Smart Sensors, Measurement and Instrumentation 41

Alice James
Avishkar Seth
Subhas Chandra Mukhopadhyay

IoT System Design

Project Based Approach

Springer

Smart Sensors, Measurement and Instrumentation

Volume 41

Series Editor

Subhas Chandra Mukhopadhyay, School of Engineering, Macquarie University, Sydney, NSW, Australia

More information about this series at http://www.springer.com/series/10617

Alice James · Avishkar Seth ·
Subhas Chandra Mukhopadhyay

IoT System Design

Project Based Approach

Springer

Alice James
School of Engineering
Macquarie University
Sydney, NSW, Australia

Avishkar Seth
School of Engineering
Macquarie University
Sydney, NSW, Australia

Subhas Chandra Mukhopadhyay
School of Engineering
Macquarie University
Sydney, NSW, Australia

ISSN 2194-8402 ISSN 2194-8410 (electronic)
Smart Sensors, Measurement and Instrumentation
ISBN 978-3-030-85862-9 ISBN 978-3-030-85863-6 (eBook)
https://doi.org/10.1007/978-3-030-85863-6

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Dedicated to

Our beloved parents

James Joseph

Gladys James

Late Nitin Vijay Seth

Zenobia Nitin Seth

Late Dyutindra Nath Mukhopadhyay

Rabi Rani Mukhopadhyay

Preface

We live in an age where Internet technology has enabled a global network in which people and devices are closer than ever before. The origin of the idea of the 'Internet' dates back to the early 1960s when scientists and researchers were trying to find a means to create a global network to share information. However, the Internet was truly enabled and commercialized with the creation of the World Wide Web in 1989. With its wide compatibility among different types of technologies, Internet technology soon became an available feature in most devices. The characteristic feature of Internet functionality that is enabled in these things led to the development of the term Internet of Things or 'IoT'. The 'thing' may be an entity, a physical item, or a device with the ability to transmit/receive data over a network. The hardware is integrated with sensors, firmware, and other features with the goal of communication between different devices and systems over the Internet.

The applications of IoT in today's world have increased as innovation pushes the paradigm for new technology. Consumer applications of IoT such as vehicular networks, smart health care, smart wearables, and home automation have grown in recent times. In the industrial sector, with the introduction of Industry 4.0, IoT applications have skyrocketed. It includes applications in transport, manufacturing, agriculture, and supply chain. Some of the new terms coined include Enterprise IoT (EIoT), Internet of Medical Things (IoMT), and Ocean of Things to name a few.

With the level of importance and wide application area, the need to research and design such systems is essential and challenging. For somebody who is just starting to learn about the concept of IoT, it can be a confusing subject. To add to the confusion, the number of resources one needs to study to get even a basic grasp on the topic is tremendous. With many of the resources only providing a theoretical perspective to these concepts, the practical understanding of a user could be lacking.

This book aims to provide a brief introduction to the core concepts of IoT with unique detail to the system design process. It provides a practical approach to designing and developing these IoT systems. The intended readers of this book include college and academics who are specialized in electronics engineering, telecommunication engineering, and information technology, data scientists,

researchers working with electronic systems, hobbyists, professionals, and anybody interested in a project-based approach to IoT.

The majority of the material from this book was used to teach undergraduate and postgraduate engineering students at Macquarie University, Sydney. The students performed the laboratory exercises in groups of 2–3 in a time frame of 3–4 h. The practicals provide a mix of guided exercise, theory, and self-performing tasks to enable a deeper understanding of the concepts. In the initial weeks, all the students perform a common task based on the week's guidelines. However, in the concluding weeks, the student groups perform an IoT group project. In this manner, the material in this book provides a sufficient introduction to building a practical IoT-based project for users.

The complete book is divided into twelve chapters. It is our view that expertise in designing IoT systems is achieved by using it in different applications. Many types of projects have also been detailed. With hardware projects, a special emphasis on software-based simulation projects has been shown for readers having minimal access to hardware components.

Chapter 1 aims to provide a brief overview of the IoT systems design. It explains the role IoT plays in the big picture. The chapter introduces the fundamentals of sensors, a fundamental element of IoT, and building blocks of IoT, and also describes the purpose of the book.

Chapter 2 describes a project implementation towards designing an IoT-based device. The chapter goes into great detail to explain the execution process of the IoT project, providing step-by-step guidelines along with source code for the user to perform it on the first go. A great skill that comes in handy when prototyping is mastering the ability to troubleshoot any problem. The topics also go into detail to mention the common debugging issues and challenges faced while performing the projects.

Chapter 3 provides essential inputs to the design perspective of building an IoT project. Designing an IoT node requires careful technical considerations that need to be fulfilled before the deployment of a device. The chapter looks into the individual elements that constitute an IoT node, from sensors to data analytics, providing design considerations and application-specific estimations. A key parameter to note when designing is making appropriate assumptions whenever necessary.

Chapter 4 introduces the single-board computer systems in depth. It provides a brief comparison between the commercially available computing boards that are suited for IoT-based systems. The chapter also provides the fundamental programming knowledge required to code these microprocessors. The Raspberry Pi, a popular single-board computer, is introduced with guidelines provided for the set-up and installations of software to make it an IoT device.

Chapter 5 introduces programming implemented on microcontrollers. The highly popular microcontroller board, Arduino, is used here. The Arduino boards have various types of different embedded microcontroller boards that enable simple to complex IoT projects. Coding them for IoT applications is extremely useful for users.

Chapters 6–8 feature the wireless transmission protocols used by IoT devices. These chapters provide hands-on exercises and tasks for users to learn and practically implement the concepts on the microcontroller/microprocessor boards. The top three protocols used by IoT devices, i.e. Wi-Fi, Bluetooth (BLE), and LoRaWAN technologies, are discussed here.

The topics and concepts of cloud computing in IoT-based projects are discussed in Chap. 9. This chapter provides the importance of the cloud in IoT systems. It also discusses some of the useful concepts of cloud development services like PaaS, IaaS, and SaaS.

The concepts of AI and machine learning are discussed in Chap. 10. Machine learning applications have seen tremendous growth in almost all sectors. The chapter discusses the links and connections between IoT and machine learning. It also shows how AI can be used in IoT-based projects. Further, key concepts of edge computing and data analysis are also explained briefly in this chapter.

Chapter 11 provides readers some key simulation software ideas. The chapter also shows projects performed on these IoT device simulation-based services for users who wish to get a hands-on experience but have low availability of hardware equipment.

Lastly, Chap. 12 showcases some of the projects performed by the students of Macquarie University. The projects provide the wide applications of IoT in today's life. The projects go in depth to explain the hardware, software, and programs written to execute the prototype of IoT devices.

The authors would like to express their sincere gratitude towards Macquarie University for providing the research laboratories and resources which made this book possible. We would like to thank many of the students and colleagues that were involved in the projects of this book.

Alice is very grateful to her parents Gladys and James for their immense love and support. A very special mention to her grandparents for their blessings, dear Aunts Sheeba and Salomy for their constant motivation, and her loving cousins Sheldon, Suzanne, Joshua, and Johann for always being there for her. Avishkar is truly thankful to his parents Zenobia and Nitin for their love and blessings, and a special mention to her grandparents and sister Karishma for their constant motivation.

Alice and Avishkar would like to acknowledge their families for their unequivocal support throughout and for which mere expression of thanks likewise does not suffice.

Sydney, Australia

Alice James
alice.james@mq.edu.au
Avishkar Seth
avishkar.seth@mq.edu.au
Subhas Chandra Mukhopadhyay
subhas.mukhopadyay@mq.edu.au

Contents

About the Authors

Miss Alice James received her B.E. in information technology from Mumbai University, India, in 2018 and master's degree in engineering in electronics from Macquarie University, Australia, in 2020. She is currently pursuing her master's degree in research in engineering in the domain of sensor design and medical technology at Macquarie University. She currently holds the position of Vice-Chair in IEEE Women in Engineering Macquarie Student Branch. Her research interests include flexible sensors, smart sensing systems, robotics, and machine learning.

Mr. Avishkar Seth received his B. Eng. degree in electronics from Mumbai University, India, in 2017, and the M. Eng. degree in electronics from Macquarie University, Sydney, Australia, in 2020. He is currently pursuing a Master of Research degree in AI-enabled wearable sensors for healthcare applications from Macquarie University, Sydney. His research interest includes smart sensing systems, IoT, AI, and robotics. He is Vice-Chair of IEEE Student Branch at Macquarie University.

Dr. Subhas Chandra Mukhopadhyay (M'97, SM'02, F'11) graduated from the Department of Electrical Engineering, Jadavpur University, Calcutta, India, with a Gold Medal and received the Master of Electrical Engineering degree from Indian Institute of Science, Bangalore, India. He has Ph.D. (Eng.) degree from Jadavpur University, India, and Doctor of Engineering degree from Kanazawa University, Japan.

Currently, he is working as Professor of Mechanical/Electronics Engineering and Discipline Leader of the Mechatronics Degree Programme of the School of Engineering, Macquarie University, Sydney, Australia. He has over 31 years of teaching and research experiences.

His fields of interest include smart sensors and sensing technology, wireless sensor networks, Internet of Things, electromagnetics, control engineering, magnetic bearing, fault current limiter, electrical machines and numerical field calculation, etc.

He has authored/co-authored over 450 papers in different international journals, conferences, and chapter. He has edited eighteen conference proceedings. He has also edited thirty-two special issues of international journals as lead guest editor and thirty-five books with Springer-Verlag.

He was awarded numerous awards throughout his career and attracted over AUD 6.2 M on different research projects.

He has delivered 374 seminars including keynote, tutorial, invited, and special seminars.

He is Fellow of IEEE (USA), Fellow of IET (UK), and Fellow of IETE (India). He is Topical Editor of IEEE Sensors Journal and Associate Editor of IEEE Transactions on Instrumentation. He has organized many international conferences as either General Chair or Technical Programme Chair. He is Founding Chair of the IEEE Sensors Council New South Wales Chapter.

Chapter 1
IoT System Design—The Big Picture

1.1 Introduction

Internet of Things of IoT has penetrated our society, our day-to-day life and it is not a buzzword anymore. Throughout our life, we come across different IoT-based applications which make our life more comfortable, enjoyable, and purposeful. A few years back it would have been a dream to think of something so wonderful will be in our life as is explained in Fig. 1.1. A simple toaster can be a self-aware smart intelligent IoT enabled system if it is configured with different sensors and internet connectivity. It can then send an appropriate message about the detailed conditions of the toast as well as when done via the internet. It is now possible due to IoT as the toaster is a thing that can communicate with anything, to anyone, located anywhere and at any time.

1.2 Sensors—Fundamental Element of IoT

To make a successful IoT system, sensors are essential and play the most important role as it provides all real-world information. A **sensor** is a device that measures a physical/chemical/biological quantity and converts it into a signal usually electrical form which can be read by an observer or by an instrument. The signal can be analyzed, processed, stored, and transmitted to any place either by a wired or wireless medium. Usually, it is assumed that an ideal sensor is only sensitive to the measured property, needs to be insensitive to any other property and it should not influence the measured property. Sensors are essential elements for any system especially if the performance of the system is aimed to be improved, it is important to know the current situation. The current measured parameters are obtained from sensors which are base values to determine whether improvement is required or not. Figure 1.2 shows some of the different sensors used for our daily life.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_1

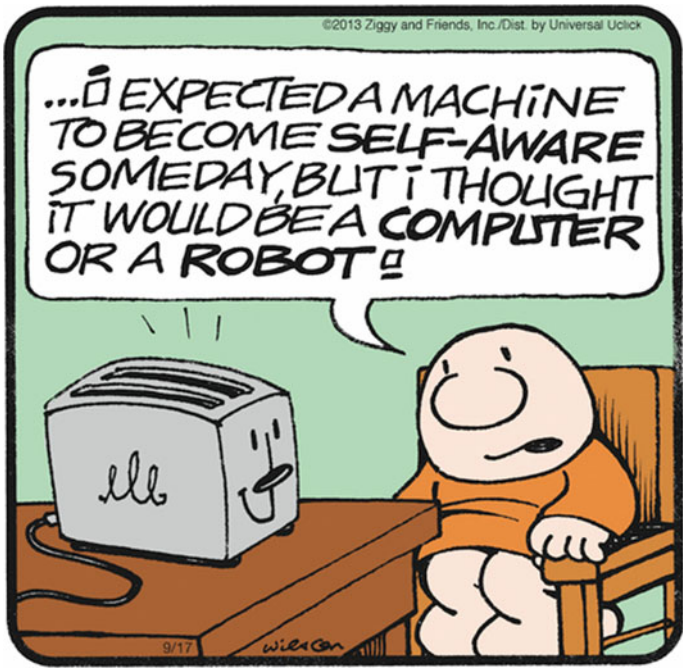


Fig. 1.1 IoT as a dream a few years back

Sensors provide outputs in a variety of different ways, either analog or digital, and then the outputs may be in wide forms. For example, the analog signal output from a sensor can be either in voltage or in current form, can be in direct current form, or an alternating current form. The voltage can be in milliVolt or Volts. It is required to transform the output signal into a standard value between 0 to 3.3 V before the signal is interfaced with an embedded processor.

