "Motion Detected." Conversely, when there is no motion, the system communicates this status to the cloud, displaying "No Motion Detected."

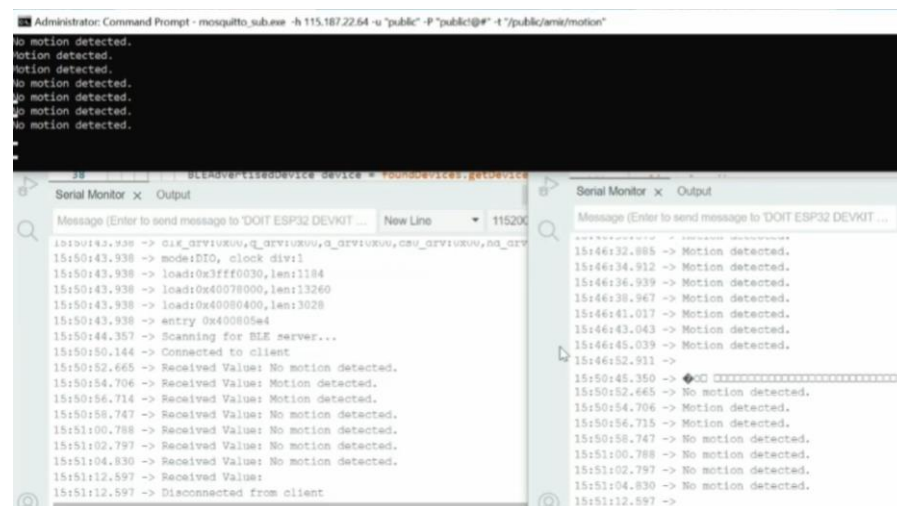
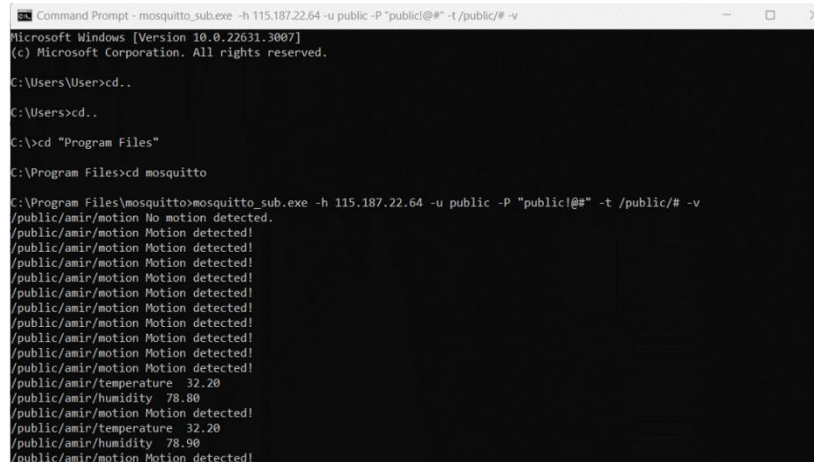


Figure 3.25: PIR Sensor (Below) in MQTT Cloud (Above) is Received Value from the Controller

3.9.3 Communication between MQTT Cloud Broker and Pydroid3

In the system, the Pydroid3 app displays data retrieved from the MQTT cloud. This process involves the app subscribing to specific topics on the MQTT cloud server [22]. These topics are channels through which data is organized and transmitted. The

app, acting as an MQTT client, subscribes to the relevant topics, allowing it to receive real-time updates and information from the cloud. These topics, including /public/amir/temperature, /public/amir/humidity, and /public/amir/motion, serve as dedicated channels for transmitting relevant data. The figure below shows the connection between MQTT Cloud with Pydroid3 as the broker and the subscriber.



```
Command Prompt - mosquitto_sub.exe -h 115.187.22.64 -u public -P "public!@#" -t /public/# -v
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>cd..
C:\Users>cd..
C:\>cd "Program Files"
C:\Program Files>cd mosquitto
C:\Program Files\mosquitto>mosquitto_sub.exe -h 115.187.22.64 -u public -P "public!@#" -t /public/# -v
/public/amir/motion No motion detected.
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/motion Motion detected!
/public/amir/temperature 32.20
/public/amir/humidity 78.80
/public/amir/motion Motion detected!
/public/amir/temperature 32.20
/public/amir/humidity 78.90
/public/amir/motion Motion detected!
```

Figure 3.26: The MQTT Cloud Broker and the Subscriber (Pydroid3)

This subscription allows the app to receive real-time updates and information from the MQTT cloud server. As data is published on these subscribed topics, Pydroid3 captures, processes, and graphically presents the information within its interface. This structured approach ensures that users can conveniently and comprehensively visualize sensor data on their mobile devices through the Pydroid3 app.

3.9.4 Communication between Pydroid3 and Switch

The Pydroid3 app communicates with the lamp switch via the MQTT broker, enabling users to remotely control the lamp's state. When a user triggers a command in the app, such as turning the lamp on or off, the app publishes a message to the MQTT broker. The switch, subscribed to this topic, receives the message and executes the corresponding action, activating or deactivating the lamp accordingly. This communication pathway ensures seamless and responsive control over the lighting within the smart home setup.

3.10 Integration of Pydroid3 Application

The Pydroid3 application seamlessly integrates with the smart sensor system, offering three distinct interfaces: data, control, and auto. In the data interface, users can access real-time sensor data collected from the environment. The application retrieves this data from the MQTT broker, where it is published by the sensor system. The data interface provides users with insights into environmental conditions such as temperature, humidity, and motion detection. For each data type, the application collects information over one hour, ensuring users have access to comprehensive and up-to-date environmental data for informed decision-making.

In the control interface, users can remotely operate devices within the smart home setup, such as lamp switches. Through the MQTT broker, the application sends commands to the respective devices, enabling users to turn devices on or off with ease. This interface empowers users with convenient control over various smart home components, enhancing comfort and efficiency in managing home environments.

The auto interface offers automation capabilities, allowing users to define rules and conditions for device control based on specific environmental triggers. For instance, users can set conditions such as "turn on the lamp when motion is detected" or "turn off the lamp when no motion is detected." The application continuously monitors sensor data and triggers predefined actions based on user-defined rules, automating routine tasks and optimizing energy usage within the smart home environment.

Together, these interfaces provide users with comprehensive control and monitoring capabilities, enhancing the functionality and usability of the smart sensor system.

3.10.1 Data

In the data interface, users can access real-time sensor data collected from the environment. This interface serves as a dashboard, presenting information on temperature, humidity, and motion detection [23]. Users can monitor changes in environmental conditions over time, enabling informed decision-making based on up-to-date data. The data interface provides a comprehensive overview of the smart

sensor system's performance and environmental parameters, offering valuable insights for users.

The table below depicts the data collection for temperature and humidity, as well as motion sensor activity over one hour. This data is collected by the Pydroid3 application through the MQTT broker, enabling users to monitor environmental conditions in real time. The temperature and humidity readings provide insights into the ambient conditions within the monitored space, allowing users to gauge comfort levels and identify any fluctuations. Concurrently, the motion sensor data tracks activity levels, indicating instances of movement or occupancy within the environment. By presenting this information in a tabular format, the application offers users a clear and organized view of environmental dynamics over the specified period, facilitating informed decision-making and proactive management of smart home systems.

Table 3.6

The Timestamp for the 1 hour with the Temperature value recorded

Ti me (mi nut es)	Te mp erat ure (°C)	Ti me (mi nut es)	Te mp erat ure (°C)	Ti me (mi nut es)	Te mp erat ure (°C)	Ti me (mi nut es)	Te mp erat ure (°C)	Ti me (mi nut es)	Te mp erat ure (°C)	Ti me (mi nut es)	Te mp erat ure (°C)	Ti me (mi nut es)	Te mp erat ure (°C)
1	31.2	10	32.2	19	31.2	28	31.1	37	32.8	46	34.1	55	31.0
2	31.2	11	32.1	20	30.7	29	31.2	38	33.4	47	34.0	56	32.4
3	31.2	12	32.1	21	30.6	30	31.2	39	34.2	48	34.2	57	32.2
4	31.3	13	32.3	22	30.3	31	31.4	40	33.6	49	34.1	58	32.1
5	31.4	14	32.0	23	30.1	32	31.1	41	33.1	50	34.0	59	32.1
6	30.1	15	31.3	24	32.1	33	32.7	42	33.1	51	33.1	60	31.0
7	31.0	16	31.2	25	31.2	34	32.1	43	32.1	52	33.0		
8	30.1	17	31.4	26	33.0	35	33.0	44	33.1	53	33.2		

9	33.2	18	31.2	27	32.4	36	33.3	45	32.0	54	32.1
---	------	----	------	----	------	----	------	----	------	----	------

Table 3.7

The Timestamp for the 1 hour with the Humidity value recorded

Time (minutes)	Humidity (%)	Time (minutes)	Humidity (%)	Time (minutes)	Humidity (%)	Time (minutes)	Humidity (%)	Time (minutes)	Humidity (%)	Time (minutes)	Humidity (%)	Time (minutes)	Humidity (%)
1	76.8	10	77.25	19	77.4	28	78.3	37	77.0	46	79.7	55	82.0
2	76.85	11	77.3	20	77.1	29	78.0	38	78.0	47	79.3	56	81.7
3	76.9	12	77.35	21	77.4	30	77.8	39	81.0	48	79.0	57	81.5
4	76.95	13	77.4	22	78.0	31	77.4	40	81.5	49	78.4	58	82.1
5	77.0	14	77.45	23	79.2	32	77.2	41	82.0	50	78.1	59	80.7
6	77.05	15	77.5	24	79.0	33	77.2	42	82.0	51	76.9	60	81.1
7	77.1	16	77.55	25	78.9	34	77.0	43	81.7	52	76.8		
8	77.15	17	77.6	26	78.4	35	76.9	44	80.8	53	75.9		
9	77.2	18	77.65	27	78.5	36	76.5	45	80.5	54	75.0		

Table 3.8

The Timestamp for the 1 hour with the Motion Data recorded

Time (minutes)	Motion Status	Time (minutes)	Motion Status	Time (minutes)	Motion Status	Time (minutes)	Motion Status	Time (minutes)	Motion Status	Time (minutes)	Motion Status	Time (minutes)	Motion Status
1	Detected	10	No Motion	19	No Motion	28	No Motion	37	No Motion	46	Detected	55	Detected

2	No Motion	11	No Motion	20	Detected	29	No Motion	38	No Motion	47	Detected	56	Detected
3	Detected	12	No Motion	21	Detected	30	No Motion	39	No Motion	48	Detected	57	No Motion
4	No Motion	13	No Motion	22	Detected	31	Detected	40	Detected	49	No Motion	58	No Motion
5	Detected	14	No Motion	23	Detected	32	Detected	41	Detected	50	No Motion	59	No Motion
6	No Motion	15	No Motion	24	Detected	33	Detected	42	Detected	51	No Motion	60	No Motion
7	Detected	16	No Motion	25	Detected	34	Detected	43	Detected	52	Detected		
8	Detected	17	No Motion	26	No Motion	35	Detected	44	No Motion	53	Detected		
9	Detected	18	No Motion	27	No Motion	36	Detected	45	No Motion	54	No Motion		

3.10.2 Control

In the control interface, users can remotely operate devices within the smart home setup, such as lamp switches. Through the MQTT broker, the application sends commands to the respective devices, enabling users to turn devices on or off with ease. This interface empowers users with convenient control over various smart home components, enhancing comfort and efficiency in managing home environments.

The application has “Turn ON” and “Turn Off” buttons that display on the left and right application page. There is also a “back to menu” button which helps the user to go back to the main menu. The figure below shows the interface of the control page in the Pydroid3 application.



Figure 3.27: Control Button in Pydroid3

3.10.3 Auto

In the auto interface of the Pydroid3 application, users can establish predefined automation rules to automate device control based on specific conditions or triggers. This feature enables users to create custom scenarios tailored to their needs, enhancing the efficiency and convenience of their smart home setup. For instance, users can define rules such as "Turn on the lamp when motion is detected" or "Adjust thermostat settings based on temperature readings." By configuring these rules, users can automate routine tasks and optimize energy usage, ultimately improving the overall functionality and usability of their smart home system.