The sensors used for any IoT system depend on the specific application and many parameters are considered along with their performance figures when chosen. Availability of sensors such as the location of sources, delivery schedule, payment options, a brand name of the product, a continuation of supply and the cost, cost of sensor itself, delivery cost are important.

1.3 Building Blocks of IoT System

IoT is a network of items—each embedded with sensors—which are connected to the Internet. The fundamental building blocks for making an IoT system are shown in Fig. 1.3. Sensors are the fundamental element but the raw data from the sensors need some kind of processing. So the embedded processor is another important

Fig. 1.2 Sensors used in our day-to-day life

Fig. 1.3 Building blocks of an IoT system

block in the IoT system. To upload measured data to the cloud, internet connectivity is essential. The wireless transceiver (WiFi, Zigbee, LoRa, GSM, etc.) is to be properly chosen depending on the application. Since the data is available in the cloud, security plays an important role in IoT systems. The last big thing is the data analysis, data visualization, and all software-related applications where machine learning and artificial intelligence have their role to play.

1.4 IoT Data and Information Processing

IoT system generates a huge amount of real-time data due to connected sensors in IoT nodes. All data are collected and stored on the server. Data carries information that is to be extracted. While information is extracted it is very important to use knowledge as data can be misinterpreted in different ways. So with wisdom and proper knowledge, extraction of real information from sensor data is very useful and desirable, the whole picture is represented in Fig. 1.4.

1.5 Challenges of Design of IoT System

In recent times there are many IoT systems reported and are available around us. Though many new applications are coming up there are many research challenges which are listed below.

- Availability of internet at everywhere at all time
- Acceptability among the society and individual
- Low-cost smart sensor node development
- Energy harvesting
- Computational ability
- Security issues
- Scalability, Reliability
- Fault Tolerance
- Power Consumption of nodes and transceiver

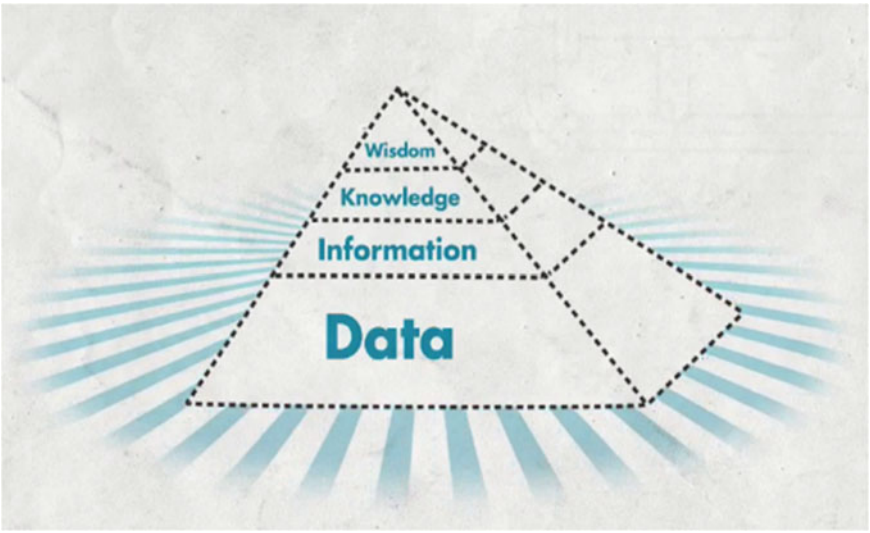


Fig. 1.4 IoT Data and Information processing

Another important issue to be considered is to dispose of the sensors and electronics after their lifetimes. It is expected trillions of sensors will be available due to IoT nodes within a few years. The nodes will need to be re-used in some way as disposing of them will pollute the environment electronically and is known as e-Waste. The designers and engineers are aware of the problems and a lot of efforts are now being put into the design of environmentally friendly nodes. Engineers need to think of designing nodes that are non-hazardous and bio-degradable.

1.6 Purpose of the Book

The book is intended to provide a step-by-step approach to engineers, designers especially higher undergraduate, and postgraduate students to design IoT systems. Though there are many books and numerous research papers are available in the public domain, there is a lack of a simple approach towards the design of IoT systems with the emphasis on the big picture. Starting from sensors interfacing, embedded processing, wireless communication, cloud computing, and data analysis will be presented in hands-on mode. The plan for the complete topis is presented in Table 1.1.

Table 1.1 IoT Systems Design- Session Plan

Week No	Activity	Comment
#1	Introduction to an IoT system	In this week, students will be given a developed IoT system in 4 stages to implement Stage 1: Sensor interface Stage 2: Serial monitor results Stage 3: configuration of the transmission module Stage 4: Transmission of data to cloud and visualization
#2	Introduction to Microcontroller/ microprocessor and filter design	In this week, students will be introduced to Arduino and Raspberry pi configuration They will learn the basic setup and how to program it Along with this, they will develop a Bandpass/ Bandstop filter to understand and eliminate any noise
#3	Sensor Interfacing and introduction to cloud computing	In this week, students will learn sensor interfacing and creating their own library/header files, etc. Along with this, students will learn the basics of cloud computing and the transmission of data in an IoT system

(continued)

Table 1.1 (continued)

Week No	Activity	Comment
		For instance: understanding cloud computing services available (PaaS, IaaS, SaaS, etc.) Understanding what is an API, how does one assign a payload, etc.
#4	Designing user interface	In this week, students will continue the interfacing from the previous week and further complete the configuration and setup of an API, format payload, transmit data They will design an interface to visualize the transmitted data
#5	Modules of wireless communication—Bluetooth	Along with this, students will be introduced to the first module of wireless communication i.e. Bluetooth. In this workshop, they will learn how to configure the Bluetooth module Interfacing with microcontroller (Arduino) and transmission of data Understand the limitations of the module Implementation of Bluetooth in the microprocessor (Raspberry pi)
#6	Modules of wireless communication—WiFi	In this week, students will be introduced to the first module of wireless communication i.e. WiFi In this workshop, they will learn how to configure the WiFi module Interfacing with microcontroller (Arduino) and transmission of data Understand the limitations of the module Implementation of WiFi in the microprocessor (Raspberry pi)
#7	Modules of wireless communication—LoRa	In this week, students will have their third demonstration Along with this, students will be introduced to the first module of wireless communication i.e. LoRa In this workshop, they will learn how to configure MKR1300 Interfacing with sensor, inter-device communication and transmission of data Understand the limitations of the module
#8–#10	Project work	A project can be implemented during the 3 sessions

Suggested Reading

1. S.C. Mukhopadhyay, Intelligent sensing, instrumentation and measurements. Smart Sens. Measur. Instrum. **5** (2013). ISBN: 978-3-642-37026-7 (Print); 978-3-642-37027-4 (Online), April 2013
2. S. Nagender Kumar, S. C. Mukhopadhyay, Smart homes: design, implementation and issues. Smart Sens. Meas. Instrum. **14** (2015). 978-3-319-13556-4
3. H. Ghayvat, S.C. Mukhopadhyay, Wellness protocol for smart homes: an integrated framework for ambient assisted living. Smart Sens. Meas. Instrum. **24** (2016). ISBN: 978-3-319-52047-6 (Print); 978-3-319-52048-3 (Online)
4. E.A. Md Eshrat, S.C. Mukhopadhyay, Smart nitrate sensors, internet of things enabled real time water quality monitoring. Smart Sens. Meas. Instrum. **35** (2019). ISBN 978-3-030-20094-7
5. S.C. Mukhopadhyay, H. Leung (eds.), *Lecture Notes in Electrical Engineering, Advances in Wireless Sensors and Sensors Networks* (Springer, 2010). ISSN 1876-1100, ISBN 978-3-642-12706-9
6. S.C. Mukhopadhyay, J.A. Jiang (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 3, Wireless Sensor Networks and Ecological Monitoring* (Springer, 2013). ISBN 978-3-642-36364-1
7. S.C. Mukhopadhyay (ed.), *Smart Sensors, Measurement and Instrumentation, Vol. 9, Internet of Things: Challenges and Opportunities* (Springer, 2014). ISBN 978-3-319-04222-0
8. O.A. Postolache, E. Sazonov, S.C. Mukhopadhyay (eds.), *Sensors in the Age of the Internet of Things: Technologies and Applications* (IET Book, 2019). https://digital-library.theiet.org/content/books/ce/pbce122e. ISBN: 9781785616341, e-ISBN: 9781785616358
9. C. Wang, M. Daneshmand, M. Dohler, X. Mao, S.C. Mukhopadhyay, R. Qingyang Hu, H. Wang (eds.), *IEEE Sensors Journal, Special issue on Internet of Things: Architecture, Protocols and Services*, vol. 13(10) (2013). ISSN: 1530-437X
10. R. Morello, S. Mukhopadhyay, E. Gaura, Z. Li, D. Slomovitz, S.R. Samantaray (eds.), IEEE Sensors Journal, Special issue on Smart Sensors for Smart Grids and Smart Cities, IEEE Sens. J. **17**(23) (2017). ISSN: 1530-437X
11. R. Shankaran, S.C. Mukhopadhyay (eds.), Security in IoT Enabled Sensors', A special issue of Sensors (2018). ISSN: 1424-8220
12. F. Akhter, S. Khadivizand, H. Reza Siddiquei, Md E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
13. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using graphite/PDMS sensors. Sens. Actuators, A **286**, 43–50 (2019)
14. Md. E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417 (2018)
15. N. Afsarimanesh, Md. E.E. Alahi, S.C. Mukhopadhyay, M. Kruger, Development of IoT-based impedometric biosensor for point-of-care monitoring of bone loss. IEEE J. Emerg. Sel. Topics Circuits Syst. **8**(2), 211–220 (2018)
16. T. Islam, S.C. Mukhopadhyay, N.K. Suryadevara, Smart sensors and internet of things: a postgraduate paper. IEEE Sens. J. **17**(3), 577–584 (2017)
17. H. Ghayvat, J. Liu, M.E.E. Alahi, S.C. Mukhopadhyay, X. Gui, Internet of things for smart homes and buildings: opportunities and challenges. Aus. J. Telecommun. Digital Econ. **3**(4), 33–47 (2015). https://doi.org/10.18080/ajtde.v3n2.23. ISSN: 2203-1693
18. H. Ghayvat, J. Liu, S.C. Mukhopadhyay, X. Gui, Wellness sensor networks: a proposal and implementation for smart home for assisted living. IEEE Sens. J. **15**(12), 7341–7348 (2015)
19. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350

20. N.K. Suryadevara, S.C. Mukhopadhyay, S.D.T. Kelly, S.P.S. Gill, WSN-based smart sensors and actuator for power management in intelligent buildings. IEEE Trans. Mechatron. **20**(2), 564–571 (2015). https://doi.org/10.1109/TMECH.2014.2301716
21. N.K. Suryadevara, S.C. Mukhopadhyay, Internet of things: challenges and opportunities. Eur. Bus. Rev. 22–24 (2014). http://www.europeanbusinessreview.com/?p=5049
22. N.K. Suryadevara, S.C. Mukhopadhyay, Determining wellness through an ambient assisted living environment. IEEE Intell. Syst. 30–37 (2014)
23. N.K. Suryadevara, S.C. Mukhopadhyay, Internet of things: a review and future perspective. Eur. Bus. Rev. 18–20 (2014). http://www.europeanbusinessreview.com/?p=4431
24. N.K. Suryadevara, S.C. Mukhopadhyay, R. Wang, R.K. Rayudu, Forecasting the behavior of an elderly using wireless sensors data in a smart home. Eng. Appl. Artif. Intell. **26**(10), 2641–2652 (2013). https://doi.org/10.1016/j.engappai.2013.08.004. ISSN: 0952-1976
25. S.D. Tebje Kelly, N.K. Suryadevara, S.C. Mukhopadhyay, Towards the implementation of IoT for environmental condition monitoring in homes. IEEE Sens. J. **13**(10), 3846–3853 (2013)
26. N.K. Suryadevara, S.C. Mukhopadhyay, Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sens. J. **12**(6), 1965–1972 (2012)
27. N.K. Suryadevara, A. Gaddam, R.K. Rayudu, S.C. Mukhopadhyay, Wireless sensors network based safe home to care elderly people: behaviour detection. Sens. Actuators A: Phys. **186**, 277–283 (2012). https://doi.org/10.1016/j.sna.2012.03.020
28. R. Singh, A. Gehlot, L.R. Gupta, B. Singh, M. Swain, *Internet of Things with Raspberry PI and Arduino* (CRC Press, Taylor & Francis Group, 2020)
29. S.S. Iyengar, N. Parameshwaran, V.V. Phoha, N. Balakrishnan, C.D. Okoye, *Fundamentals of Sensor Network Programming* (Wiley, IEEE Press, 2011)
30. N. Mukherjee, S. Neogy, S. Roy, *Building Wireless Sensor Networks, Theoretical and Practical Perspective* (CRC Press, Taylor & Francis Group, 2016)
31. V. Sharma, A. Pughat (eds.), *Energy Efficient Wireless Sensors Networks* (CRC Press, Taylor & Francis Group, 2018). ISBN 978-1-4987-8334-7

Chapter 2
IoT System Design—A Project Based Approach

2.1 Introduction

Internet of Things, or IoT, means physical devices connected via the internet transmitting data with little to no human intervention. The concept of IoT dates back to 1982 when a smart vending machine was developed using an existing Cola machine. Although the idea of connected devices was realised early, the technology was not ready to adopt such devices in daily usage due to communication and chip size constraints. Soon, with the advancement of technology, the number of connected devices exceeded the number of people and IoT became a commonly used term for people.

This chapter introduces and discusses an IoT system focussing on building a complete IoT prototype device. It is aimed at students, hobbyists or professionals looking to enter into the world of electronics and would like to understand the basic working of these systems. A project experience-based learning approach was taken in this chapter. The main focus of this chapter is to describe in detail the steps and instructions to build the project with a short period in a laboratory. The major part of the project was programming the microcontroller and sending data to the cloud for analysis via LoRaWAN network. The project enables the users to see the results on the serial monitor of Arduino IDE as well as on the web via The Things Network and ThingSpeak websites. The group project allows users to communicate and share their ideas and knowledge, thus, enabling the users to attain a high level of technical experience and interpersonal skills.

The knowledge of microcontrollers and sensors is a very important skill for any engineer. Engineering students from any discipline at any University can learn the basic technique towards IoT Systems Design. The researchers are expected to complete specific tasks within a few hours. Overall, this project will allow to understand the fundamentals of an IoT system design.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_2

2.2 Description

The project description given to the users is elaborated in this section.

2.2.1 Objectives

The major aim of this project is to give the users a first-hand experience of building a complete IoT system. The following points describe the objectives in detail:

- Understanding the different components involved in an IoT system.
- Developing a complete working IoT system in a short amount of time of about 3–4 h.
- Experimenting with sensors and realising their function.
- Give users an idea of the microcontroller hardware functionality in building the system.
- Learning about the LoRa transmission protocol used.
- Experimenting with data bits and payload formats.
- Sending the data wirelessly to the cloud for visualisation and analytics.

2.2.2 Hardware Used in the Project

The hardware components used in this project mainly involve the sensors and microcontroller unit. The following components are required to build this system:

- Microcontroller: Arduino MKR WAN 1300 and Antenna
- Micro USB cable: Arduino MKR WAN 1300 compatible
- Sensors:
  - PIR Sensor Module-Motion Sensing
  - HC-SR04 Ultrasonic Module-Distance Measuring Sensor
  - AM2302 DHT22 Temperature and Humidity Sensor Module-Measuring ambient temperature and humidity
- Prototyping Solderless Breadboard
- Jumper Cables.

2.2.3 Software/Applications Used in the Project

The following are the software/applications used in this project:

- Microcontroller Programming: Arduino IDE
- Web Application for LoRaWAN network connectivity: The Things Network (TTN)
- Could data analytics and visualisation: ThingSpeak

2.3 Motivation

The motivation of this project came from the requirement to teach engineering students a quick and guided way to build a complete IoT system. Many of the courses focus solely on a theoretical perspective of the subject to teach the students. While this improves the users understanding and capabilities to comprehend the topic, it does not allow the user to experience building them on their own.

A project-based learning approach allows users to get practical exposure, learn debugging and find innovative solutions to solving complex problems. The group-based activities encourage interpersonal skills and enhance the learning experience that comes from sharing of ideas.

The overview of the various stages of the development of the IoT system that has been implemented in a four-hour lectorial (Lecture + Practical) are given in this section. The project guidelines provide in-depth codes and steps to be able to perform the project in one go.

2.4 Overview of Project Implementation

The section elaborates on the four main stages given as.

2.4.1 Microcontroller and Sensor Interface

Every IoT system is enabled with various sensors for measurement of parameters of interest. Interfacing sensors to microcontroller and obtaining data make it very useful to prototype IoT systems.

Sensors are used to measure physical parameters such as temperature, humidity, moisture etc. and provide electrical signals to communicate with the interfaced microcontroller. In this project the MKR1300 Arduino board is used that permits LoRa connectivity. The main aim to utilize this device to enable a low power Smart Node for an IoT system. Figure 2.1 shows the MKR1300 along with the 2 dB antenna that ranges for 433/868/915 MHz carrier frequencies.

The Smart sensor node consists of different sensors that generate digital data. The primary goal is to build a device that can translate into different applications.

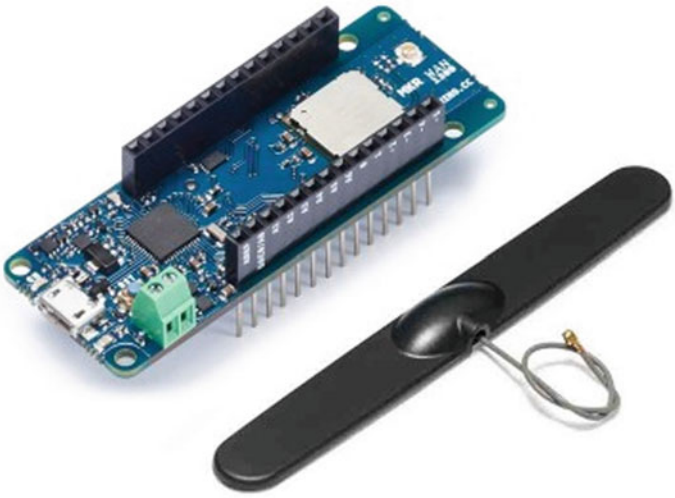


Fig. 2.1 The MKR1300 Arduino board that enables LoRa transmission

For this project the sensor used are an infrared sensor i.e. PIR, an ambient temperature and humidity monitoring sensor i.e. DHT22/AM2302 and a distance sensor i.e. Ultrasonic HC-SR04 as seen in Fig. 2.2.

To understand and complete the sensor interface to the microcontroller, the following steps are followed.

2.4.1.1 Arduino IDE Installations

The Arduino IDE 1.8.13 version has been used in this project. The IDE platform is always updated to the latest versions compatible with the new boards. The important aspect in the IDE software to realise is the availabilities of the various board support packages.

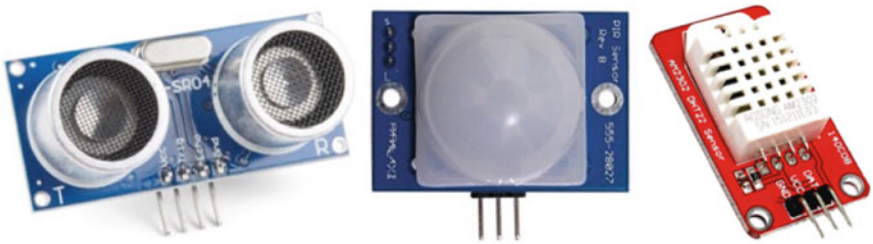


Fig. 2.2 (Left to right) HC-SR04 Ultrasonic sensor, Parallax PIR sensor, DHT22 temperature humidity sensor

For instance, to use the MKR1300 board, the 'Arduino SAMD (32-bits ARM Cortex-M0+) Board' package is installed. This can be checked on the 'Arduino IDE > Tools > Board:', as seen in Fig. 2.3.

For the implementation of the DHT sensor, the Arduino will directly read the float values of the temperature and humidity. The library for DHT sensor can be found in this GitHub repository (https://github.com/AliceJames-1/IoT_Systems_Design_Book).

To add a custom zip library into the Arduino IDE, the user needs to go to Arduino IDE > Sketch > Include Library > Add.ZIP library and then select the necessary files to upload and include as shown in Fig. 2.4.

The same method will be used to implement further in the project stage is the payload format, as this project uses the custom 'Cayenne LPP' format library which is also found in this GitHub repository (https://github.com/AliceJames-1/IoT_Systems_Design_Book).

2.4.1.2 Circuit Diagrams

In this section, the circuit diagram of the system developed is provided. The sensors are interfaced to the microcontroller using the digital pins to enable sensor readings.

The pin configurations for the sensors are as follows:

1. PIR sensor:

   VCC > VCC
   GND > GND
   OUT > Digital pin 6

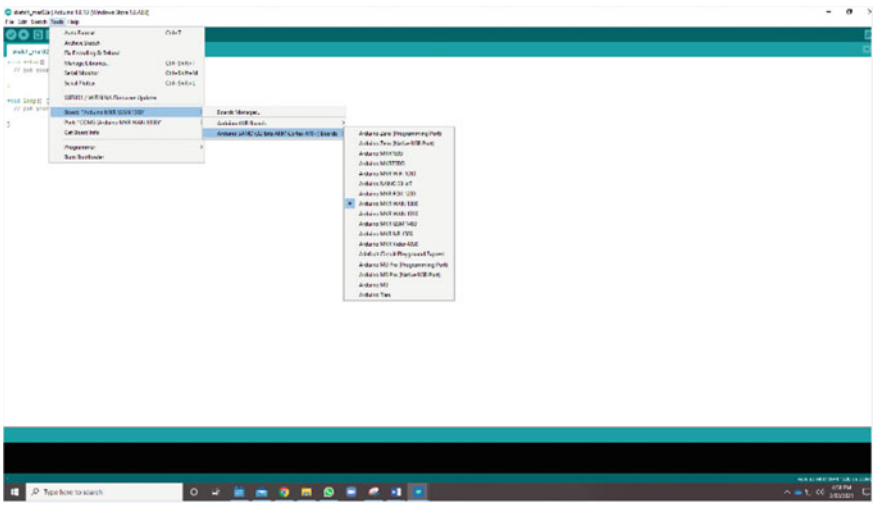


Fig. 2.3 The Arduino IDE Board package configurations

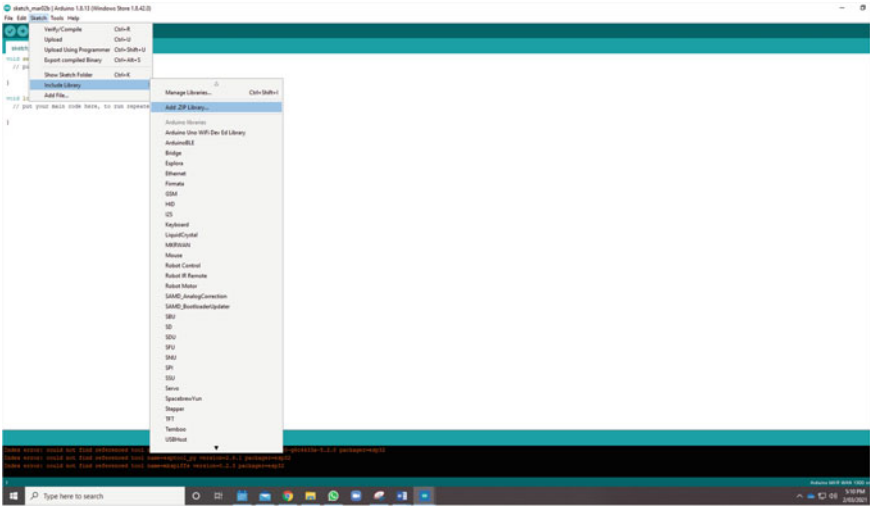


Fig. 2.4 The Arduino IDE library manager that allows addition of custom library

2. DHT22 sensor:

   VCC > VCC
   GND > GND
   DATA > Digital pin 7

3. Ultrasonic sensor:

   VCC > VCC
   GND > GND
   ECHO > Digital pin 2
   TRIG > Digital pin 3

The circuit diagram using these pin configurations can be seen in Fig. 2.5. The board used is the MKR1300.

The circuit assembled for the project is illustrated in Fig. 2.6, the red wires represent VCC, the blue represents the GND and the yellow represents the data cable.

2.4.1.3 Sensor Reading and Code Syntax

In this section, the specific code syntax to be included for the sensor execution. There is no ADC (Analog to Digital converter) used in this project as all the outputs are digital.