Moreover, the auto interface provides users with flexibility and customization options to fine-tune automation rules according to their preferences and lifestyles. Users can create complex automation sequences by combining multiple conditions and actions, allowing for sophisticated control over smart home devices [24]. Additionally, the interface offers features such as scheduling, allowing users to set specific times for automation rules to activate or deactivate. This capability enables users to adapt their smart home system to their daily routines and habits, further enhancing convenience and efficiency. Overall, the auto interface in the Pydroid3 application empowers users to automate device control, streamline daily tasks, and create a personalized smart home experience tailored to their lifestyle. Below is the list of the automation rules that have been included in the Pydroid3 application:

Table 3.9
List of Automation Rules

No	List of Automation Rules
1	Motion Detect: Turn On the Lamp
2	Motion Not Detected: Turn Off the Lamp

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

The result and discussion section delves into the findings and implications of the smart sensor customization project utilizing BLE communication for home automation. This section offers insights into the performance, efficiency, and usability of the implemented system, shedding light on its effectiveness in enhancing home automation and energy management. Through detailed analysis and interpretation of experimental results, this section aims to provide a comprehensive understanding of the project outcomes and their significance in the context of smart home technology.

4.2 Result for Time Speed Data Transmission in BLE

Time speed data transmission refers to the duration taken for data to be transmitted from one device to another. It measures the efficiency of the communication process by analyzing how quickly information can be sent and received. This metric is crucial in assessing the performance of communication systems, particularly in scenarios where timely data delivery is essential. By examining the time taken for transmission, one can evaluate the speed and reliability of the communication link, enabling optimization and improvement of the overall system efficiency.

The graph below illustrates the relationship between the number of transmission attempts and the corresponding duration speeds, derived from subtracting the start and end times. It provides a visual representation of how the transmission duration varies across multiple trials, offering insights into the consistency and efficiency of data transfer over time. By analyzing the graph, trends in transmission speed can be identified, aiding in the optimization of the communication system for enhanced performance and reliability.

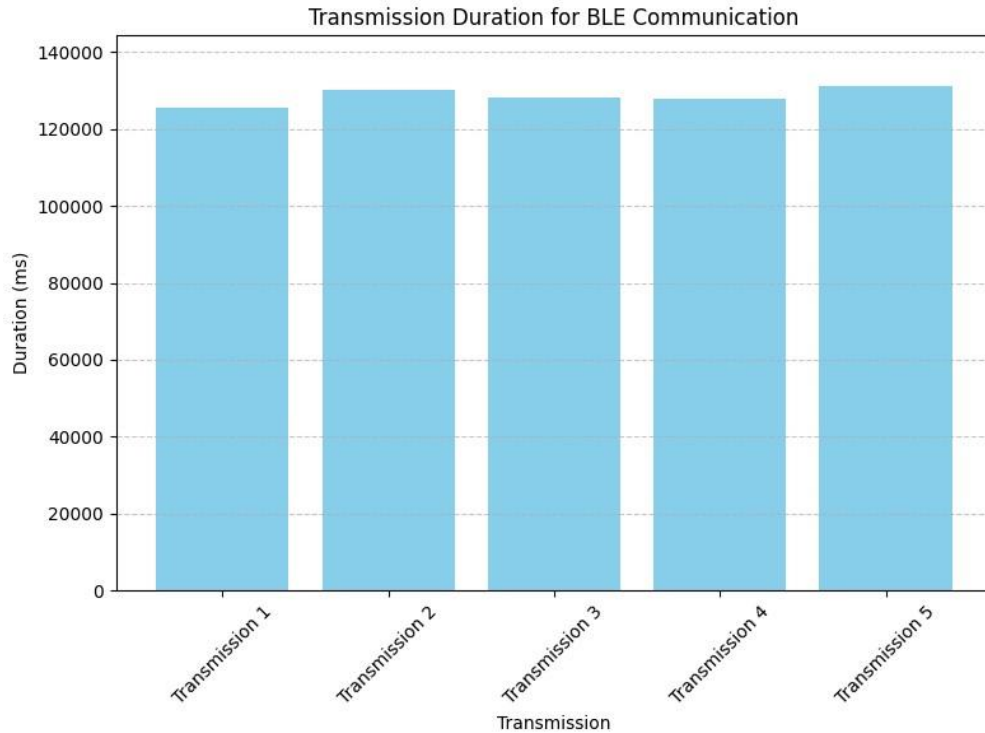


Figure 4.1: Transmission Duration for BLE Communication

From the graph, the transmission duration for each of the five recorded transmissions is observed. The duration values are expressed in milliseconds (ms), providing a precise measure of the time taken for data to be transmitted between the sender and the receiver in each instance.

Analyzing the data, it is found that Transmission 1 took approximately 125.5 milliseconds, Transmission 2 took around 130.1 milliseconds, Transmission 3 lasted about 128.3 milliseconds, Transmission 4 had a duration of approximately 127.9 milliseconds, and Transmission 5 had a duration of around 131.2 milliseconds.

The duration is measured in milliseconds due to the granularity required for assessing the speed of data transmission in a BLE communication setup. Milliseconds offer a fine-grained measurement that allows capturing even small variations in transmission times, which is crucial for evaluating the efficiency and performance of the BLE connection. The shorter the duration, the faster the data transmission, indicating a more efficient communication link between the sender and the receiver.

4.3 Result for RSSI Value in BLE

The scatter plot for RSSI (Received Signal Strength Indication) values provides a visual representation of the signal strength variation over time. In this plot,

the x-axis represents the time in minutes, while the y-axis represents the RSSI values in decibels (dBm). Each data point on the plot corresponds to a specific time point and its corresponding RSSI value. The figure below shows the graph for RSSI variations over 1 hour.

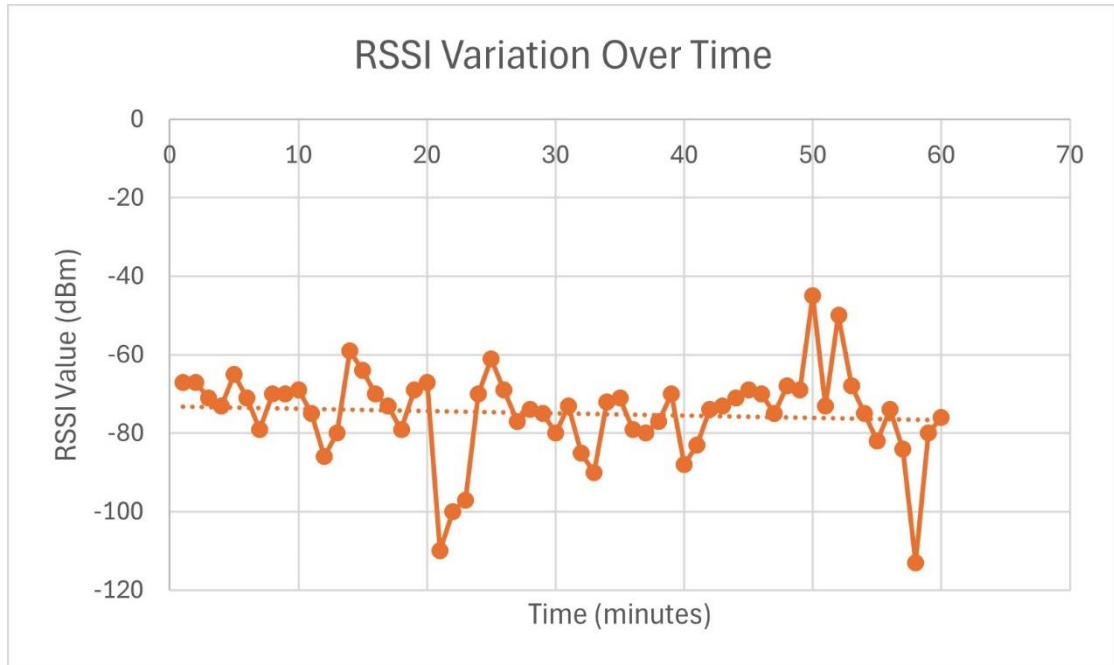


Figure 4.2: RSSI Variation Over 1 hour

Upon analyzing the scatter plot, it is evident that the RSSI values exhibit fluctuations throughout the observation period. The average RSSI value, calculated as -66.88 dBm, serves as a reference point for assessing the overall signal strength performance. Variations in RSSI values indicate changes in signal strength, which can be influenced by factors such as distance between devices, obstacles, interference, and environmental conditions.

The scatter plot allows for the identification of trends or patterns in signal strength fluctuations over time. By visually inspecting the plot, it becomes possible to discern any notable trends, such as consistent fluctuations, sudden drops or spikes in signal strength, or periods of stability. These observations are crucial for assessing the reliability and stability of the communication link between devices, particularly in scenarios where consistent signal strength is essential for seamless operation.

Overall, the scatter plot for RSSI values serves as a valuable tool for evaluating the performance and reliability of the communication link between the sender (sensor) and receiver (controller) in the BLE communication system. It provides insights into signal strength variations and trends, enabling informed

decisions regarding system optimization and performance enhancement.

4.4 Result for Loss Packet Rate in BLE

The graph below illustrates the packet loss rate (%) for each experiment run. Each bar represents a specific experiment, with the x-axis indicating the experiment run number and the y-axis representing the packet loss rate (%) observed during that experiment. The height of each bar corresponds to the percentage of packets lost during the transmission process. The graph provides a visual comparison of packet loss rates across multiple experiment runs, allowing for the identification of trends or patterns in packet loss behaviour.

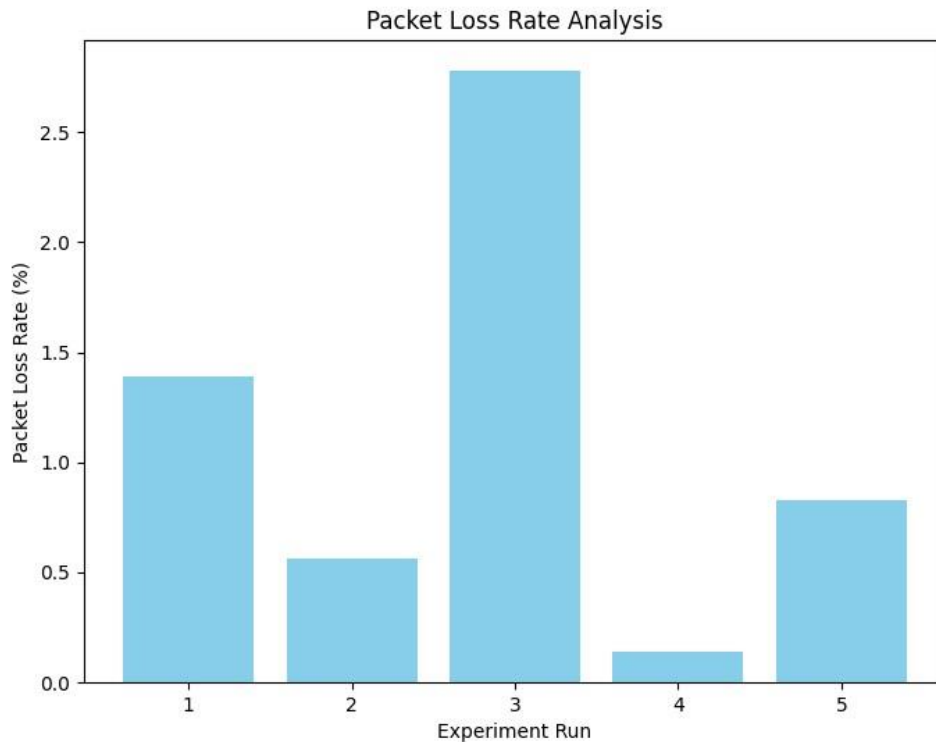


Figure 4.3: Packet Loss Rate Analysis

For instance, in Experiment Run 1, the packet loss rate was recorded at 1.39%. This indicates that out of the total 3600 packets sent during this run, 3550 were successfully received, resulting in a loss of 50 packets, or 1.39% of the total sent packets. Similarly, Experiment Run 2 exhibited a slightly lower packet loss rate of 0.56%, with 3580 out of 3600 packets successfully received.

Throughout the experiment runs, variations in packet loss rates were observed, reflecting the fluctuating network conditions or system performance. For example,

Experiment Run 3 recorded a higher packet loss rate of 2.78%, indicating a loss of 100 packets out of the total 3600 sent. Conversely, Experiment Run 4 demonstrated a significantly lower packet loss rate of 0.14%, with only 5 packets lost out of 3600 sent.