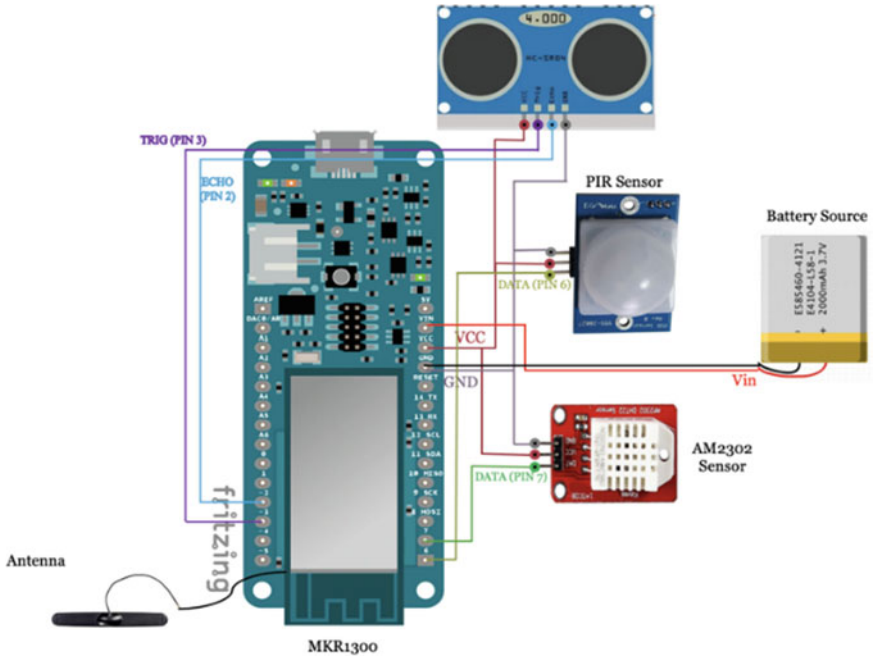


Fig. 2.5 The circuit diagram designed using Fritzing to display the pin configuration

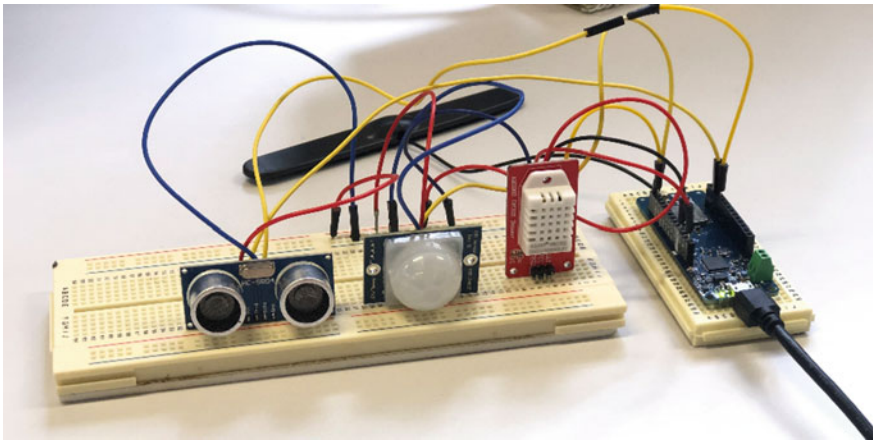


Fig. 2.6 The circuit assembled for the smart node using MKR1300

1. PIR sensor

The syntax to read from the PIR sensor is given as shown below, the primary aim is to assign an integer value to the PIR state to interpret the data transmitted easily.

The readings of the sensor are assigned to the 'pirState' variable.

```
void PiRSensor () {

 val = digitalRead(PIR);  // read input value

  if (val == HIGH) {
    if (pirState == LOW) {
      // the PIR is turned on
      pirState = 1;
      Serial.println(pirState);
    }
  } else {
    if (pirState == HIGH) {
      // the PIR is turned off
      pirState = 0;
      Serial.println(pirState);
  }
}
}
```

2. DHT22 sensor

The syntax to read from the DHT sensor is given as shown below, the primary aim is to assign an integer value to the DHT data pin out to interpret the data transmitted easily.

The code uses the default readings available in the 'DHT Tester' code provided in the library. The library provided in the GitHub repository provide the necessary header files to configure the sensor and enable and distinguish the data incoming from the data pin.

```
// Read humidity as percentage (the default)
float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();

   // Check if any reads failed and exit early (to try
again).
  if (isnan(h) || isnan(t)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
//print the values of the results
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
```

3. Ultrasonic sensor

The syntax to read from the Ultrasonic sensor is given as shown below, the primary aim is to assign an integer value to the Ultrasonic echo and trigger pin out to interpret the data transmitted easily.

The code initialises the 'Trigger' pin and then reads from the 'Echo' pin to further calculate the distance with respect to the speed of sound.

```
void Ultrasonic(){
 // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time
in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave
divided by 2 (go and back)
  // Displays the distance on the Serial Monitor
  Serial.println("Distance: ");
  Serial.print(distance);
  Serial.println(F(" cm"));}
```

2.4.2 Serial Monitor Results

The results for the sensor readings based on the syntax provided earlier are similar to the results shown in Fig. 2.8.

To view the serial monitor, the user may go to the Arduino IDE > Tools > Serial Monitor as shown in Figs. 2.7 and 2.8.

2.4.3 Configuration of the Transmission Module

This section elaborates the implementation of the LoRa transmission in the MKR1300. The transmission and gateway platform used in this project is the 'The Things Network' (TTN). The implementation stages are given below in sequence.

Fig. 2.7 The figure illustrates navigation to serial monitor in Arduino IDE

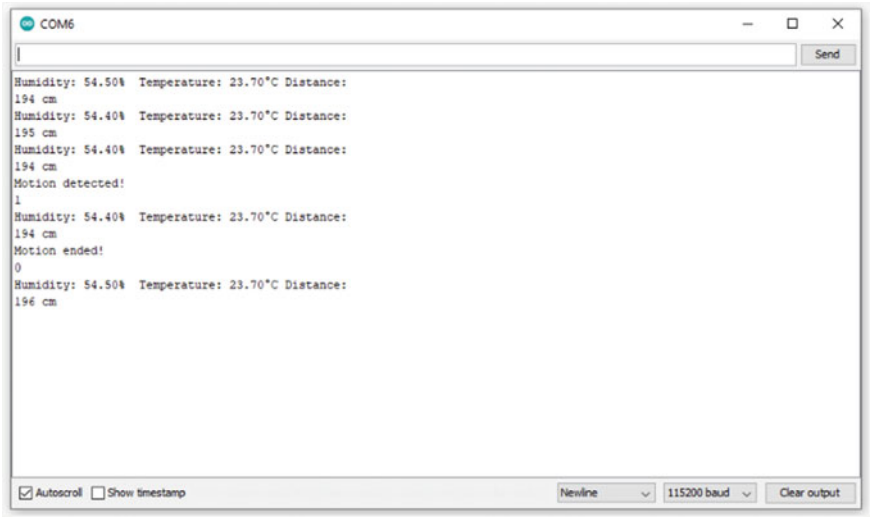


Fig. 2.8 The serial monitor snippet for all the three sensor readings using the Arduino IDE

2.4.3.1 Device Configuration and ID

The first stage is Device Configuration and ID, as the platform TTN registration is set to use OTAA in the project. To register the device with TTN the user requires the DeviceEUI, that is unique to user device.

To get this, first to download the 'MKRWAN' library from the Arduino IDE. Referring to Fig. 2.4, the steps are: Arduino IDE > Sketch > Include Library > Manage Library and type 'MKRWAN' as shown in Fig. 2.9 and install it.

Once, installed, the user may now use the 'mkrwan_01_get_deveui' file from the GitHub repository (https://github.com/AliceJames-1/IoT_Systems_Design_Book) and upload the code on the MKR1300 device. The result may be like Fig. 2.10. Save this result to use further in the registration process.

2.4.3.2 Setting up TTN

In this stage, the user first need to create an account with 'The Things Network' (TTN). Once created, the user may click on console given in the top-right corner of the screen and click on applications.

To add a new application, the user need to verify the locations of the gateways in the respective area and select the handler accordingly. In this project, the gateway handler used is: ttn-handler-asia-se.

The application registration form is shown in Fig. 2.11. The application ID can be unique to different nodes, one TTN_account holder may have many applications and 1 TTN_Application may have multiple devices registered. Once an application is registered, the user will see a dashboard similar to Fig. 2.12.

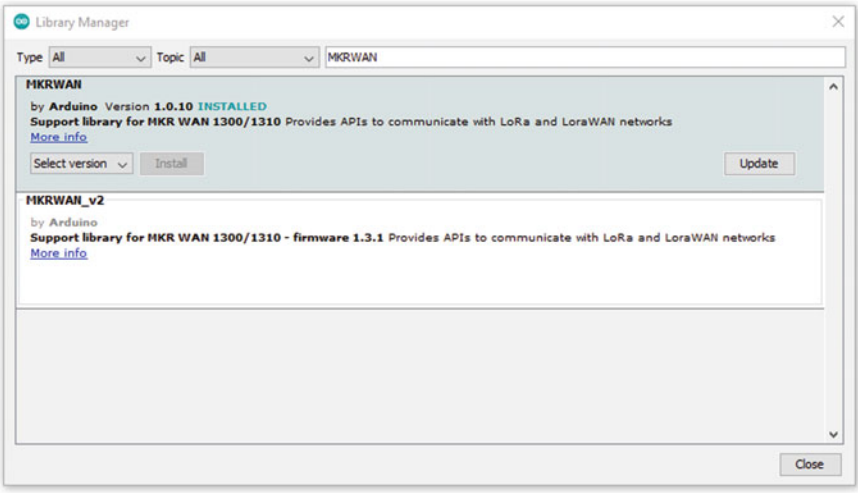


Fig. 2.9 The default library manager in the Arduino IDE, to download the predefined libraries

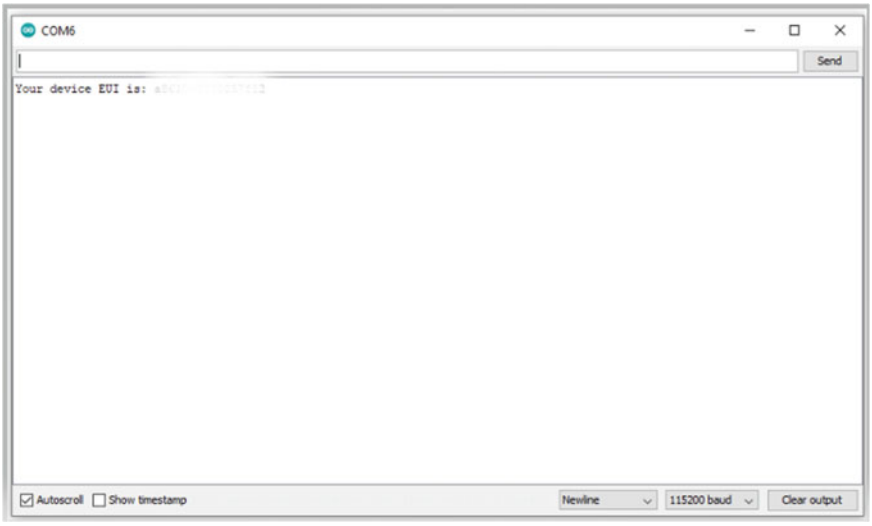


Fig. 2.10 The serial monitor's results for the DeviceEUI (this value is unique for all devices)

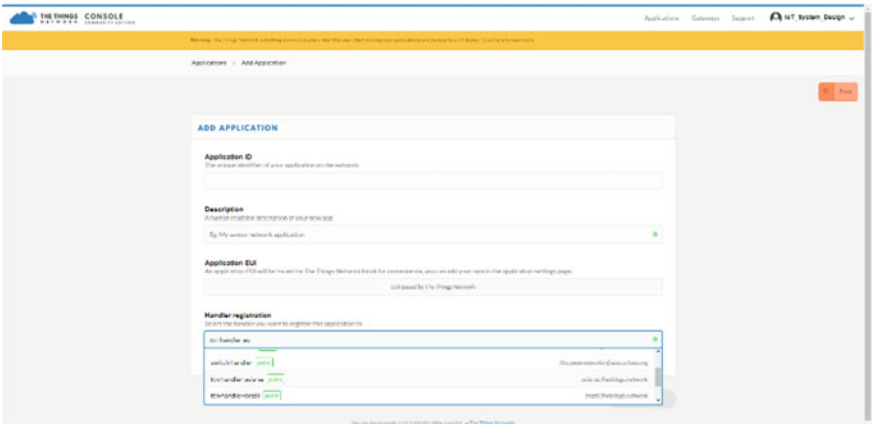


Fig. 2.11 The screenshot of the TTN dashboard for application registration

2.4.3.3 Registration of the Device

Once the user has created the application, they can now register a device using the DeviceEUI acquired earlier.

Within the application overview, scroll down, and click on register a device. The user may provide a unique name to the device and provide the details of the DeviceEUI and click register. Figure 2.13, shows a registered device overview.

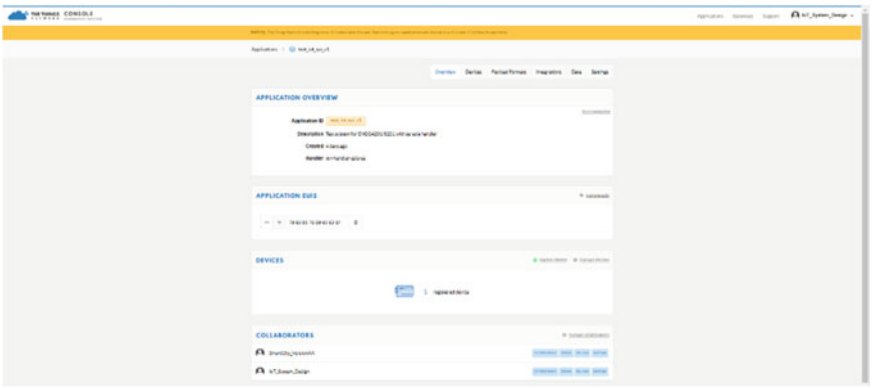


Fig. 2.12 The screenshot of the TTN dashboard showing the overview of the application created

2.4.3.4 First Communication

To verify the registration of the device, the user may now run the mkr-wan_02_hello_world code provided from the GitHub repository. The TTN device overview to check the status of the device as previously seen in Fig. 2.13. The user may validate the connection via the serial monitor of the IDE as seen in Fig. 2.14.

2.4.3.5 Payload Assignment

The payload format used in this project is the Cayenne LPP payload format. The user may download the library from the given GitHub repository and include in the Arduino IDE. The format to assign the payload to the data varies for different sensor

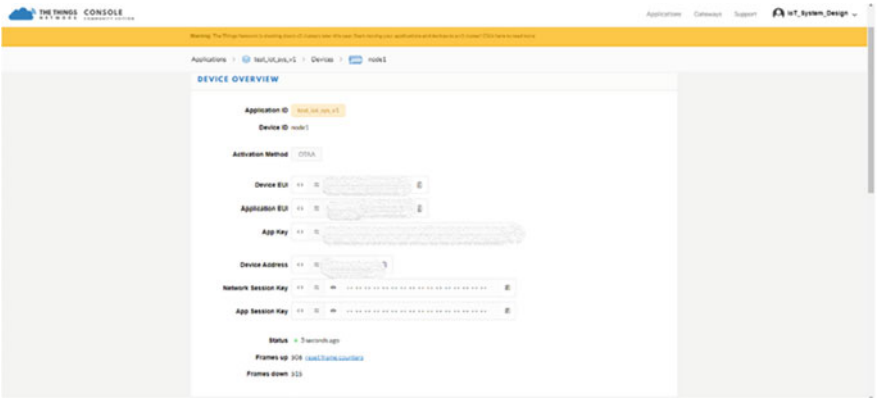


Fig. 2.13 The screenshot of the TTN dashboard that shows the registered device overview

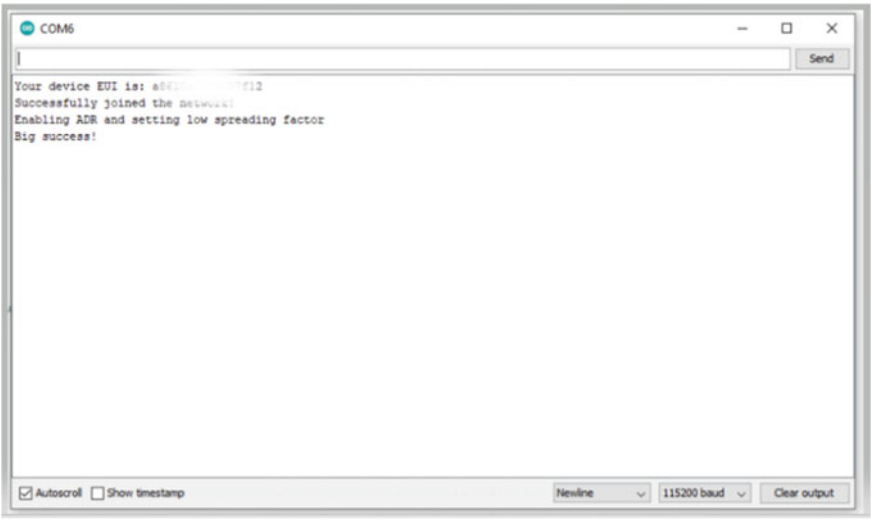


Fig. 2.14 The serial monitor result for the first communication

inputs. The payload assigned in this project has been shown below.

```
// Prepare Cayenne LPP
lpp.reset();
lpp.addTemperature(4, t);  // for temperature
lpp.addRelativeHumidity(3, h);  // for humidity
lpp.addDigitalOutput(2, pirState);  // for PIR count
lpp.addAnalogOutput(4, distance);  // for ultrasonic measurement
```

However, to view the data on TTN, the user need go to the registered device overview on the TTN dashboard and click on payload formats and select Cayenne LPP as shown in Fig. 2.15.

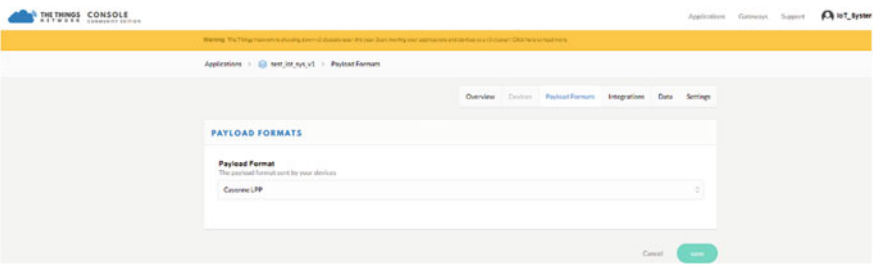


Fig. 2.15 The screenshot of the TTN dashboard to select the payload format

Alternatively, the user may choose Custom payload, and enter the '.json' code snippet to decode the payload assigned as shown in the earlier syntax in the file saved in the GitHub repository to decode the transmitted data.

```
function Decoder(bytes, port) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};
decoded.t = bytes[3] / 10;
decoded.h = bytes[6] / 2 ;
decoded.pirState = bytes[9];
decoded.distance = (bytes[12]<<8 | bytes[13])/100;
  // if (port === 1) decoded.led = bytes[0];
  return decoded}
```

2.4.3.6 Sensor Data on TTN

Once the payload format is set and saved, the user may now transmit the sensor readings using the code 'PIR_DHT_Ultrasonice_ttn_transfer' given in the GitHub repository.

When the code is uploaded, the data will be transmitted for a set frequency of time (which can be modified) and the result may be similar to Fig. 2.16.

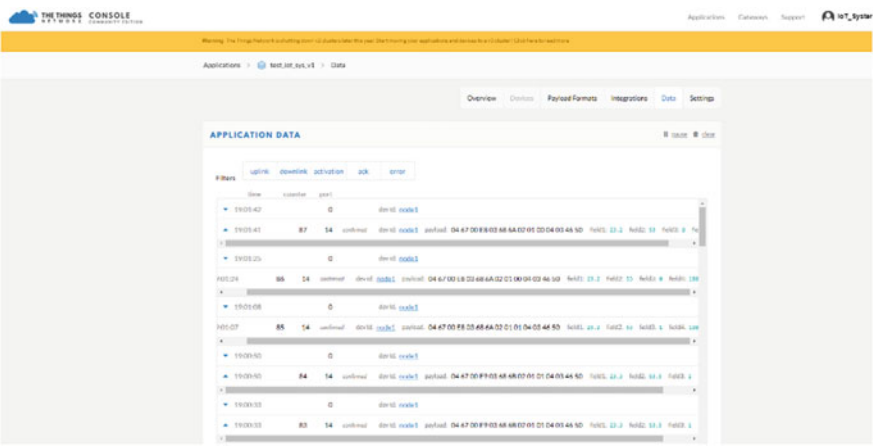


Fig. 2.16 The screenshot displaying the data extracted from the payload decoder on the TTN dashboard

2.4.4 Transmission of Data to Cloud and Visualisation

To transmit the data from the TTN server to the cloud storage is the penultimate stage of an IoT systems design. For this, the user first need to create a ThingSpeak account, as this is the cloud storage platform that will be used in this project. Once the account is created, the user need to add their own channel and create the necessary fields to display the sensor data, as shown in Fig. 2.18.

Next the user, will resume to the TTN application and add integration under the Tab Integrations on the top-right corner. Under the add integration, the user need to select ThingSpeak and they will then be prompted to the integration form as seen in Figs. 2.17 and 2.18.

Now, all details from the created channel such as Channel ID and the API need be added and finally click register. The process ID can be unique to every application integration.

The next stage is to modify the custom payload to transmit data to ThingSpeak. To do this, the user first need to check the field and the respective data they have assigned. Once completed, then the decoder, need to include the field name and the sensor payload assigned. This can be seen in the syntax provided ahead.

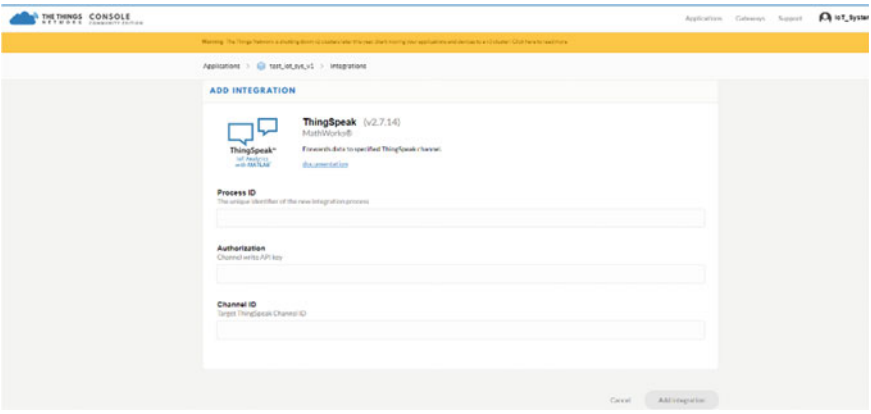


Fig. 2.17 The screenshot displaying the data extracted from the payload decoder on the TTN dashboard

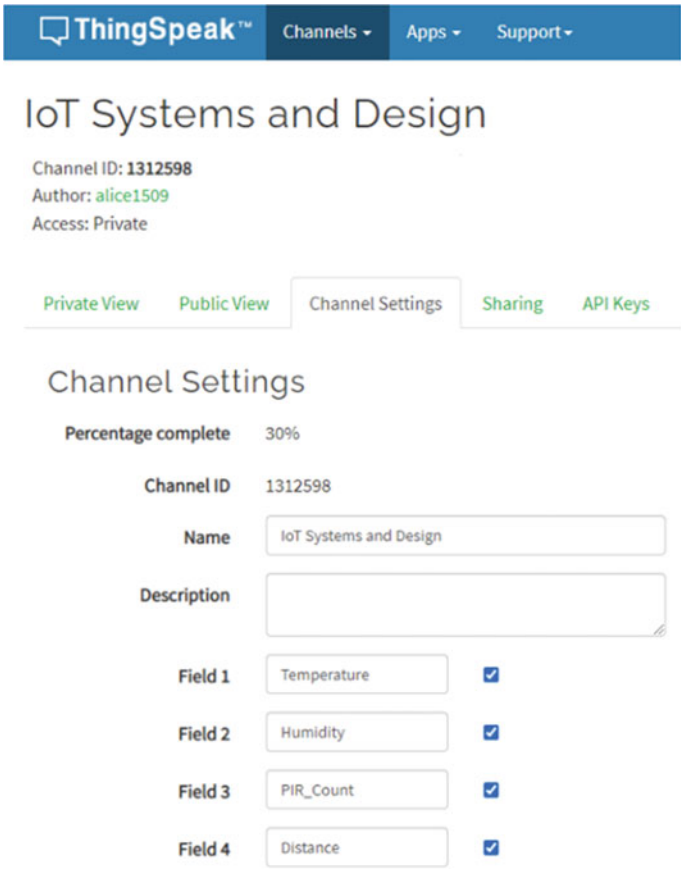


Fig. 2.18 The screenshot displaying the channel configuration for the cloud storage and data visualisation

```
function Decoder(bytes, port) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};
decoded.t = bytes[3] / 10;
decoded.h = bytes[6] / 2 ;
decoded.pirState = bytes[9];
decoded.distance = (bytes[12]<<8 | bytes[13])/100;
  // if (port === 1) decoded.led = bytes[0];

  return {
    field1: decoded.t,
    field2: decoded.h,
    field3: decoded.pirState,
    field4: decoded.distance, }}
```

The final results will be available on the ThingSpeak dashboard as seen in Fig. 2.19. This dashboard is updated and live as long as the data is transmitted from the smart node. This data can then be further used for analysis and investigations by obtaining the excel results from the ThingSpeak channel. Thus, the stages of developing an IoT system is complete.

2.5 Guidelines to Users

This section of the chapter discusses the guidelines provided to the users to implement the project. Thus allowing the users to do their research and learn to troubleshoot and debug their errors. They also learn how to find innovative ways to tackle complex problems arising at every stage of development.

The Smart Node is designed in 4 stages. Each stage has a specific duration which may be checked before moving on to the proceeding stage.

- Stage 1: Microcontroller and Sensor Interface—45 min
- Stage 2: Serial Monitor Results—30 min
- Stage 3: Configuration of the Transmission module—1 h 10 min
- Stage 4: Transmission of data to cloud and visualisation—1 h

2.5.1 *Microcontroller and Sensor Interface*

The first stage discusses the microcontroller and sensors interfacing. Interface the Arduino with the three sensors provided.