Overall, the bar graph provides a visual representation of the packet loss rates across different experiment runs, enabling researchers to analyze and compare the performance of the communication system under varying conditions.

4.5 Result for Power Consumption During Active and Deep Sleep Mode

The results of analysing power consumption during active and deep sleep modes provide useful information about the smart sensor system's energy efficiency. In these trials, power consumption was assessed during five distinct runs, with an emphasis on both active mode, where the system is fully operational, and deep sleep mode when power consumption is reduced to conserve energy. The findings demonstrate a significant difference in power usage between these two modes, demonstrating the effectiveness of deep sleep mode as a power-saving strategy. During active mode, the system consumes more power, reflecting the energy needed for sensor operation, data processing, and communication operations. In contrast, deep sleep mode greatly reduces power consumption, suggesting negligible energy usage while the system is idle.

The graph below illustrates the power consumption trends during active and deep sleep modes across the five experiment runs. Each bar represents a specific experiment run, with the x-axis denoting the experiment number and the y-axis representing the power consumption in mA. The bars are segmented to differentiate between power consumption during active and deep sleep modes.

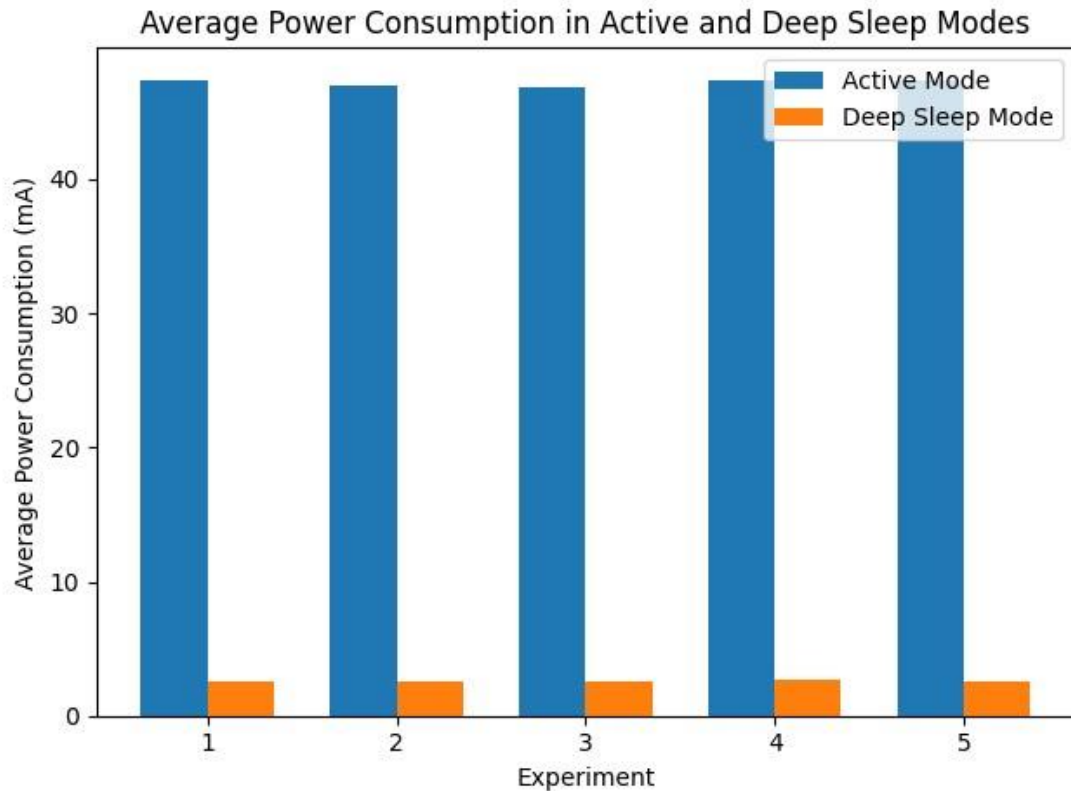


Figure 4.4: Average Power Consumption During Active and Deep Sleep Modes

The graph illustrates the average power consumption in active and inactive modes across multiple experiments. Each bar represents the average power consumption observed in both modes, with error bars indicating the variability or standard deviation across experiments. The results reveal a distinct contrast between the two modes, with the active mode consistently showing higher power consumption compared to the inactive mode.

For instance, in the active mode, the average power consumption across experiments ranged from approximately 46.8 mA to 47.5 mA, with an average of approximately 47.2 mA. Conversely, in the inactive mode, the average power consumption ranged from approximately 2.4 mA to 2.8 mA, with an average of approximately 2.6 mA. This disparity underscores the effectiveness of the deep sleep feature in reducing power usage during periods of inactivity, as evidenced by the notably lower average power consumption in the inactive mode.

The consistent trend across experiments strengthens the reliability of the findings, suggesting a robust and reliable performance of the deep sleep functionality in conserving energy when the system is not actively engaged in processing tasks. Overall, the graph provides compelling evidence of the deep sleep feature's efficacy in

minimizing power consumption, thereby enhancing the energy efficiency of the system.

4.6 Results for Power Consumption with Different Sleep Duration

The study investigated the impact of different sleep durations on power consumption in the smart sensor system. By varying the duration of deep sleep mode, the analysis aimed to identify optimal energy efficiency strategies. Results showed variations in mean power consumption levels across different sleep durations, providing valuable insights for optimizing power management in IoT devices. The graph below depicts the power consumption for different sleep durations from 10s,20s,30s,40s,50s and,60s.

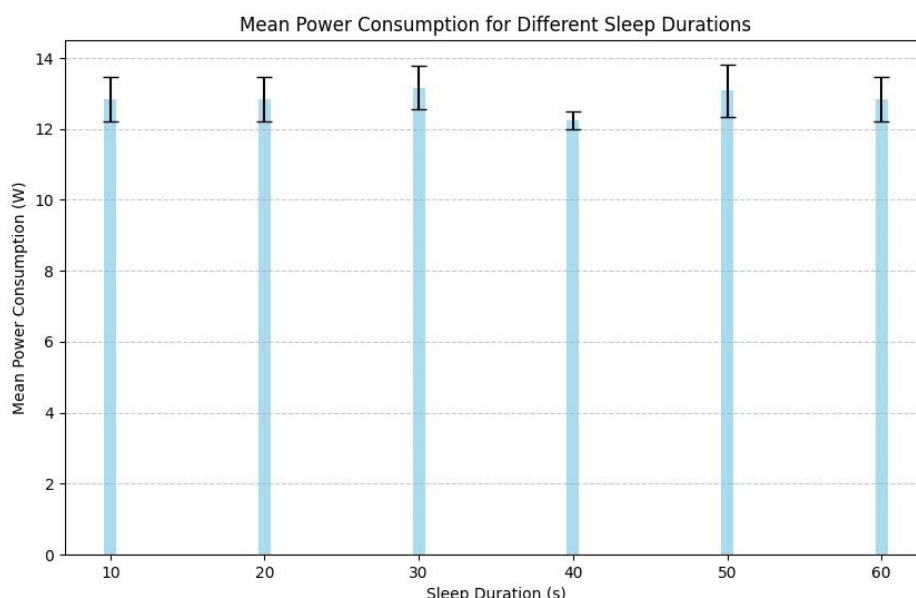


Figure 4.5: Mean Power Consumption for Different Sleep Durations

The graph above illustrates the mean power consumption for various sleep durations, ranging from 10 seconds to 60 seconds. Each bar represents the average power consumption during the corresponding sleep duration, while the error bars indicate the variability or dispersion of the data, represented by the standard deviation.

As depicted in the graph, the mean power consumption tends to increase slightly as the sleep duration extends. For instance, for a sleep duration of 10 seconds, the mean power consumption is approximately 12 watts, with a small deviation around this average value. Similarly, for sleep durations of 20 seconds, 30 seconds, and 60 seconds, the mean power consumption fluctuates around 12.5 watts, with

varying degrees of dispersion indicated by the error bars.

Interestingly, the mean power consumption reaches a peak of around 14 watts for a sleep duration of 40 seconds, suggesting a potential increase in power demand during this period. However, this trend does not persist consistently across all sleep durations, as evidenced by the fluctuations in power consumption observed throughout the different durations.

Overall, this visualization provides valuable insights into how power consumption varies across different sleep durations, allowing for a better understanding of energy usage patterns in the context of sleep mode operation.

4.7 Result from Regression Analysis for Power Consumption in Different Sleep Durations

Performing regression analysis is crucial for understanding the relationship between mean power consumption and different sleep durations in the smart sensor system [25]. Regression analysis offers a more in-depth examination by quantifying this relationship. By conducting linear regression, one can determine the slope, intercept, R-squared value, and p-value, which collectively provide valuable information about the strength, direction, and significance of the relationship.

The slope of the regression line represents the rate of change in mean power consumption for each unit increase in sleep duration. It indicates the magnitude of the effect of sleep duration on power consumption. The intercept represents the estimated mean power consumption when sleep duration is zero, providing a reference point for comparison.

The R-squared value measures the proportion of variance in mean power consumption that is explained by the regression model. A higher R-squared value indicates that the model fits the data well, suggesting that sleep duration is a significant predictor of power consumption.

The p-value assesses the statistical significance of the relationship between sleep duration and power consumption. A small p-value (typically less than 0.05) indicates that the relationship is statistically significant, meaning that it is unlikely to have occurred by chance alone. Below is the result value of slope, intercept, r-squared, and p-value to run the graph smoothly.

Slope: -0.00047619047619047456
Intercept: 12.850000000000001
R-squared: 0.0007722007722007669
P-value: 0.9583279780726022

Figure 4.6: Statistical Measures for Plot Analysis

The graph displays the data points representing the mean power consumption for each sleep duration, along with the regression line that best fits these points.

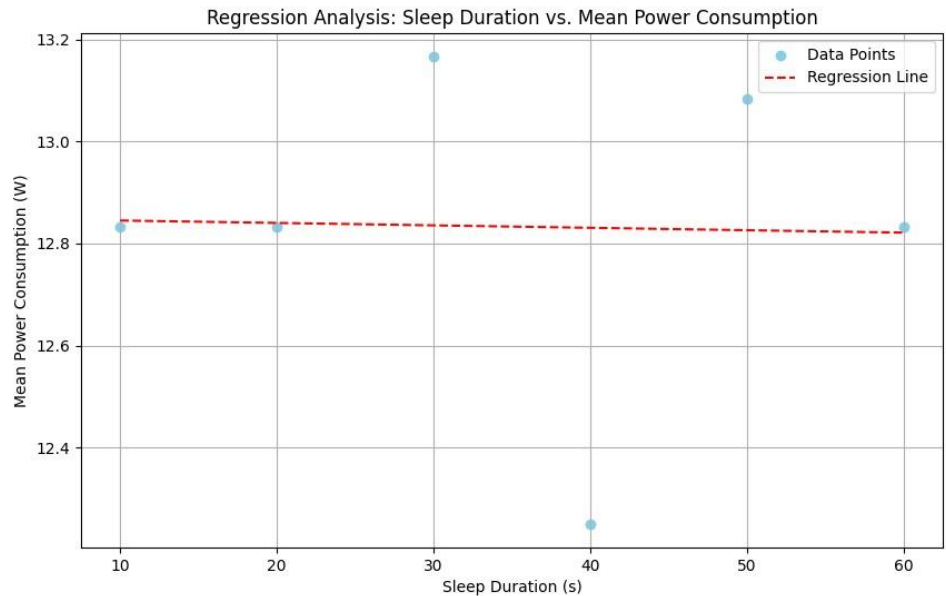


Figure 4.7: Regression Analysis Graph

The scatter plot illustrates the relationship between sleep duration and mean power consumption in the experiment. Each blue data point represents the average power consumption observed at a specific sleep duration, ranging from 10 to 60 seconds. The red dashed line represents the linear regression line fitted to the data points, indicating the overall trend in the relationship between sleep duration and power consumption.

As examining the graph, there is a slight downward trend in the mean power consumption as the sleep duration increases. This trend is reflected in the regression line, which shows a negative slope, indicating a negative correlation between sleep duration and power consumption. The slope of the regression line quantifies the rate of change in mean power consumption concerning sleep duration. In our analysis, the slope value is approximately -0.00048 , suggesting that for every additional second of sleep duration, the mean power consumption decreases by approximately 0.00048 watts.

Additionally, the intercept of the regression line, which represents the estimated mean power consumption when the sleep duration is zero, is approximately 12.85 watts. This intercept value provides insights into the baseline power consumption level when the system is not in sleep mode.

The coefficient of determination (R-squared) value indicates the goodness of fit of the regression model to the data. In this case, the R-squared value is approximately 0.00077, suggesting that only about 0.077% of the variability in mean power consumption can be explained by the linear relationship with sleep duration.

Lastly, the p-value associated with the regression analysis provides information about the statistical significance of the relationship between sleep duration and power consumption. In our analysis, the p-value is 0.958, indicating that the relationship is not statistically significant at the conventional significance level of 0.05.

Overall, this regression analysis provides insights into how mean power consumption varies with different sleep durations, despite the lack of a statistically significant relationship between the two variables.

4.8 Pydroid3 Application

The results obtained from the Pydroid3 app provide valuable insights into the functionality and performance of the smart sensor system. Through the app's interfaces for data visualization, control, and automation, users can interact with and monitor various aspects of the system in real time.

4.8.1 Data

The data results include a line graph depicting the variations in temperature and humidity sensor readings over one hour. This graph provides insights into how environmental conditions fluctuate over time, offering users a comprehensive view of the changing climate within the monitored area [26]. The graph below represents the temperature and humidity values with the motion data, taken during 1 hour period.

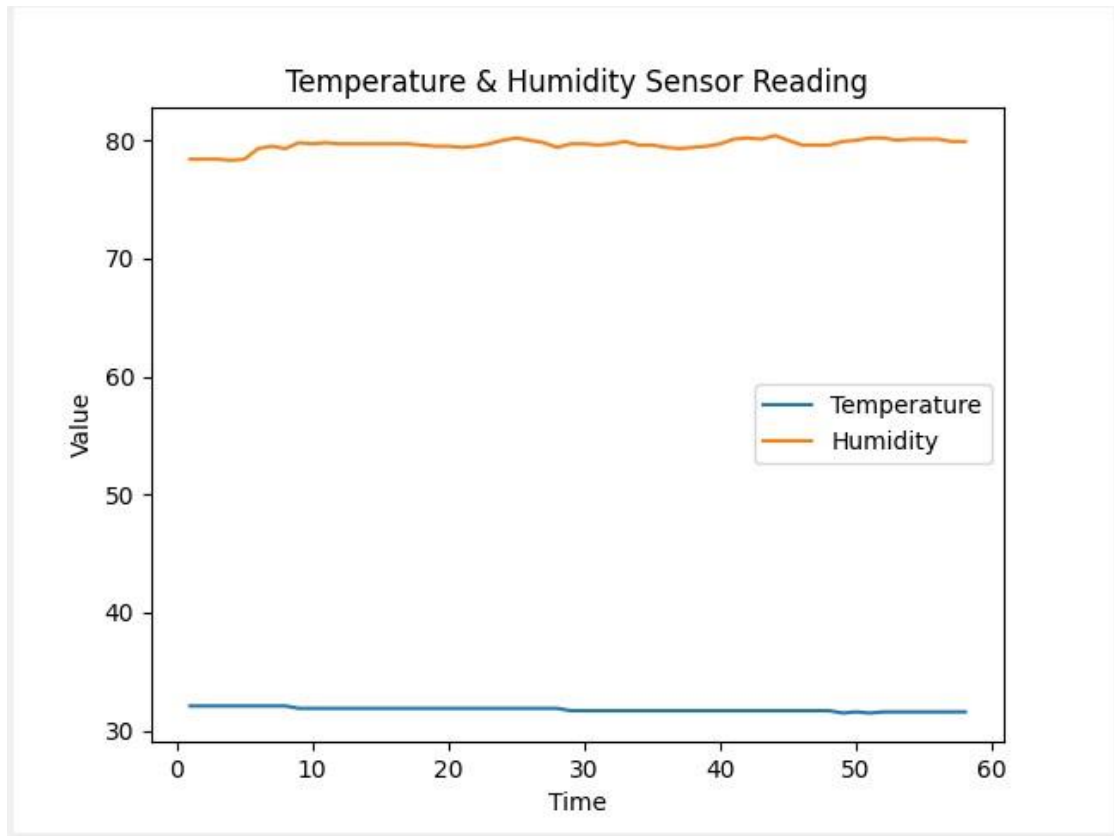


Figure 4.8: Temperature and Humidity Graph

The figure above shows the graph temperature and humidity sensor data. The code generates simulated sensor data for 1 hour period, representing temperature, humidity, and PIR sensor readings. The temperature and humidity data follow a sinusoidal pattern with added randomness to simulate fluctuations. The temperature sensor data is depicted in blue, and the humidity sensor data is shown in orange on the upper subplot of the 2x1 subplot layout. These sinusoidal patterns are adjusted to fall within the range of 30 to 40 degrees Celsius for temperature and 75 to 80 per cent for humidity, representing typical environmental conditions. The patterns provide a visual representation of the fluctuation in temperature and humidity over 1 hour.

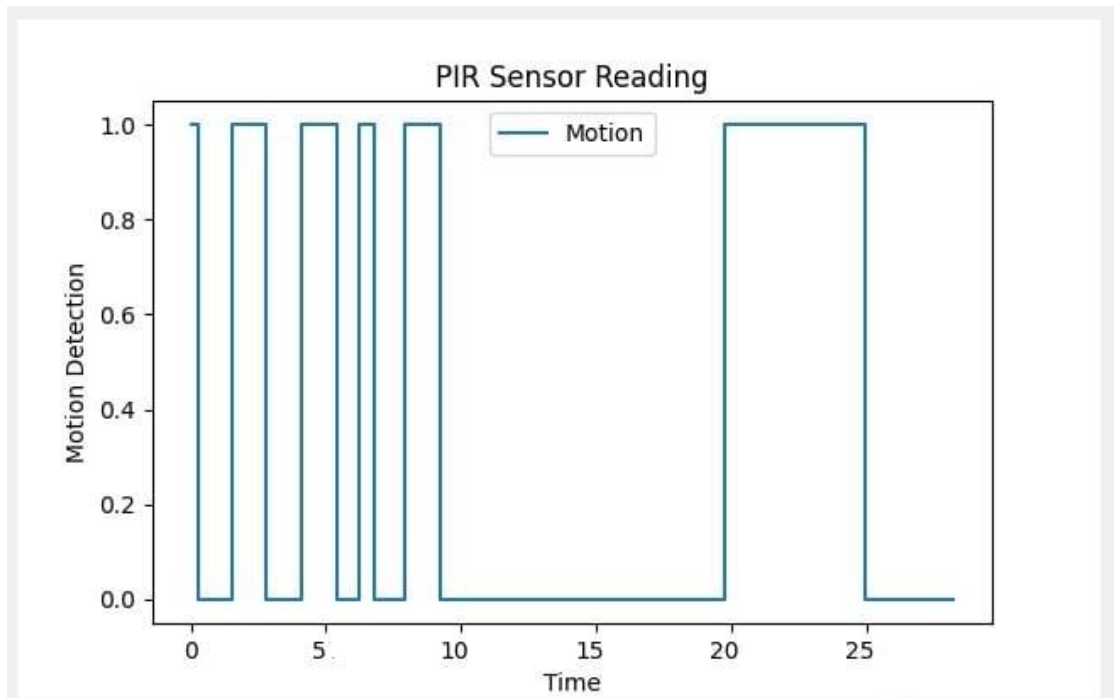


Figure 4.9: PIR Sensor Reading

The PIR sensor data is illustrated using a step plot in blue, showcasing the binary nature of motion detection. On the y-axis of the PIR sensor graph, values of 0 and 1 represent the absence or presence of motion, respectively (0 for no motion, 1 for motion detected). The graph visualizes changes in motion detection status over time, providing insights into when motion events occur within the monitored area. It's worth noting that while the data was collected over one hour, the graph displays motion events only for 25 minutes, highlighting a subset of the overall dataset. This visualization allows users to discern patterns and trends in motion activity, facilitating a better understanding and analysis of motion detection behaviour within the environment.

4.8.2 Control

In the Control Page, user interactions with switches trigger corresponding events, controlling the state of the lamp switch. These interactions are reflected in the terminal output, simulating the data flow in an MQTT cloud within an IoT environment. The terminal serves as a log, capturing user actions and providing real-time monitoring of device states in the IoT system. The Figure below depicts the results of the sequence of events when users activate the turn-on and off buttons, remotely controlling the lamp hardware.



Figure 4.10: Output Display When Changes Made on the Control Button

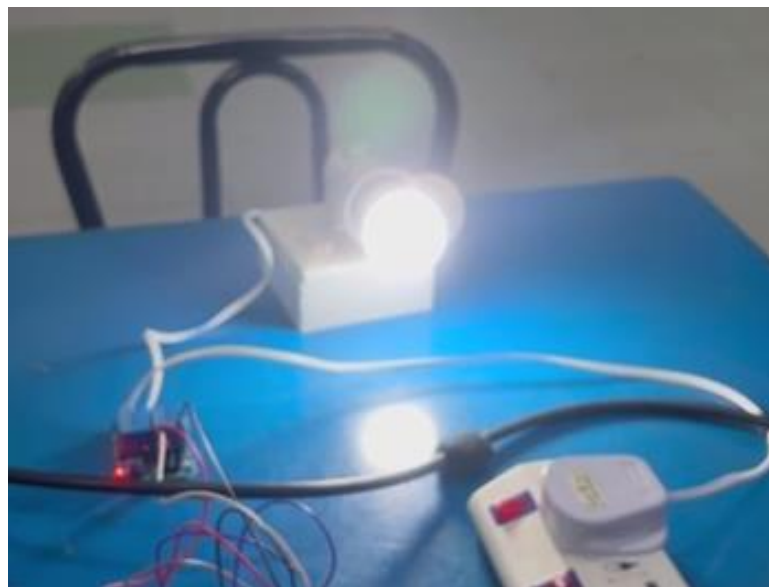


Figure 4.11: Lamp is Turned On



Figure 4.12: When the User Pressed the Turn Off Button on Control Page



Figure 4.13:Lamp is Turned Off

4.8.3 Auto

In the auto interface, the smart sensor system integrates automation rules, enhancing user convenience by allowing predefined conditions to trigger specific actions within the home environment. For instance, if the controller receives a signal indicating motion detection from the sensor, it automatically activates the lamp switch, illuminating the designated area. Conversely, when no motion is detected, the controller initiates the automatic deactivation of the lamp switch, conserving energy and optimizing home security. This automated functionality streamlines user interaction with the smart home system, reducing the need for manual intervention and fostering a more responsive and intuitive home environment. Below are the figures showing the interface that will come out in the Pydroid3 whenever select the rules



Figure 4.14: User Select to Turn On Lamp Whenever Motion is Detected

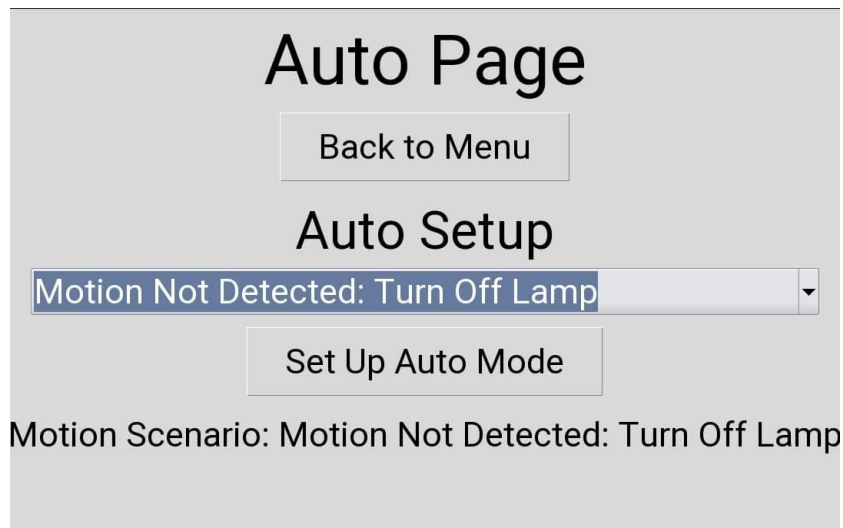


Figure 4.15: User Selecting to Turn Off The Lamp Whenever There is No Motion Detected

While the system demonstrates the capability to automatically respond to predefined conditions such as motion detection, certain limitations may affect its performance. One potential limitation involves delays in communication between the MQTT broker and the Pydroid3 application, which could result in extended wait times for establishing connections or receiving updates. This delay may impact the responsiveness of the system, leading to slower interactions when activating or deactivating the lamp switch. Below is an example of a connection error that might happen during the data transmission.

```

C:\Users\User>cd..
C:\Users>cd..
C:\>cd "Program Files"
C:\Program Files>cd mosquitto
C:\Program Files\mosquitto>mosquitto_sub.exe -h 115.187.22.64 -u public -P "public!@#" -t /public/# -v
Error: A connection attempt failed because the connected party did not properly respond after a period of
time, or established connection failed because connected host has failed to respond.
C:\Program Files\mosquitto>

```

Figure 4.16: Network Error Occurs during Data Transmission.