- DHT22 sensor
- PIR sensor
- Ultrasonic Sensor

The tasks for the design involve:

- The Arduino is connected to the PC and the LED blink code is run (Note: We are using an MKR1300 board, therefore we need to add the necessary board packages on Arduino IDE).
- The sensors are connected properly following the instruction in the datasheet.
- PIR Sensor is interfaced to the Arduino and its working is checked by displaying the output on the serial monitor.
- Next, only the DHT22 sensor is interfaced, the necessary libraries files are added, and the results are shown on the serial monitor.
- Finally, the Ultrasonic Sensor is interfaced and the results are shown on the serial monitor.

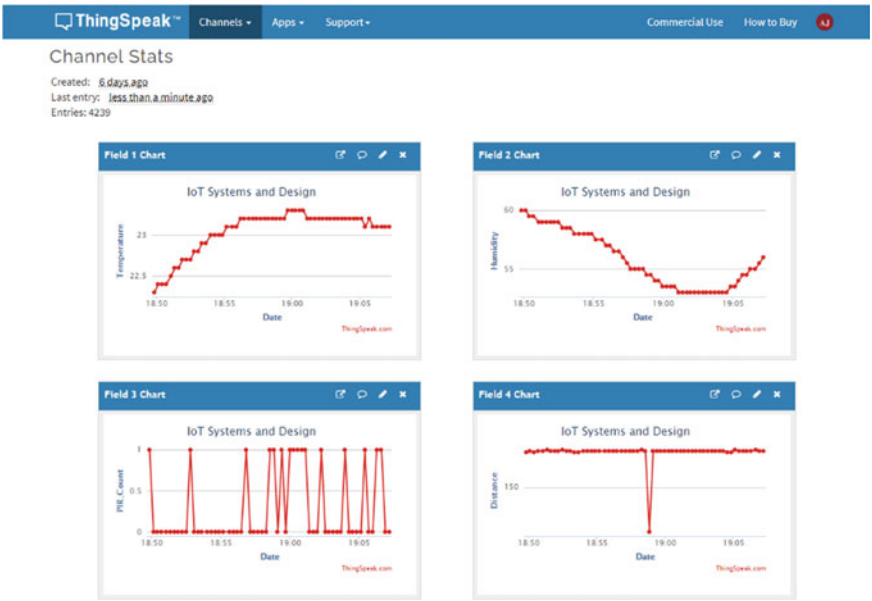


Fig. 2.19 The screenshot displaying the final dashboard of the channel, that gives the sensor results graphically

2.5.2 Serial Monitor Results

After stage 1 is complete, the sensor circuitry and the pin configuration are understood.

- Next, in stage 2, all the three sensors to the Arduino together using the jumper cables and breadboard are connected.
- In the Arduino program, the code with the necessary changes to enable all three sensors working together and displaying data every 25 s is updated.
- The output of the three sensors working on the serial monitor of the Arduino IDE is shown.
- A working node will output the reading of the temperature in degree Celsius/ Fahrenheit, Humidity in percentage, PIR state in binary value i.e. 0 or 1, and distance in cm or mm.

2.5.3 Configuration of the Transmission Module

Once stage 2 is complete, multiple sensor interfacing and programming of the Arduino is learned. Although the results are shown on the serial monitor, the wireless transmission of data is still yet to be achieved. Stage 3 discusses the configuration of the LoRa module in the MKR1300. We will use TTN to transmit Data and connect the gateway.

1. Getting a Unique Device ID:
    - To configure the gateway settings, the user requires the unique device ID for the board.
    - Thus, we need to determine the DeviceEUI for the MKR1300 one's group uses. This result will be displayed on the serial monitor.
2. Setting up TTN:
    - The user will create an application on TTN using the following instructions.
    - Next, by logging in to the TTN website an application is created.
    - Go to https://www.thethingsnetwork.org/.
    - Sign up with the login details. Each console can have many applications. For this project the console used was IoT_System_Design.
    - Once logged in, user needs to click on the console in the top-right corner.
    - Click on Applications.
    - Click on +add application.
    - Write the group number and optionally follow this format (ex: group2)
    - Description: write the project Name
    - Pick the handler ==> ttn-handler-asia-se
    - Click on add application

3. Register the device:
   - Once, the user has registered and created an application, the user now need to register the device in their respective application (i.e. 1 application has 1 registered device).
   - Follow these instructions:
     (i) Go to the application e.g. 'group2'
     (ii) Click on overview on the top-right corner
     (iii) Scroll down, to click on +register device
     (iv) Write the unique device name e.g. node1, and copy the DeviceEUI from the previous step on the given tab.
     (v) Click Register!
4. First Communication:
   - After the user registers the device, it is now required to check communication. We will use the appEUI and appKey to transmit the data from the registered device.
   - Run a 'Hello World' payload using the Arduino IDE and check via Serial monitor and the application page on TTN to validate this communication.
5. Payload Assignment:
   - This is an important task. In TTN, we have the choice to assign a 16/32-bit payload, it is set to a default of 32 bit.
   - We need to determine which payload format we are using (Suggestion: Cayenne LPP, will be good, to begin with!).
   - Once it is done, we can modify our code on the Arduino IDE as required. Once we have assigned the sensor data to the payload we need to configure the payload decoder on TTN.
   - For this, it is required to visit the Payload formats on TTN under the application on the top right of the screen.
   - Further, in the decoder section, we now write the decoder function to interpret the data that is sent using the payload we have used.
   - Depending on the payload we are using, we need to configure our decoder bytes accordingly.
6. Sensor Data on TTN:
   - Now we can upload the code from IDE with a set timer (i.e. how frequently we want to send the data), that will transmit measured data to the TTN server, where it can be viewed in the 'Data' Tab on the top right of our application.

2.5.4 *Transmission of Data to Cloud and Visualisation*

In this stage, the user can now transmit data from TTN to a cloud server i.e. ThingSpeak.

- To do this, first, an account with https://thingspeak.com/ is created.
- Channels are added, to create fields to display measured sensor data.
- Next, we go back to TTN, and under the application, we will find a Tab 'Integrations'.
- We will now +add integration i.e. 'ThingSpeak' to provide the API and the Channel ID when prompted and register our application to the channel.
- Once, completed, we will now go back to our payload format and modify it to transmit the data to ThingSpeak.

These are the stages of a complete IoT system. After this stage, data is extracted from the cloud server to perform analysis for the respective purpose that it has been created, for instance, a Smart City application.

2.6 Outcome of the Project

The IoT system is complete when the system receive data wirelessly from the designed node on a cloud platform and can visualise the data. Figure 2.20 shows the final results obtained on the ThingSpeak web platform displaying the 4 graphs. Each graph represents the data obtained from the individual sensor measurement over time.

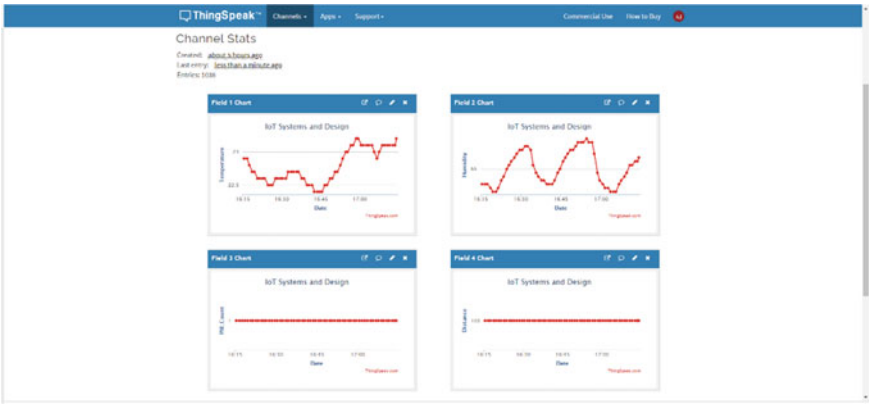
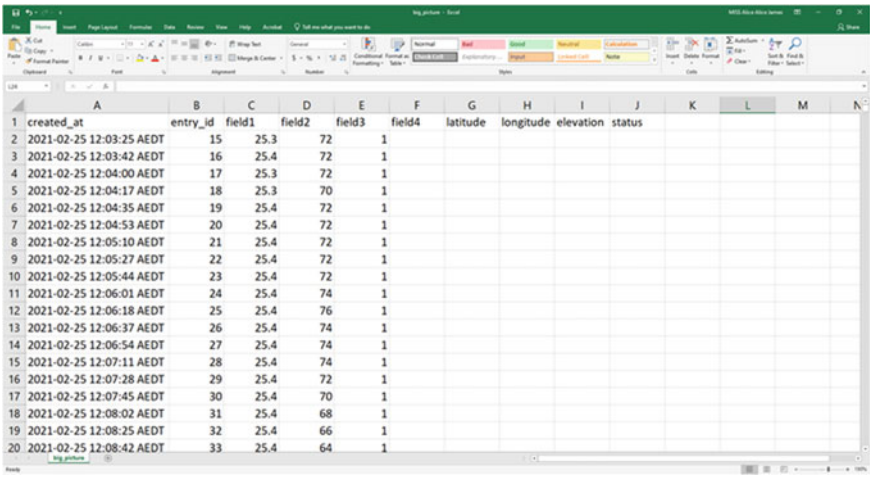


Fig. 2.20 The final screenshot of the ThingSpeak dashboard displaying the sensor readings graphically with respect to time



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	created_at	entry_id	field1	field2	field3	field4	latitude	longitude	elevation	status				
2	2021-02-25 12:03:25 AEDT	15	25.3	72	1									
3	2021-02-25 12:03:42 AEDT	16	25.4	72	1									
4	2021-02-25 12:04:00 AEDT	17	25.3	72	1									
5	2021-02-25 12:04:17 AEDT	18	25.3	70	1									
6	2021-02-25 12:04:35 AEDT	19	25.4	72	1									
7	2021-02-25 12:04:53 AEDT	20	25.4	72	1									
8	2021-02-25 12:05:10 AEDT	21	25.4	72	1									
9	2021-02-25 12:05:27 AEDT	22	25.4	72	1									
10	2021-02-25 12:05:44 AEDT	23	25.4	72	1									
11	2021-02-25 12:06:01 AEDT	24	25.4	74	1									
12	2021-02-25 12:06:18 AEDT	25	25.4	76	1									
13	2021-02-25 12:06:37 AEDT	26	25.4	74	1									
14	2021-02-25 12:06:54 AEDT	27	25.4	74	1									
15	2021-02-25 12:07:11 AEDT	28	25.4	74	1									
16	2021-02-25 12:07:28 AEDT	29	25.4	72	1									
17	2021-02-25 12:07:45 AEDT	30	25.4	70	1									
18	2021-02-25 12:08:02 AEDT	31	25.4	68	1									
19	2021-02-25 12:08:25 AEDT	32	25.4	66	1									
20	2021-02-25 12:08:42 AEDT	33	25.4	64	1									

Fig. 2.21 The final screenshot of the dataset downloaded from the ThingSpeak server

As seen, the first graph represents the temperature, field 2 shows the humidity, 3 shows the PIR count, and 4 shows the distance measured by the ultrasonic sensor. Although these sensors have not been placed strategically for any specific application, the goal of the project was only to understand the connections and wireless transmission.

While visualizing data is necessary, for analysis and development, generating datasets is crucial. Figure 2.21 shows the dataset with values obtained from the sensor readings.

Suggested Reading

1. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
2. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
3. D. Thomas, R. Shankaran, M. Orgun, S. Mukhopadhyay, A secure barrier coverage scheduling framework for WSN-based IoT applications. *IEEE Transactions on Green Communications and Networking* (2021)
4. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the Internet of Things and serious games. Internet Things **13**, 100332 (2021)
5. S.C. Mukhopadhyay, S.K.S. Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)

6. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement **178**, 109424 (2021)
7. S. Pal, M. Hitchens, T. Rabehaja, S. Mukhopadhyay, Security requirements for the internet of things: a systematic approach. MDPI Sens. 5897 (2020)
8. F. Akhter, S. Khadivizand, H.R. Siddiquei, M.E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
9. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using Graphite/PDMS sensors. Sens. Actuators A **286**, 43–50 (2019)
10. M.E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417 (2018)
11. N.K. Suryadevara, S.C. Mukhopadhyay, Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sen-s. J. **12**(6), 1965–1972 (2012)
12. S.C. Mukhopadhyay, J.A. Jiang (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 3, Wireless Sensor Networks and Ecological Monitoring* (Springer, 2013). ISBN: 978-3-642-36364-1
13. S.C. Mukhopadhyay (ed.), *Smart Sensors, Measurement and Instrumentation, Vol. 9, Internet of Things: Challenges and Opportunities* (Springer, 2014). ISBN: 978-3-319-04222-0
14. H. Leung, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 13, Intelligent Environmental Monitoring* (Springer, 2015). ISBN: 978-3-319-12891-7
15. S.C. Mukhopadhyay (ed.), *Smart Sensors, Measurement and Instrumentation, Vol. 15, Wearable Electronics Sensors: For Safe and Healthy Living* (Springer, 2015). ISBN: 978-3-319-18190-5
16. B. George, V.J. Kumar, J.K. Chowdhury, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 25, Advanced Interfacing Techniques for Sensors Measurement Circuits and Systems for Intelligent Sensors* (Springer, 2017). ISBN: 978-3-319-55368-9
17. R. Yan, X. Chen, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 26, Structural Health Monitoring: An Advanced Signal Processing Perspective* (Springer, 2017). ISBN: 978-3-319-56125-7
18. S.C. Mukhopadhyay, T. Islam (eds.), *Wearable Sensors: Applications, design and implementation, IOP Series on Sensors and Sensor System* (2017). ISBN: 978-0-7503-1505-0 (ebook), ISBN: 978-0-7503-1503-6 (print), ISBN: 978-0-7503-1504-3 (mobi). https://doi.org/10.1088/978-0-7503-1505-0
19. S. Mukhopadhyay, T. Islam (eds.), *Innovative Technologies and Services for Smart Cities*, Special issues in MDPI Electronics (2019). www.mdpi.com/journal/electronics. ISBN: 978-3-03921-181-4
20. O.A. Postolache, E. Sazonov, S.C. Mukhopadhyay (eds.), *Sensors in the Age of the Internet of Things: Technologies and Applications* (IET Book, 2019). https://digital-library.theiet.org/content/books/ce/pbce122e. ISBN: 9781785616341, e-ISBN: 9781785616358
21. N.K. Suryadevara, S.C. Mukhopadhyay (eds.), *Assistive Technology for the Elderly* (Elsevier, 2020). ISBN: 978-0-12-818546-9
22. S.C. Mukhopadhyay, B. George, J.K. Roy, T. Islam (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 36, Interdigital Sensors: Progress over the Last Two Decades* (Springer, 2021). ISBN: 978-3-030-62683-9
23. A.K. Sangaiah, S.C. Mukhopadhyay (eds.), *Intelligent IOT Systems in Personalized Health Care* (Elsevier, 2021). ISBN: 978-0-12-821187-8
24. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350
25. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based

smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045
26. N.K. Suryadevara, S.C. Mukhopadhyay, Smart homes: design, implementation and issues. Smart Sens. Meas. Instrum. **14** (2015). 978-3-319-13556-4
27. R. Yan, X. Chen, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 26, Structural Health Monitoring: An Advanced Signal Processing Perspective* (Springer, 2017). ISBN 978-3-319-56125-7
28. S.C. Mukhopadhyay, A. Mason (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 4, Smart Sensors for Real-Time Water Quality Monitoring* (Springer, 2013). ISBN: 978-3-642-37005-2-1
29. S.C. Mukhopadhyay (ed.), *Lecture Notes in Electrical Engineering Vol. 96, New Developments in Sensing Technology for Structural Health Monitoring* (Springer, 2011). ISBN: 978-3-642-21098-3

Chapter 3
Design Considerations for IoT Node

3.1 Introduction

IoT systems consist of many components such as sensors, interfacing electronics embedded processor, wireless transceiver, internet connectivity as well as data storage facilities. When measured data are uploaded to the server, data analysis can be done on available data. To make a successful IoT node, the selection of each component is extremely useful and important from consideration of cost as well as performance over the long term. This chapter will provide some guidelines on the selection of components towards the development of an efficient IoT node.

3.2 Sensors

It is the most fundamental element of a complete IoT system as the sensor provides real-time information of the "Thing". Without sensors, no IoT system is complete so the selection of sensors to be given utmost priority. There are many types of sensors available around us. Sensors may be chosen based on the requirements of the project. There are many suppliers available from whom the sensors can be procured. Performance figures such as accuracy, sensitivity, ease of use, range, drift, etc. are very important parameters. However, sometimes the cost of the sensor, availability, delivery time, replacement options, and the brand of the supplier is to be given more attention. The sensor may be readily available in the market to complete a project before the deadline. Similarly, each component's cost will add up to prevent the design and development of a low-cost IoT system.

In terms of the output of a sensor many choices are to be considered: Analog output sensors come in different configurations, with 0 to ±100 mV, 0 to 3.3 V or 5.5 V or 12 V, 4 to 20 mA current output, audio frequency 0 to 20 kHz signals, or in the ultrasonic range 20 kHz to 1 MHz. If the sensor provides analog output, the

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_3

interfacing the signal to the embedded processor may be through an analog to digital converter (ADC) after a signal conditioning circuit.

Sensors provide digital output that is discrete in nature either LOW (usually 0 V) and High (usually 3.3, 5, or 12 V, etc.). They are easy to interface to embedded processors usually through digital input. The sensor output may be in the form of bit, byte, TTL (transistor-transistor logic), open-collector tri-state, and also are available in different protocols such as I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface), and so on. Proper selection of sensors is extremely useful. In recent times, there are many sensors are available which come with interfacing electronics but may be slightly expensive. Those sensors are convenient to interface with the embedded processor and may be useful in many applications. The operating delay needs to be considered in any time-critical applications. A single-board computer consists of a memory unit, microprocessor, and input/output functions all on a single circuit board making it a fully functioning computing system.

3.3 Smart Sensors

An extended version of the sensor with the capability of data processing, data interpretation, advanced learning, adaptation, and a fusion of data from multiple sources, is known as a smart sensor. Usually, a smart sensor contains an embedded processor to have the ability to perform complex sensing and actuation tasks for high-level applications. The function of a smart sensor is defined with the ability of information processing, integration, compensation, and communication.

3.4 Interfacing Electronics Circuits

Designing an IoT systems with standard available sensors in the market can be challenging. The sensors may provide output that may not be suitable to connect directly to the microcontroller if the sensors are not available as a complete package. Many times the sensor available as a package may be too expensive for the IoT system and may not provide the options of changing any functionality. Interfacing a sensor to an embedded processor needs consideration of many issues such as changing the available voltage level of the output of the sensors (amplification or attenuation), filtering to reduce/eliminate noises in the signal output, converting current to voltage or vice-versa, impedance matching, signal conditioning, and other such functions. Designing the interfacing electronic circuits is the process of selecting proper electronic components between the sensor and the embedded processor so that the signals from the sensor are suitable for processing. The interfacing electronic circuits help to get reproducible outputs from the sensors for measurement, capturing, analysing, and storing information for the processor.

The most important thing is to check the full-scale output voltage of the sensor under normal conditions and convert the output to the nominal operating voltage of the microcontroller (usually 3.3 or 5 V). The common circuit used for this to achieve is a high impedance amplifier configured around an operational amplifier.

Another useful and common circuit used in sensor applications is the filter. There is always noise involved with the measured sensor signal. Low-pass, High-pass, Band-pass, and Band-stop filters are commonly used depending on the application.

In many applications, digital filters are used which can avoid changing the electronic components such as resistors or capacitors. The following section will explain the implementation of analog filters in the digital domain. Figure 3.1 shows a simple active low-pass analog filter.

The output expression of the filter can be written as

$$\frac{V_{out}}{V_{in}} = \frac{\frac{1}{j\omega C1}}{R1 + \frac{1}{j\omega C1}} = \frac{1}{1 + j\omega R1C1} = \frac{1}{1 + j\frac{\omega}{\omega_c}}$$

where $\omega_c = 1/(R_1C_1)$ and is known as angular cut-off frequency.

Assuming the cut-off frequency as 150 Hz, we can get

$$R_1C_1 = 1/\omega_c = 1/(2 * \pi * 150) = 0.00106$$

Let us assume $R_1 = 1\ k\Omega$
So $C_1 = 1.06\ \mu F$
The gain formula is

$$M = 1/(\sqrt{x})$$
$$x = (1 + \omega^2 R_1^2 C_1^2)$$

The gain at 100 Hz is

$$M_{100} = 0.8324$$

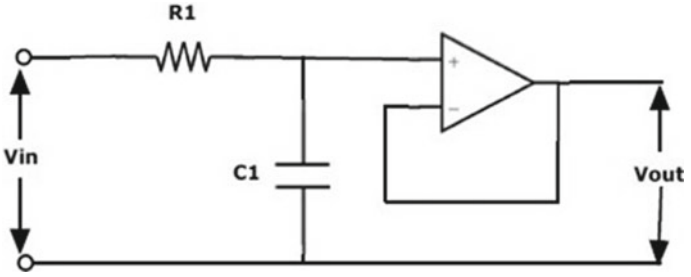


Fig. 3.1 An active low pass filter

The gain at 1000 Hz is

$$M_{1000} = 0.14857.$$

The transfer function of the low-pass filter is,

$$M = 1/(1 + RCs);$$

R is the resistance value, C is the capacitance value and s is the Laplace operator. Assuming Bilinear transformation we have,

$$s = (z - 1) * 2/[T * (z + 1)];$$

T is the sampling period and z is the z-operator.
Replacing s and after simplification, we get,

$$V_o(n) = [V_{in}(n) - V_{in}(n - 1) - (1 - (2RC/T)\, V_o(n - 1)]/(1 + (2RC/T));$$

Suffix n is for the current sampling value, (n − 1) is the previous sampling value. The above equation will implement the low-pass filter with appropriate values of R, C and T values.

3.5 Embedded System

An Embedded System is a fully functional microprocessor/microcontroller based on hardware and software integration to perform specific functions individually or as part of a larger system. In an IoT system, it usually acts as a controlling/processing unit with IC (Integrated Circuit) core for its computing functions. Each command sent to a larger electrical or mechanical system is created into digital data bits sent by the embedded system unit which is programmed specifically. The design considerations of an IoT based system begin at the root level of embedded system. From a single microcontroller to a suite of processors with associated peripherals and networks, and from no user interface to complicated graphical user interfaces, there are many different levels of complexity. An embedded system's complexity varies greatly depending on the tasks for which it was intended. Figure 3.2 depicts the overall structure of an IoT SoC platform's core components. A typical IoT system consists of most of the mentioned components if not all.

An IoT device embedded system can have a very complex design with multitude of on-chip sensor elements, microprocessor and SoCs and using many wireless communication methods like WiFi, LoRaWAN etc. It may also contain just one or two elements in some systems. The power unit may have different power sources such as solar, AC power, battery. Processing units may have microcontroller/microprocessors units. The IoT device also requires RF wireless communication,

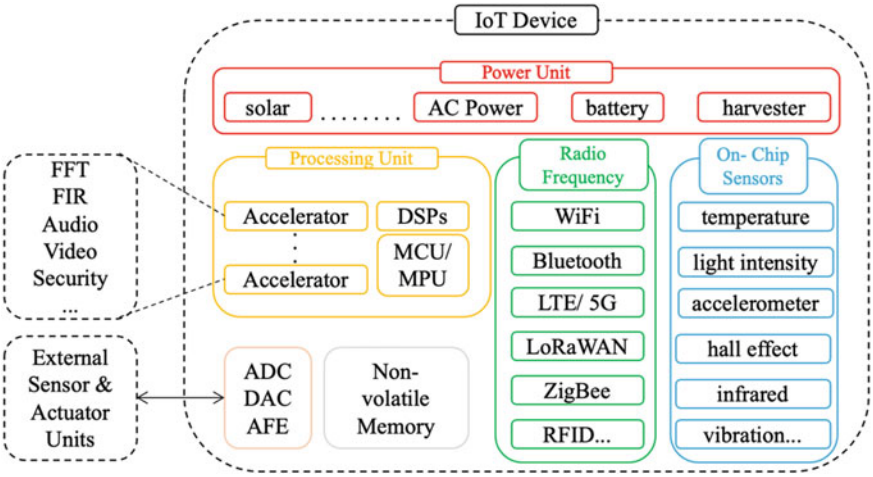


Fig. 3.2 An IoT embedded device's general architecture

such as Long Term Evolution (LTE), Long Range Wide Area Network (LoRaWAN) or Radio Frequency Identification (RIFD) protocols. Sometimes, the devices have embedded on-chip sensors such as accelerometer, temperature to perform detection. The Analog to Digital Converter (ADC), Digital to Analog Converter (DAC), and Analog Front Ends (AFE) allow external sensor/actuator units to interface with the IoT embedded unit. Further, the processing unit also allows external peripheral unit processing such as FFT (Fast Fourier Transform), Finite Impulse Response (FIR) filters among many to enhance its functions if required. Lastly, memory elements such as RAM, ROM are also present in every embedded system to handle storage elements. These units will be discussed in this book in great detail from the stand point of an IoT system design.

The core unit is a microprocessor/microcontroller. Understanding the difference between the two is very important to distinguish the use cases and design considerations of an IoT system. Table 3.1 shows the distinguishing attributes between the two electronic devices.

Table 3.1 discusses and shows that microprocessors differ from microcontrollers and depending on the IoT system requirements, one of them, or sometimes both can be included in the system. For example, an IoT node designed to assist blind people requiring a Computer Vision application, Image to voice and GPS location access would require a microprocessor type embedded system design in its core. Such a device would require a lot of computing power and processing large datasets of image and sound signals. On the other hand, a smart city node that counts passing pedestrians, and tracks environmental data wirelessly to a cloud might benefit from using a microcontroller for its functioning. In this way, noting down the functional requirements can ease out the selection process of an embedded system.