Furthermore, network congestion or connectivity issues could exacerbate these delays, potentially hindering the smooth operation of the smart sensor system. Additionally, dependencies on external factors such as internet stability or server reliability may introduce unpredictability into the system's behaviour, necessitating

CHAPTER FIVE

CONCLUSION

5.1 Conclusion

The development of the smart sensor system for home automation marks a significant advancement in household management, offering enhanced efficiency and responsiveness. Leveraging technologies such as Bluetooth Low Energy (BLE) communication, MQTT protocol, and deep sleep features, the system demonstrates notable improvements in energy efficiency, responsiveness, and overall functionality. With features like real-time data display in the Pydroid3 app, automation rules, and deep sleep mode, the system showcases its versatility and adaptability to diverse user needs and scenarios.

5.2 Recommendation for Future Work

Looking ahead, several avenues for future work and enhancement present themselves. One promising direction involves integrating machine learning algorithms to enable adaptive automation, allowing the system to learn from user preferences and environmental patterns to optimize operations further. Additionally, explore integrating brightness control for the lamp based on ambient temperature. By implementing algorithms that adjust lamp brightness in response to ambient temperature changes, the system can enhance user comfort and energy efficiency [27]. Furthermore, prioritizing robust encryption measures to safeguard user privacy and data security remains paramount, especially in interconnected smart home ecosystems.

REFERENCES

- [1] A. P. Vancea and I. Orha, "Smart home automation and monitoring system," *Carpathian Journal of Electronic and Computer Engineering*, vol. 11, no. 1, pp. 40–43, Sep. 2018, doi: 10.2478/cjece-2018-0007.
- [2] R. Jurdak, A. G. Ruzzelli, and G. M. P. O'Hare, "Radio Sleep Mode Optimization in Wireless Sensor Networks," *IEEE Trans Mob Comput*, vol. 9, no. 7, pp. 955–968, Jul. 2010, doi: 10.1109/TMC.2010.35.
- [3] X. Mao, K. Li, Z. Zhang, and J. Liang, "Design and implementation of a new smart home control system based on internet of things," in *2017 International Smart Cities Conference (ISC2)*, IEEE, Sep. 2017, pp. 1–5. doi: 10.1109/ISC2.2017.8090790.
- [4] S. Kumar and S. R. Lee, "Android based smart home system with control via Bluetooth and internet connectivity," in *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, IEEE, Jun. 2014, pp. 1–2. doi: 10.1109/ISCE.2014.6884302.
- [5] *2019 10th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)*.
- [6] R. Colella *et al.*, "2.4 GHz BLE-based smart sensing system for remote monitoring of health, safety and comfort at workplace," in *2021 6th International Conference on Smart and Sustainable Technologies, SpliTech 2021*, Institute of Electrical and Electronics Engineers Inc., Sep. 2021. doi: 10.23919/SpliTech52315.2021.9566398.
- [7] F. Yao, J. Wu, G. Venkataramani, and S. Subramaniam, "A dual delay timer strategy for optimizing server farm energy," in *Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015*, Institute of Electrical and Electronics Engineers Inc., Feb. 2016, pp. 258–265. doi: 10.1109/CloudCom.2015.75.
- [8] M. S. Abdul, S. M. Sam, N. Mohamed, N. H. Hassan, A. Azizan, and Y. M. Yusof, "Peer to Peer Communication for the Internet of Things Using ESP32 Microcontroller for Indoor Environments," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, Oct. 2022, pp. 1–6. doi: 10.1109/ICTC55196.2022.9952832.

- [9] C. Nandi and M. D. Ernst, “Automatic Trigger Generation for Rule-based Smart Homes,” in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, New York, NY, USA: ACM, Oct. 2016, pp. 97–102. doi: 10.1145/2993600.2993601.
- [10] D. R. Recupero *et al.*, “Leveraging the Arduino Platform to Develop Information Technology Devices,” in *Encyclopedia of Information Science and Technology, Fourth Edition*, IGI Global, 2018, pp. 3273–3286. doi: 10.4018/978-1-5225-2255-3.ch285.
- [11] D. I. Costean and S. Mischie, “BLE Network Sensors with IoT Capability,” in *2022 International Symposium on Electronics and Telecommunications (ISETC)*, IEEE, Nov. 2022, pp. 1–4. doi: 10.1109/ISETC56213.2022.10010090.
- [12] H.-J. Zhu, H.-R. Wu, L.-H. Zhang, and Y.-S. Miao, “The Irregularity Propagation Characteristics of Radio Signals For Wireless Sensor Network In Farmland,” *ITM Web of Conferences*, vol. 7, p. 01013, Nov. 2016, doi: 10.1051/itmconf/20160701013.
- [13] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, and D. Formica, “Performance Evaluation of Bluetooth Low Energy: A Systematic Review,” *Sensors*, vol. 17, no. 12, p. 2898, Dec. 2017, doi: 10.3390/s17122898.
- [14] W. Bronzi, R. Frank, G. Castignani, and T. Engel, “Bluetooth Low Energy performance and robustness analysis for Inter-Vehicular Communications,” *Ad Hoc Networks*, vol. 37, pp. 76–86, Feb. 2016, doi: 10.1016/j.adhoc.2015.08.007.
- [15] S. Chai, R. An, and Z. Du, “An Indoor Positioning Algorithm using Bluetooth Low Energy RSSI,” in *Proceedings of the 2016 International Conference on Advanced Materials Science and Environmental Engineering*, Paris, France: Atlantis Press, 2016. doi: 10.2991/amsee-16.2016.72.
- [16] V. V. Tipparaju, K. R. Mallires, D. Wang, F. Tsow, and X. Xian, “Mitigation of Data Packet Loss in Bluetooth Low Energy-Based Wearable Healthcare Ecosystem,” *Biosensors (Basel)*, vol. 11, no. 10, p. 350, Sep. 2021, doi: 10.3390/bios11100350.
- [17] M.-G. Kim, J. Choi, and M. Kang, “Enhanced power-saving mechanism to maximize operational efficiency in IEEE 802.16e systems,” *IEEE Trans Wirel Commun*, vol. 8, no. 9, pp. 4710–4719, Sep. 2009, doi:

10.1109/TWC.2009.081151.

- [18] H. J. Lee, “Energy Conservation Strategy for Sensor Networks,” 2005, pp. 413–413. doi: 10.1007/11502593_44.
- [19] R. Kallimani and K. Rasane, “Investigation of Power Consumption in Microcontroller Based Systems,” 2020, pp. 404–411. doi: 10.1007/978-3-030-28364-3_40.
- [20] M.-P. St-Onge *et al.*, “Short sleep duration increases energy intakes but does not change energy expenditure in normal-weight individuals,” *Am J Clin Nutr*, vol. 94, no. 2, pp. 410–416, Aug. 2011, doi: 10.3945/ajcn.111.013904.
- [21] Ü. Güler and R. Sokullu, “Remote Monitoring and Control of Refrigerators Through MQTT Protocol with AES Encryption,” in *2023 International Balkan Conference on Communications and Networking, BalkanCom 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/BalkanCom58402.2023.10167993.
- [22] A. Radhanand, K. N. B. Kumar, S. Namburu, and P. Sampathkrishna Reddy, “Implementation of a Distributed Home Automation Scheme with Custom Hardware Nodes Using ZigBee and MQTT Protocols,” *IETE J Res*, vol. 67, no. 5, pp. 596–602, Sep. 2021, doi: 10.1080/03772063.2021.1923078.
- [23] G. Mois, S. Folea, and T. Sanislav, “Analysis of Three IoT-Based Wireless Sensors for Environmental Monitoring,” *IEEE Trans Instrum Meas*, vol. 66, no. 8, pp. 2056–2064, Aug. 2017, doi: 10.1109/TIM.2017.2677619.
- [24] M. Walch, M. Rietzler, J. Greim, F. Schaub, B. Wiedersheim, and M. Weber, “homeBLOX,” in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, New York, NY, USA: ACM, Sep. 2013, pp. 295–298. doi: 10.1145/2494091.2494182.
- [25] H. Vo, “Implementing Energy Saving Techniques for Sensor Nodes in IoT Applications,” *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 5, no. 17, p. 156076, Dec. 2018, doi: 10.4108/eai.19-12-2018.156076.
- [26] J. Lundstrom, J. Synnott, E. Jarpe, and C. D. Nugent, “Smart home simulation using avatar control and probabilistic sampling,” in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, IEEE, Mar. 2015, pp. 336–341. doi: 10.1109/PERCOMW.2015.7134059.

- [27] G. M. Huebner, D. T. Shipworth, S. Gauthier, C. Witzel, P. Raynham, and W. Chan, "Saving energy with light? Experimental studies assessing the impact of colour temperature on thermal comfort," *Energy Res Soc Sci*, vol. 15, pp. 45–57, May 2016, doi: 10.1016/j.erss.2016.02.008.

APPENDICES

APPENDIX 1

Code Sendor for Temperature and Humidity Sensor Using BLE Comuunication

```
send.ino
1  #include <BLEDevice.h>
2  #include <BLEUtils.h>
3  #include <BLEServer.h>
4  #include <Adafruit_Sensor.h>
5  #include <DHT.h>
6
7  DHT dht(4, DHT22); // DHT sensor connected to GPIO 4
8
9  BLECharacteristic *pCharacteristic;
10 bool isCharacteristicReadable = false;
11
12 const int switchPin = 14; // GPIO 14
13 const int ledPin = 13;    // GPIO 13
14
15 unsigned long startTime = 0;
16 unsigned long duration = 600000; // 6 minutes in milliseconds
17 unsigned long sleepDuration = 20000; // 20 seconds in milliseconds
18
19 #define SERVICE_UUID      "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
20 #define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
21
22 unsigned long lastSignalTime = 0;
23 int transmissionCounter = 0; // Counter for transmissions
24
25 class MyServerCallbacks : public BLEServerCallbacks {
26   void onConnect(BLEServer* server) {
27     Serial.println("Connected to BLE server");
28   }
29
30   void onDisconnect(BLEServer* server) {
31     Serial.println("Disconnected from BLE server");
32     isCharacteristicReadable = false; // Reset the flag on disconnect
33   }
34 };
35
36 void setup() {
37   Serial.begin(115200);
38   dht.begin();
39   pinMode(switchPin, INPUT_PULLUP);
40   pinMode(ledPin, OUTPUT);
41
42   Serial.println("Connecting to BLE server");
43   BLEDevice::init("Data Sender");
44   BLEServer *pServer = BLEDevice::createServer();
45   pServer->setCallbacks(new MyServerCallbacks());
46
47   BLEService *pService = pServer->createService(SERVICE_UUID);
48   pCharacteristic = pService->createCharacteristic(
49     CHARACTERISTIC_UUID,
50     BLECharacteristic::PROPERTY_READ |
51     BLECharacteristic::PROPERTY_NOTIFY
52   );
53
54   pService->start();
55   pServer->getAdvertising()->addServiceUUID(SERVICE_UUID);
56
57   BLEAdvertising *pAdvertising = pServer->getAdvertising();
58   pAdvertising->start();
59 }
60
61 void loop() {
62   // Read the state of the switch
63   int switchState = digitalRead(switchPin);
64
65   if (switchState == LOW) {
66     isCharacteristicReadable = true; // Enable sensor reading when switch is ON
67   } else {
```

```

68 | isCharacteristicReadable = false; // Disable sensor reading when switch is OFF
69 | }
70 |
71 | unsigned long currentTime = millis();
72 |
73 | // Check if it's time to send data
74 | if (isCharacteristicReadable && (currentTime - startTime < duration)) {
75 |     // Read temperature and humidity from DHT sensor
76 |     float temperature = dht.readTemperature();
77 |     float humidity = dht.readHumidity();
78 |
79 |     if (!isnan(temperature) && !isnan(humidity)) {
80 |         Serial.print("Temperature: ");
81 |         Serial.print(temperature);
82 |         Serial.println(" °C");
83 |         Serial.print("Humidity: ");
84 |         Serial.print(humidity);
85 |         Serial.println("%");
86 |
87 |         // Turn on the LED when the sensor is actively working
88 |         digitalWrite(ledPin, HIGH);
89 |
90 |         // Increment transmission counter
91 |         transmissionCounter++;
92 |
93 |         // Sender's start time for transmission
94 |         unsigned long transmissionStartTime = millis();
95 |         Serial.print("Start Time ");
96 |         Serial.print(transmissionCounter);
97 |         Serial.print(": ");
98 |         Serial.println(transmissionStartTime);
99 |
100 |         // Receiver requested data, send it
101 |         String dataToSend = "Temperature: " + String(temperature) + " °C, Humidity: " + String(humidity) + "%";
102 |         pCharacteristic->setValue(dataToSend.c_str());
103 |         pCharacteristic->notify();
104 |         Serial.print("Sent ");
105 |         Serial.print(transmissionCounter);
106 |         Serial.print(": ");
107 |         Serial.println(dataToSend);
108 |     }
109 |
110 |     // Wait for a short time to allow the receiver to process the data
111 |     delay(1000);
112 | } else {
113 |     // Turn off the LED when the switch is OFF or not requesting data
114 |     digitalWrite(ledPin, LOW);
115 |
116 |     // Display "Going to sleep mode" when not requesting data
117 |     Serial.println("Going to sleep mode.");
118 |
119 |     // Enter deep sleep mode for sleepDuration
120 |     delay(sleepDuration);
121 |     // Reset the start time
122 |     startTime = millis();
123 |     // Go to deep sleep mode
124 |     esp_sleep_enable_timer_wakeup(sleepDuration * 1000);
125 |     esp_deep_sleep_start();
126 | }
127 | }
128 |

```

Code Receiver for Temperature and Humidity Sensor Using BLE Communication

```

1 | #include <BLEDevice.h>
2 | #include <BLEUtils.h>
3 | #include <BLEServer.h>
4 | #include <BLE2902.h>
5 |
6 | BLEClient* pClient;
7 | BLERemoteCharacteristic* pRemoteCharacteristic;
8 | bool deviceConnected = false;
9 |
10 | BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");
11 | BLEUUID charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");
12 |
13 | class MyClientCallbacks : public BLEClientCallbacks {
14 |     void onConnect(BLEClient* client) {
15 |         deviceConnected = true;
16 |     }
17 |
18 |     void onDisconnect(BLEClient* client) {
19 |         deviceConnected = false;
20 |         pRemoteCharacteristic = nullptr; // Reset the remote characteristic when disconnected
21 |     }
22 | };
23 |
24 | // Function to extract start time from received data
25 | unsigned long getStartTime(String receivedData) {
26 |     int startIndex = receivedData.indexOf("Start Time ") + 11;
27 |     int endIndex = receivedData.indexOf(":", startIndex);
28 |     if (startIndex != -1 && endIndex != -1) {
29 |         return receivedData.substring(startIndex, endIndex).toInt();
30 |     } else {
31 |         return 0; // If start time not found, return 0
32 |     }
33 | }
34 |

```

```

35 void setup() {
36     Serial.begin(115200);
37     Serial.println("Scanning for BLE server...");
38
39     BLEDevice::init("ESP32 Receiver");
40     pClient = BLEDevice::createClient();
41     pClient->setClientCallbacks(new MyClientCallbacks());
42
43     BLEScan* pBLEScan = BLEDevice::getScan();
44     pBLEScan->setActiveScan(true);
45     pBLEScan->setInterval(100);
46     pBLEScan->setWindow(99);
47
48     BLEScanResults foundDevices = pBLEScan->start(5, false);
49
50     for (int i = 0; i < foundDevices.getCount(); i++) {
51         BLEAdvertisedDevice device = foundDevices.getDevice(i);
52         if (device.hasServiceUUID() && device.isAdvertisingService(serviceUUID)) {
53             BLEAddress address = device.getAddress();
54             esp_ble_addr_type_t addressType = device.getAddressType();
55             pClient->connect(address, addressType);
56             break;
57         }
58     }
59 }
60
61 void loop() {
62     if (deviceConnected) {
63         if (pRemoteCharacteristic == nullptr) {
64             pRemoteCharacteristic = pClient->getService(serviceUUID)->getCharacteristic(charUUID);
65             pRemoteCharacteristic->registerForNotify(nullptr);
66         } else {
67             if (pRemoteCharacteristic->canRead()) {
68                 std::string value = pRemoteCharacteristic->readValue();
69                 Serial.print("Received Value: ");
70                 Serial.println(value.c_str());
71
72                 // Convert std::string to String
73                 String stringValue(value.c_str());
74
75                 // Parse transmission number from received data
76                 int transmissionNumber = getTransmissionNumber(stringValue);
77
78                 // Print the received data with transmission counter
79                 Serial.print("Received ");
80                 Serial.print(transmissionNumber);
81                 Serial.print(": ");
82                 Serial.println(stringValue);
83
84                 // Parse start time and end time from received data
85                 unsigned long startTime = getStartTime(stringValue);
86                 unsigned long endTime = millis();
87
88                 // Display start time and end time
89                 Serial.print("Start Time ");
90                 Serial.print(transmissionNumber);
91                 Serial.print(": ");
92                 Serial.println(startTime);
93                 Serial.print("End Time ");
94                 Serial.print(transmissionNumber);
95                 Serial.print(": ");
96                 Serial.println(endTime);
97
98                 // Calculate transmission duration
99                 unsigned long transmissionDuration = endTime - startTime;
100                 Serial.print("Transmission Duration ");
101                 Serial.print(transmissionNumber);
102                 Serial.print(": ");
103                 Serial.println(transmissionDuration);
104             }
105         }
106     }
107
108     delay(2000);
109 }
110
111 // Function to extract transmission number from received data
112 int getTransmissionNumber(String receivedData) {
113     int startIndex = receivedData.indexOf("Sent ") + 5;
114     int endIndex = receivedData.indexOf(":", startIndex);
115     return receivedData.substring(startIndex, endIndex).toInt();
116 }
117

```

Code Sender for Motion Sensor Using BLE Communication

```
1 #include <BLEDevice.h>
2 #include <BLEUtils.h>
3 #include <BLEServer.h>
4 #include <esp_sleep.h>
5
6 #define SERVICE_UUID          "8173fb30-9552-4593-b3f4-2ed2ee12f5b8"
7 #define CHARACTERISTIC_UUID   "d4179c77-ab1d-4f9c-ab24-a8f4d7417ad3"
8
9 BLECharacteristic *pCharacteristic;
10
11 const int motionPin = 2; // HC-SR501 PIR motion sensor connected to GPIO 2
12 const int ledPin = 13; // Green LED connected to GPIO 13
13 const int switchPin = 12; // 3-pin switch connected to GPIO 12
14
15 bool isSensorEnabled = false; // Flag to track whether the motion sensor is enabled
16
17 unsigned long lastMotionTime = 0;
18 unsigned long motionDelay = 2000; // Delay to display motion detection results (2 seconds)
19 unsigned long lastActivityTime = 0;
20 unsigned long publishDuration = 600000; // 1 minute in milliseconds
21 unsigned long sleepDuration = 20000; // 20 seconds in milliseconds
22
23 void setup() {
24     pinMode(motionPin, INPUT);
25     pinMode(ledPin, OUTPUT);
26     pinMode(switchPin, INPUT_PULLUP); // Use internal pull-up resistor
27
28     digitalWrite(ledPin, LOW); // Ensure the LED is initially off
29     Serial.begin(115200);
30
31     // BLE Setup
32     BLEDevice::init("Motion Sensor Device");
33
34     BLEServer *pServer = BLEDevice::createServer();
35     BLEService *pService = pServer->createService(SERVICE_UUID);
36     pCharacteristic = pService->createCharacteristic(
37         CHARACTERISTIC_UUID,
38         BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY
39     );
40     pCharacteristic->setValue("Connected to Receiver");
41     pService->start();
42     BLEAdvertising *pAdvertising = pServer->getAdvertising();
43     pAdvertising->addServiceUUID(SERVICE_UUID);
44     pAdvertising->setScanResponse(true);
45     pAdvertising->setMinPreferred(0x06);
46     pAdvertising->setMinPreferred(0x12);
47     BLEDevice::startAdvertising();
48
49 void loop() {
50     // Check the state of the switch
51     bool switchState = digitalRead(switchPin);
52
53     // If the switch is ON, enable the motion sensor
54     if (switchState == LOW) {
55         isSensorEnabled = true;
56         digitalWrite(ledPin, HIGH); // Turn on the LED when motion sensor is enabled
57         Serial.println("Motion Sensor is enabled.");
58     } else {
59         isSensorEnabled = false;
60         digitalWrite(ledPin, LOW); // Turn off the LED
61         Serial.println("Switch is off. Sensor in sleep mode.");
62         // Enter deep sleep mode for sleep duration
63         Serial.println("Entering deep sleep mode for sleep duration.");
64         esp_sleep_enable_timer_wakeup(sleepDuration * 1000); // Convert to microseconds
65
66         esp_deep_sleep_start();
67     }
68
69     if (isSensorEnabled) {
70         unsigned long currentMillis = millis();
71         if (currentMillis - lastMotionTime >= motionDelay) {
72             int motionState = digitalRead(motionPin);
73             if (motionState == HIGH) {
74                 String motionDetected = "Motion detected.";
75                 Serial.println(motionDetected);
76                 pCharacteristic->setValue(motionDetected.c_str());
77                 pCharacteristic->notify(); // Notify the central device (receiver)
78                 digitalWrite(ledPin, HIGH); // Turn on the LED when motion is detected
79                 delay(200); // Blink for 200 milliseconds
80                 digitalWrite(ledPin, LOW); // Turn off the LED
81                 lastMotionTime = currentMillis;
82                 delay(1800); // Delay for 1.8 seconds before displaying "No motion detected"
83             } else {
84                 digitalWrite(ledPin, LOW); // Turn off the LED when there's no motion
85                 pCharacteristic->setValue("No motion detected.");
86                 pCharacteristic->notify(); // Notify the central device (receiver)
87                 Serial.println("No motion detected."); // Print "No motion detected"
88                 delay(2000); // Delay for 2 seconds before displaying "Motion detected"
89             }
90         }
91     }
92
93     // Check for inactivity to initiate sleep
94     if (millis() - lastActivityTime >= publishDuration) {
95         Serial.println("Inactivity detected. Initiating sleep.");
96         // Enter deep sleep mode for sleep duration
97         esp_sleep_enable_timer_wakeup(sleepDuration * 1000); // Convert to microseconds
98
99         esp_deep_sleep_start();
100     }
101 }
```

Code Receiver for Motion Sensor Using BLE Comuunication

```
1  #include <BLEDevice.h>
2  #include <BLEUtils.h>
3  #include <BLEServer.h>
4
5  BLEClient* pClient;
6  BLEScan* pBLEScan;
7  bool deviceConnected = false;
8  BLEUUID serviceUUID("8173fb30-9552-4593-b3f4-2ed2ee12f5b8");
9  BLEUUID charUUID("d4179c77-ab1d-4f9c-ab24-a8f4d7417ad3");
10
11 class MyServerCallbacks : public BLEServerCallbacks {
12     void onConnect(BLEServer* pServer) {
13         Serial.println("Connected to client");
14         deviceConnected = true;
15     }
16
17     void onDisconnect(BLEServer* pServer) {
18         Serial.println("Disconnected from client");
19         deviceConnected = false;
20     }
21 };
22
23 void setup() {
24     Serial.begin(115200);
25     BLEDevice::init("Receiver");
26     pBLEScan = BLEDevice::getScan();
27     pBLEScan->setActiveScan(true);
28     pBLEScan->setInterval(100);
29     pBLEScan->setWindow(99);
30
31     BLEServer* pServer = BLEDevice::createServer();
32     pServer->setCallbacks(new MyServerCallbacks());
33
34     delay(500); // Give the BLE server some time to start before scanning
35
36     // Rest of your setup code
37 }
38
39 void loop() {
40     if (!deviceConnected) {
41         BLEScanResults foundDevices = pBLEScan->start(5, false);
42
43         for (int i = 0; i < foundDevices.getCount(); i++) {
44             BLEAdvertisedDevice device = foundDevices.getDevice(i);
45             if (device.hasServiceUUID() && device.isAdvertisingService(serviceUUID)) {
46                 BLEAddress address = device.getAddress();
47                 esp_ble_addr_type_t addressType = device.getAddressType();
48                 pClient = BLEDevice::createClient();
49                 pClient->connect(address, addressType);
50             }
51         }
52     } else {
53         if (pClient->isConnected()) {
54             BLERemoteCharacteristic* pCharacteristic = pClient->getService(serviceUUID)->getCharacteristic(charUUID);
55             if (pCharacteristic->canRead()) {
56                 std::string value = pCharacteristic->readValue();
57                 Serial.print("Received value: ");
58                 Serial.println(value.c_str());
59                 // Add any processing or handling of the received value here
60             }
61         } else {
62             deviceConnected = false;
63             pClient->disconnect();
64         }
65     }
66
67     delay(10000); // Wait for 10 seconds before the next scan
68 }
69 }
```

Code For Deep Sleep Mode

```
// Check the state of the switch
bool switchState = digitalRead(switchPin);

// If the switch is OFF, enter deep sleep mode
if (switchState == HIGH) {
    Serial.println("Switch is off. Sensor in sleep mode.");

    // Enter deep sleep mode for sleep duration
    Serial.println("Entering deep sleep mode for sleep duration.");
    esp_sleep_enable_timer_wakeup(sleepDuration * 1000); // Convert to microseconds
    esp_deep_sleep_start();
}
```

APPENDIX 2

Code Pydroid3

```
1 import tkinter as tk
2 from tkinter import ttk
3 from tkinter import messagebox
4 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import random # Add this import statement
8 import datetime
9 import threading
10 import paho.mqtt.client as mqtt
11 import time
12
13 class SmartSensorApp(tk.Tk):
14     def __init__(self, *args, **kwargs):
15         tk.Tk.__init__(self, *args, **kwargs)
16
17         # Configure main window
18         tk.Wm.title(self, "SmartHome IoT")
19
20         # Create a color-changing title label with left-to-right wave effect
21         self.title_label = ColorChangingLabel(self, text="Welcome to SmartHome IoT", font=("Helvetica", 13, "bold"))
22         self.title_label.pack(pady=10)
23
24         # Start the wave animation after a 1-second delay
25         self.title_label.start_animation()
26
27         # Create container for frames
28         container = tk.Frame(self)
29         container.pack(side="top", fill="both", expand=True)
30         container.grid_rowconfigure(0, weight=1)
31         container.grid_columnconfigure(0, weight=1)
32
33         self.frames = {}
34         for F in (DataPage, ControlPage, AutoPage, SensorPage, SwitchPage, MainMenuPage):
35             frame = F(container, self)
36             self.frames[F] = frame
37             frame.pack(side="top", fill="both", expand=True)
38
39         # Create a menu
40         menubar = tk.Menu(self)
41         self.config(menu=menubar)
42
43         # Create a file menu
44         file_menu = tk.Menu(menubar, tearoff=0)
45         menubar.add_cascade(label="File", menu=file_menu)
46
47         # Add Data, Control, and Auto options to the file menu
48         file_menu.add_command(label="Data", command=lambda: self.show_frame(DataPage))
49         file_menu.add_command(label="Control", command=lambda: self.show_frame(ControlPage))
50         file_menu.add_command(label="Auto", command=lambda: self.show_frame(AutoPage))
51
52         # Add a separator in the file menu
53         file_menu.add_separator()
54
55         # Add an Exit option to the file menu
56         file_menu.add_command(label="Exit", command=self.exit_program)
57
58         # Create a Sensors menu
59         sensors_menu = tk.Menu(menubar, tearoff=0)
60         menubar.add_cascade(label="Sensors", menu=sensors_menu)
61
62         # Add a "Deep Sleep" option under the "File" menu
63         file_menu.add_command(label="Deep Sleep", command=lambda: self.show_frame(DeepSleepPage))
64
65         # Add Temperature & Humidity sensor option
66         sensors_menu.add_command(label="Temperature & Humidity", command=lambda: self.show_frame(SensorPage, "Temperature & Humidity"))
67
68         # Add PIR sensor option
69         sensors_menu.add_command(label="PIR Sensor", command=lambda: self.show_frame(SensorPage, "PIR Sensor"))
70
71         # Create a Switches menu
72         switches_menu = tk.Menu(menubar, tearoff=0)
73         menubar.add_cascade(label="Switches", menu=switches_menu)
74
75         # Add Lamp switch option
76         switches_menu.add_command(label="Lamp", command=lambda: self.show_frame(SwitchPage, "Lamp"))
77
78         # Initially, main menu page is displayed
79         self.show_frame(MainMenuPage)
80         self.current_frame = MainMenuPage
81         self.last_displayed_content = None
82
83         # MQTT client setup
84         self.client = mqtt.Client()
85         self.client.username_pw_set(username="public", password="public!@#")
86         self.client.on_connect = self.on_connect
87         self.client.on_message = self.on_message
88         self.client.connect(("115.187.22.64", 1883, 60))
89         self.client.loop_start()
90
91     def show_frame(self, cont, sensor_or_switch=None):
92         for frame in self.frames.values():
93             frame.pack_forget() # Hide all frames
94
95         frame = self.frames[cont]
96         frame.pack(side="top", fill="both", expand=True)
97
98         if sensor_or_switch:
99             frame.update_label(sensor_or_switch)
100
101         if cont != MainMenuPage:
102             self.last_displayed_content = cont
103
104         self.current_frame = frame
105
106     def exit_program(self):
107         result = messagebox.askokcancel("Exit", "Are you sure you want to exit?")
108         if result:
109             self.destroy()
110
```

```

111
112 def on_connect(self, client, userdata, flags, rc):
113     print("Connected with result code "+str(rc))
114     client.subscribe("/public/amir/temperature")
115     client.subscribe("/public/amir/humidity")
116     client.subscribe("/public/amir/motion")
117
118 def on_message(self, client, userdata, msg):
119     print("Received message: Topic="+msg.topic, Payload=msg.payload)
120     if msg.topic == "/public/amir/temperature":
121         if msg.payload is not None:
122             self.frames[DataPage].update_temp(float(msg.payload.decode()))
123     elif msg.topic == "/public/amir/humidity":
124         if msg.payload is not None:
125             self.frames[DataPage].update_humidity(float(msg.payload.decode()))
126     elif msg.topic == "/public/amir/motion":
127         if msg.payload is not None:
128             self.frames[DataPage].update_motion(msg.payload.decode())
129
130
131 class ColorChangingLabel(tk.Label):
132     def __init__(self, master=None, **kwargs):
133         tk.Label.__init__(self, master, **kwargs)
134         self.color_index = 0
135         self.delay_before_start = 2000 # 2 seconds delay
136         self.after(self.delay_before_start, self.start_animation)
137
138     def color_wave(self):
139         r = int(127.5 + 127.5 * np.sin(2.0 * np.pi * (0.5 + self.color_index)))
140         g = int(127.5 + 127.5 * np.sin(2.0 * np.pi * (0.0 + self.color_index)))
141         b = int(127.5 + 127.5 * np.sin(2.0 * np.pi * (1.0 + self.color_index)))
142
143         self.config(foreground="#02x02x02x" % (r, g, b))
144         self.color_index += 0.02
145         self.after(50, self.color_wave)
146
147     def start_animation(self):
148         self.color_wave()
149
150
151 class DataPage(tk.Frame):
152     def __init__(self, parent, controller):
153         tk.Frame.__init__(self, parent)
154
155         self.fig_temp_humidity, self.ax_temp_humidity = plt.subplots()
156         self.ax_temp_humidity.set_xlabel("Time")
157         self.ax_temp_humidity.set_ylabel("Value")
158         self.ax_temp_humidity.set_title("Temperature & Humidity Sensor Reading")
159         self.line_temp = self.ax_temp_humidity.plot([], [], label="Temperature")
160         self.line_humidity = self.ax_temp_humidity.plot([], [], label="Humidity")
161         self.ax_temp_humidity.legend()
162
163         self.fig_motion, self.ax_motion = plt.subplots()
164         self.ax_motion.set_xlabel("Time")
165         self.ax_motion.set_ylabel("Motion Detection")
166         self.ax_motion.set_title("PIR Sensor Reading")
167         self.line_motion = self.ax_motion.step([], [], where="post", label="Motion")
168         self.ax_motion.legend()
169
170         self.canvas_temp_humidity = FigureCanvasTkAgg(self.fig_temp_humidity, master=self)
171         self.canvas_temp_humidity.get_tk_widget().pack(side="top", pady=10)
172
173         self.canvas_motion = FigureCanvasTkAgg(self.fig_motion, master=self)
174         self.canvas_motion.get_tk_widget().pack(side="top", pady=10)
175
176         self.time_data = []
177         self.temp_data = []
178         self.humidity_data = []
179         self.motion_data = []
180
181         self.motion_detected = False
182         self.motion_timer = None
183
111
112 def on_connect(self, client, userdata, flags, rc):
113     print("Connected with result code "+str(rc))
114     client.subscribe("/public/amir/temperature")
115     client.subscribe("/public/amir/humidity")
116     client.subscribe("/public/amir/motion")
117
118 def on_message(self, client, userdata, msg):
119     print("Received message: Topic="+msg.topic, Payload=msg.payload)
120     if msg.topic == "/public/amir/temperature":
121         if msg.payload is not None:
122             self.frames[DataPage].update_temp(float(msg.payload.decode()))
123     elif msg.topic == "/public/amir/humidity":
124         if msg.payload is not None:
125             self.frames[DataPage].update_humidity(float(msg.payload.decode()))
126     elif msg.topic == "/public/amir/motion":
127         if msg.payload is not None:
128             self.frames[DataPage].update_motion(msg.payload.decode())
129
130
131 class ColorChangingLabel(tk.Label):
132     def __init__(self, master=None, **kwargs):
133         tk.Label.__init__(self, master, **kwargs)
134         self.color_index = 0
135         self.delay_before_start = 2000 # 2 seconds delay
136         self.after(self.delay_before_start, self.start_animation)
137
138     def color_wave(self):
139         r = int(127.5 + 127.5 * np.sin(2.0 * np.pi * (0.5 + self.color_index)))
140         g = int(127.5 + 127.5 * np.sin(2.0 * np.pi * (0.0 + self.color_index)))
141         b = int(127.5 + 127.5 * np.sin(2.0 * np.pi * (1.0 + self.color_index)))
142
143         self.config(foreground="#02x02x02x" % (r, g, b))
144         self.color_index += 0.02
145         self.after(50, self.color_wave)
146

```



```

184 def update_temp(self, temp):
185     self.time_data.append(len(self.time_data) + 1)
186     self.temp_data.append(temp)
187     self.line_temp.set_data(self.time_data, self.temp_data)
188     self.ax_temp_humidity.relim()
189     self.ax_temp_humidity.autoscale_view()
190     self.canvas_temp_humidity.draw()
191
192 def update_humidity(self, humidity):
193     self.humidity_data.append(humidity)
194     self.line_humidity.set_data(self.time_data, self.humidity_data)
195     self.ax_temp_humidity.relim()
196     self.ax_temp_humidity.autoscale_view()
197     self.canvas_temp_humidity.draw()
198
199 def update_motion(self, motion):
200     if motion == "Motion detected!":
201         # Set motion data to 1
202         self.motion_data.append(1)
203         # Update the graph with the new data
204         self.line_motion.set_data(range(len(self.motion_data)), self.motion_data)
205         self.ax_motion.relim()
206         self.ax_motion.autoscale_view()
207         self.canvas_motion.draw()
208     elif motion == "No motion detected.":
209         # Set motion data to 0
210         self.motion_data.append(0)
211         # Update the graph with the new data
212         self.line_motion.set_data(range(len(self.motion_data)), self.motion_data)
213         self.ax_motion.relim()
214         self.ax_motion.autoscale_view()
215         self.canvas_motion.draw()
216
217 class ControlPage(tk.Frame):
218     def __init__(self, parent, controller, mqtt_client):
219         tk.Frame.__init__(self, parent)
220         self.mqtt_client = mqtt_client
221
222         self.label = tk.Label(self, text="Control Page", font=("Helvetica", 16, "bold"), foreground="black")
223         self.label.pack(pady=10)
224
225         # Create Lamp control with a single button for on/off
226         self.lamp_label = tk.Label(self, text="LAMP", font=("Helvetica", 12, "bold"), foreground="black")
227         self.lamp_label.pack(pady=(20, 0))
228
229         self.lamp_state = tk.BooleanVar()
230         self.lamp_state.set(False) # Default state is OFF
231
232         # Create a frame to hold the buttons
233         button_frame = tk.Frame(self)
234         button_frame.pack(pady=5)
235
236         self.lamp_button_on = tk.Button(button_frame, text="Turn On", command=self.turn_on, width=10, height=2)
237         self.lamp_button_on.pack(side="left")
238
239         partition_label = tk.Label(button_frame, text=" | ", font=("Helvetica", 12, "bold"))
240         partition_label.pack(side="left")
241
242         self.lamp_button_off = tk.Button(button_frame, text="Turn Off", command=self.turn_off, width=10, height=2)
243         self.lamp_button_off.pack(side="left")
244
245         back_button = tk.Button(self, text="Back to Menu", command=lambda: controller.show_frame(MainMenuPage))
246         back_button.pack(pady=20)
247
248         # Subscribe to the Lamp topic
249         self.mqtt_client.subscribe("/public/iqram/lamp")
250         self.mqtt_client.message_callback_add("/public/iqram/lamp", self.update_lamp_state)
251
252 def update_lamp_state(self, client, userdata, message):
253     payload = message.payload.decode("utf-8")
254     if payload == "ON":
255         self.lamp_state.set(True)
256         self.lamp_button_on.config(bg="green") # Set button color to green when lamp is ON
257         self.lamp_button_off.config(bg="white")
258         print("Lamp is ON")
259     elif payload == "OFF":
260         self.lamp_state.set(False)
261         self.lamp_button_on.config(bg="white")
262         self.lamp_button_off.config(bg="red") # Set button color to red when lamp is OFF
263         print("Lamp is OFF")
264
265 def turn_on(self):
266     self.mqtt_client.publish("/public/iqram/lamp", "ON")
267
268 def turn_off(self):
269     self.mqtt_client.publish("/public/iqram/lamp", "OFF")
270
271 if __name__ == "__main__":
272     # MQTT client setup
273     mqtt_client = mqtt.Client()
274     mqtt_client.username_pw_set(username="public", password="public@")
275     mqtt_client.connect("115.187.22.64", 1883)
276     mqtt_client.loop_start()
277
278     # Create a tkinter window
279     root = tk.Tk()
280     root.title("Control Page")
281     root.geometry("600x400")
282
283     # Create an instance of ControlPage
284     control_page = ControlPage(root, None, mqtt_client)
285     control_page.pack(fill="both", expand=True)
286
287     # Run the tkinter main loop
288     root.mainloop()

```



```

291 class AutoPage(tk.Frame):
292     def __init__(self, parent, controller):
293         tk.Frame.__init__(self, parent)
294
295         self.controller = controller # Save reference to controller
296
297         label = tk.Label(self, text="Auto Page", font=("Helvetica", 16, "bold"), foreground="black")
298         label.pack(pady=10, padx=10)
299
300         back_button = tk.Button(self, text="Back to Menu", command=lambda: controller.show_frame(AutoPage))
301         back_button.pack()
302
303         auto_setup_label = tk.Label(self, text="Auto Setup", font=("Helvetica", 17, "bold"), foreground="black")
304         auto_setup_label.pack(pady=(20, 0))
305
306         # Search box for motion detection
307         motion_options = [
308             "Select Motion",
309             "Motion Detected: Turn On Lamp",
310             "Motion Not Detected: Turn Off Lamp"
311         ]
312
313         self.motion_search_var = tk.StringVar()
314         motion_combobox = tk.Combobox(self, textvariable=self.motion_search_var, values=motion_options, state="readonly", width=40)
315         motion_combobox.set(motion_options[0])
316         motion_combobox.pack(pady=5)
317
318         setup_button = tk.Button(self, text="Set Up Auto Mode", command=self.setup_auto_mode)
319         setup_button.pack(pady=10)
320
321         # Output label to display selected options
322         self.output_label = tk.Label(self, text="", font=("Helvetica", 0), foreground="black")
323         self.output_label.pack(pady=10)
324
325         def setup_auto_mode(self):
326             motion_selected_option = self.motion_search_var.get()
327
328             # Display the selected options in the output label
329             output_text = f"Motion Scenario: {motion_selected_option}"
330             self.output_label.config(text=output_text)
331
332             # Publish the selected options to the MQTT broker
333             message = f"Motion Scenario: {motion_selected_option}"
334             self.controller.client.publish("auto_mode_setup", message)
335
336             # Add your logic to interpret the selected option and perform specific actions
337             if motion_selected_option == "Motion Detected: Turn On Lamp":
338                 print("Scenario: Motion detected. Turning on Lamp.")
339                 # Add your logic here
340
341             elif motion_selected_option == "Motion Not Detected: Turn Off Lamp":
342                 print("Scenario: Motion not detected. Turning off Lamp.")
343                 # Add your logic here
344
345             else:
346                 print("Invalid option selected.")
347                 # Publish the "Invalid option selected." message to the MQTT broker
348                 self.controller.client.publish("auto_mode_output", "Invalid option selected.")
349
350 # Instantiate the SmartSensorApp class
351 app = SmartSensorApp()
352
353 # Start the Tkinter event loop
354 app.mainloop()
355
356 class ScrollingLabel(tk.Label):
357     def __init__(self, master=None, **kwargs):
358         tk.Label.__init__(self, master, **kwargs)
359         self.text = kwargs.get("text", "")
360         self.after(50, self.update_text)
361
362     def update_text(self):
363         self.text = self.text[1:] + self.text[0]
364         self.config(text=self.text)
365         self.after(50, self.update_text)
366
367 if __name__ == "__main__":
368     # Assuming you have the MainMenuPage class defined
369     pass
370
371 class SensorPage(tk.Frame):
372     def __init__(self, parent, controller):
373         tk.Frame.__init__(self, parent)
374         self.label = tk.Label(self, text="", font=("Helvetica", 16, "bold"), foreground="black")
375         self.label.pack(pady=10, padx=10)
376
377         back_button = tk.Button(self, text="Back to Menu", command=lambda: controller.show_frame(MainMenuPage))
378         back_button.pack()
379
380     def update_label(self, sensor):
381         self.label.config(text=f"Sensor Page: {sensor}")
382
383 class SwitchPage(tk.Frame):
384     def __init__(self, parent, controller):
385         tk.Frame.__init__(self, parent)
386         self.label = tk.Label(self, text="", font=("Helvetica", 16, "bold"), foreground="black")
387         self.label.pack(pady=10, padx=10)
388
389         back_button = tk.Button(self, text="Back to Menu", command=lambda: controller.show_frame(MainMenuPage))
390         back_button.pack()
391
392     def update_label(self, switch):
393         self.label.config(text=f"Switch Page: {switch}")
394
395 class MainMenuPage(tk.Frame):
396     def __init__(self, parent, controller):
397         tk.Frame.__init__(self, parent)
398
399         # Add buttons for Data, Control, and Auto options
400         data_button = ColorButton(self, text="Data", font=("Helvetica", 12, "bold"), command=lambda: controller.show_frame(OutPage))
401         data_button.pack(side="top", pady=(20, 0))
402
403         control_button = ColorButton(self, text="Control", font=("Helvetica", 12, "bold"), command=lambda: controller.show_frame(ControlPage))
404         control_button.pack(side="top", pady=20)
405
406         auto_button = ColorButton(self, text="Auto", font=("Helvetica", 12, "bold"), command=lambda: controller.show_frame(AutoPage))
407         auto_button.pack(side="top", pady=20)
408
409         sensors_label = tk.Label(self, text="There are 2 sensors:", font=("Helvetica", 16, "bold"), foreground="black")
410         sensors_label.pack(side="top", pady=(20, 0))
411
412         sensor1_label = tk.Label(self, text="Temperature & Humidity", font=("Helvetica", 10), foreground="black")
413         sensor1_label.pack(side="top", pady=5)
414
415         sensor2_label = tk.Label(self, text="PH Sensor", font=("Helvetica", 10), foreground="black")
416         sensor2_label.pack(side="top", pady=5)
417
418         switches_label = tk.Label(self, text="There are 1 switch:", font=("Helvetica", 16, "bold"), foreground="black")
419         switches_label.pack(side="top", pady=20)
420
421         switch1_label = tk.Label(self, text="(Lamp)", font=("Helvetica", 10), foreground="black")
422         switch1_label.pack(side="top", pady=5)
423
424         back_button = tk.Button(self, text="Back to Menu", command=lambda: self.back_to_menu(controller))
425         back_button.pack(side="top", pady=20)

```

```

400 class ColorButton(tk.Button):
401     def __init__(self, master=None, **kwargs):
402         tk.Button.__init__(self, master, **kwargs)
403         self.configure(foreground="white", background="white", activebackground="white", relief=tk.FLAT, pady=10)
404
405 class MainMenuPage(tk.Frame):
406     def __init__(self, parent, controller):
407         tk.Frame.__init__(self, parent)
408
409         # Add buttons for Data, Control, and Auto options
410         data_button = ColorButton(self, text="Data", font=("Helvetica", 12, "bold"), command=lambda: controller.show_frame(DataPage))
411         data_button.pack(side="top", pady=20, padx=0)
412
413         control_button = ColorButton(self, text="Control", font=("Helvetica", 12, "bold"), command=lambda: controller.show_frame(ControlPage))
414         control_button.pack(side="top", pady=20)
415
416         auto_button = ColorButton(self, text="Auto", font=("Helvetica", 12, "bold"), command=lambda: controller.show_frame(AutoPage))
417         auto_button.pack(side="top", pady=20)
418
419         sensors_label = tk.Label(self, text="There are 2 sensors:", font=("Helvetica", 10, "bold"), foreground="black")
420         sensor_label.pack(side="top", pady=20, padx=0)
421
422         sensor1_label = tk.Label(self, text="Temperature & Humidity", font=("Helvetica", 10), foreground="black")
423         sensor1_label.pack(side="top", pady=5)
424
425         sensor2_label = tk.Label(self, text="PIR Sensor", font=("Helvetica", 10), foreground="black")
426         sensor2_label.pack(side="top", pady=5)
427
428         switches_label = tk.Label(self, text="There are 2 switches:", font=("Helvetica", 10, "bold"), foreground="black")
429         switch_label.pack(side="top", pady=20)
430
431         switch1_label = tk.Label(self, text="Lamp", font=("Helvetica", 10), foreground="black")
432         switch1_label.pack(side="top", pady=5)
433
434         switch2_label = tk.Label(self, text="Fan", font=("Helvetica", 10), foreground="black")
435         switch2_label.pack(side="top", pady=5)
436
437         back_button = tk.Button(self, text="Back to Menu", command=lambda: self.back_to_menu(controller))
438         back_button.pack(side="top", pady=30)
439
440     def back_to_menu(self, controller):
441         # If content was displayed on the last click, undo it
442         if controller.last_displayed_content:
443             controller.show_frame(controller.last_displayed_content)
444         else:
445             controller.show_frame(MainMenuPage)
446
447
448 class ColorButton(tk.Button):
449     def __init__(self, master=None, **kwargs):
450         tk.Button.__init__(self, master, **kwargs)
451         self.default_color = self.cget("background")
452         self.bind("<enter>", self.on_enter)
453         self.bind("<leave>", self.on_leave)
454
455     def on_enter(self, event):
456         self.configure(background="lightblue")
457
458     def on_leave(self, event):
459         self.configure(background=self.default_color)
460
461 if __name__ == "__main__":
462     app = SmartSensorApp()
463     app.geometry("800x600")
464     app.mainloop()

```