Table 3.1 Comparison between microprocessor and microcontroller in IoT

Attribute	Microprocessor (MPU)	Microcontroller (MCU)
Definition	It functions as a CPU built on an integrated circuit (IC) or a system on chip (SoC)	It is the heart of an embedded system with control I/O functions
Type of system	It only has a processor and required external components, I/O and memory to be connected	It can function standalone as it has a processor, memory and I/O components
Size	With external component requirements, the circuit size becomes larger	With no external component requirements, the circuit size is relatively smaller
Complexity	The computing power is very high with a large complexity in system	The computing power is minimal and is useful only for specific/simple tasks
Number of tasks	It is useful in performing multiple tasks simultaneously	It is designed only for particular task as intended by the requirement
Power consumption	Relatively higher due to complex programs requiring more power	Relatively lower as it runs simple processing algorithms
Power supply	In applications as an IoT node, it requires a larger battery or AC power source due to its power consumption	In applications as an IoT node, it can work with smaller Ah battery types with Li-Ion or LiPo type technology
Cost	The relative cost of it is higher as it requires more complexity in design and external components	It is relatively cheaper as it requires less complexity in hardware design and less power consuming parts
Architecture	Microprocessors are based on Von Neumann model	Micro controllers arc based on Harvard architecture
Applications	IoT devices, computers, laptops, mobile devices, military system, healthcare devices, gaming consoles	Mobile phones, washing machine, ATM, keyboards, oven, watches
Types	Complex and reduced instruction set computer (CISC and RISC), application specific ICs, Digital signal processors (DSPs), graphics processor unit (GPU)	8, 16, 32 Bit microcontroller, embedded microcontroller
Example	Broadcom BCM2711 SoC with a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor found on Raspberry Pi 4	ATmega328 MCU on Arduino

3.6 Wireless Transceivers

Communication is one of the key aspects in any IoT node. Wired or wireless communication systems are available, as well as guided and unguided communication mediums. The channel in Wired Communication is a physical path, such as Co-axial Cables, Twisted Pair Cables, and Optical Fibre Links, that leads the signal between one point to another. The term “guided medium” refers to this type of medium.

Wireless transceivers are devices that allow the reception/transmission of digital data through space without wires. These devices are usually embedded in the IoT node and use different frequency and modulation techniques to communicate with other devices using specialised antennas. Wireless connectivity offers mobility, flexibility and ease of use which has now become the norm for modern electronic devices. Infrastructure is another crucial consideration. Infrastructure development and installation for wired communication systems is an expensive and time-consuming task. Wireless communication infrastructure is simple and inexpensive to set up. It also is a potential solution in disaster situations and remote areas where wired connection is difficult to set up. To completely understand building an IoT node, the design consideration of wireless communication is necessary.

The Transmitter, Channel, and Receiver are the three main components of a conventional Wireless Communication System. The block diagram of a wireless communication system is shown in Fig. 3.3.

Wireless communication takes place in three steps.

Transmission: Information or data signal is generated at the node level as digital bits. The signal from the source is sent through a Source Encoder, which turns it into a format that can be processed with signal processing techniques. This signal is then encrypted to ensure that the signal and data are safe and that no unauthorised access is possible. Channel encoding is a signal processing technique that reduces signal faults such as noise and interference. The signal is then modulated with an appropriate modulation technique (such as PSK, FSK, or QPSK, for example) so that it may be easily broadcast. After that, the modulated signal is multiplexed with additional signals using various multiplexing techniques like TDM, FDM (Time or Frequency Division Multiplexing).

Channel: This signal is then sent through the antenna which generates a voltage at specific frequency. The channel is space which could be air or water medium.

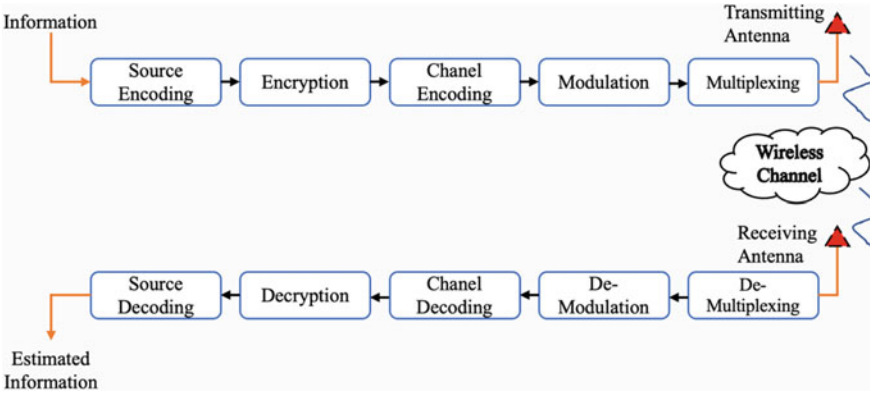


Fig. 3.3 The block diagram of a wireless communication system

Depending on factors like LoS (Line of Sight), distortion, noise etc., the signal is received at the receiving antenna.
Reception: The Receiver's role is to collect the channel's signal and recreate it as the source signal. Demultiplexing, Demodulation, Channel Decoding, Decryption, and Source Decoding are all part of a Wireless Communication System's receiving route. The receiver's role is clearly reversed to that of the transmitter, as evidenced by the components of the receiving route.

3.7 IoT Communication Protocol

The types of wireless communication protocols in IoT are distinguished by many parameters like range, bandwidth, power consumption, etc. Table 3.2 provides a brief overview of the various wireless communication protocols in IoT.

Table 3.2 provides the necessary information to help in deciding which communication protocol to use in selecting an IoT node design project. Further, below are some of the examples of areas of projects that will have these protocols applicable.

1. Industrial IoT: Industrial applications, such as robotics, medical devices, and software-defined manufacturing processes, are included in the IIoT. The most applicable protocols in this domain are LoRaWAN, Cellular and Zigbee.
2. Smart City: A smart city is an urban environment that gathers data using various electrical systems and sensors to create a smart environment. LoRaWAN technology the most suitable here due to its long range and low data rates and power consumption.
3. Smart Home: A smart home is a home environment where regular electronic devices have IoT capability and have an increased sensing mechanism to provide a better lifestyle. Zigbee, BLE, and WiFi are the most used and applicable protocols for such applications due to its range, availability and use case.

Table 3.2 Comparison between the types of wireless communication protocols

Attribute	WiFi	Bluetooth/BLE	LoRaWAN	LTE-M	Zigbee
Typical range	15–100 m	50–150 m	2–20 km	1–10 km	10–100 m
Throughput	Up to 700 Mbps	100 kbps–2 Mbps	10–50 kbps	Up to 1 Mbps	20–250 kbps
Power consumption	Medium	Low	Low	Medium	Low
Cost	High	Low	Low	High	Medium
Topology	Star, Mesh	P2P, Star, Mesh	Star	Star	Mesh

4. Wearables: Products like smartwatch, fitness trackers and electronic devices that are worn by humans to provide sensor data are commonly known as wearables. BLE is the most useful protocol here due to its low power consumption and data rates.
5. Aggrotech: Smart agriculture devices are becoming increasingly popular with IoT technology becoming affordable. Such IoT devices help to identify useful environment and soil parameters in an agriculture setting. LoRaWAN is the most applicable protocol here due to its long range and low power consumption that will be required in these rural settings.

In this book we will develop IoT nodes using WiFi, BLE and LoRaWAN based communication protocols.

3.8 Power/Energy for Sensor Nodes

Wireless sensor networks have proven critical to a variety of Internet of Things applications, such as environmental sensing and healthcare monitoring. Due to the need for sustainable and reliable operation of the sensing nodes, major research efforts have been devoted to powering these nodes. Maintaining a stable power to the wireless nodes comes down to finding the key parameters that consume power at the μW level. Some applications of IoT can have nodes to function for continuous operation over the period of several months to years.

When selecting the correct power source for designing an IoT sensor node, many factors have to be taken under consideration. The process of selecting the correct power source is complex and involves multiple calculations for run-time, data frequency, CPU usage etc. Table 3.3 provides the suitable power source for different IoT applications depending on the sensor type and operation frequency.

Table 3.3 shows the processing capability necessary to perform fundamental operations and processing operations on recorded signals from various sensors in various IoT applications. The power source types are divided into 4 types from Energy Harvesters, Coin-Cell Battery, Rechargeable Li-ion battery, and AC Power outlet.

Some types of energy harvesters used in IoT applications are solar, RF, vibration, wind, etc. New research has also found that sweat is used as an energy harvester in E-Skin and wearable applications. These devices usually are used in ambient monitoring, body area network IoT applications.

Healthcare applications with low frequency of operations (50–900 Hz) rely on low power sourced electronics. These devices have embedded sensing system that measure respiratory and heart rate biomarkers. They usually have coin-cell battery technology used due to its light weight, small size, high cell voltage, etc. The anode is usually zinc or lithium based and they have at 1.5–3 V and can be easily stacked.

Electrocardiography (ECG), electromyography (EMG), and electroencephalography (EEG) applications operate at about 1 kHz to 1 MHz and thus require

Table 3.3 Suitable power source considerations based on the IoT application area

IoT Application	Sensor type	Operating processing frequency (Hz)	Suitable energy source
Healthcare apps	Respiratory	Low	Coin-cell battery
	EEG	Medium	Rechargeable Li-ion battery
	Heart rate	Low	Coin-cell battery
	EMG	Medium	Rechargeable Li-ion battery
	ECG	Medium	Rechargeable Li-ion battery
Security surveillance	Video	High	Rechargeable Li-ion battery
	Audio HD	High	Rechargeable Li-ion battery
	Video HD	Very high	Power outlet
Smart city	Video LD	High	Rechargeable Li-ion battery
	Image	High	Rechargeable Li-ion battery
	Acoustic	High	Rechargeable Li-ion battery
Fitness activity	Magnetometer	Medium	Rechargeable Li-ion battery
	Gyroscope	Medium	Rechargeable Li-ion battery
	Accelerometer	Medium	Rechargeable Li-ion battery
Structural apps	Vibration	Very high	Power outlet
	Velocity	High	Rechargeable Li-ion battery
Ambient monitoring	Temperature	Very low	Energy harvesters
	Humidity	Very low	

rechargeable Li-ion battery as best suited power source. These batteries charge rapidly, perform longer, and offer a higher power density than conventional batteries, enabling for more battery life in a smaller size ratio.

Security surveillance IoT applications usually involve heavy recording and thus power processing power requirements. Although the low quality video and HD audio can be powered by Li-ion battery, HD video requires a stable AC power source due to very high processing requirements. Similarly, rechargeable Li-ion batteries are also suitable in smart city and fitness/physical activity tracking applications.

3.9 IoT Security

The IoT is a huge market worth billions of dollars in the consumer, enterprise, and industrial world. Their application areas are multiplying by the day, implying that they will have a greater impact on our lives in the future.

Table 3.4 depicts the various components of the IoT ecosystem, as well as security issues for each.

Each new device which arrives is a potential entry point for ransomware, spyware, or other virus to be injected into a system, causing harm to everything it encounters. This can only be avoided with careful security planning and thorough follow-up.

Since IoT devices have a wide range of data formats and computing capacity, there is no "all-purpose" solution for securing an IoT implementation. Every component of the system has its own range of security concerns. Robust cybersecurity design will consider each component and the system as a whole. Some of the general practices and use cases one can implement in their IoT node design are given below:

- Deciding the level of security depending on the value of data is helpful.
- Creating strong and unique passwords is critical in any IoT based system.

Table 3.4 Various components of the IoT ecosystem and security issues

IoT application layer	IoT features	Possible security concern
Web and mobile	High data volume process	Poor user/third part authentication
	Open/closed platforms	Code
	Variable regulations	Vulnerability testing
Cloud	Private/public/mix cloud deployment	Program/code
		Policy organisation
Communications	2G, 3G, LTE, 5G	Insecure communications and packet loss
	MQTT, Z-wave, Mesh RF, WiFi	
	DSL Fibre, LPWAN	
	WiFi, Bluetooth	
Gateways/edge devices	Variable communication protocols	Policy organisation
	Time critical data analysis	Denial-of-service (DoS)
		Faulty hardware design
IoT sensor/ actuators	Limited power source	Design errors
	Low Bandwidth	Software/firmware design
Data types	Sensitive data: video audio, location	Users, policy management
	Technical data: monitored environmental data	Data storage

- For custom projects, building a personal web server to gather IoT sensor data reduces the threat level.
- Installing and updating software/firmware with necessary security protocols such as firewall helps prevent virus and hackers from getting access to the sensitive data.
- Performing analysis on data at the cloud/gateway level is recommended rather than on the node level where security could be compromised.

3.10 IoT Access Control

Access control is a crucial part of data security since it determines who has access to and uses company data and resources. It regulates the individuals and users based on the levels of security protocol been designed on the system. IoT communication network can have multiple devices, sensors and control systems on the same IP which could be connected wired or wirelessly. One of the main fields where IoT security is implemented and operated through an IoT network is building access control. Protecting private information which may pose security risks if leaked is critical to every organisation. Some of the features of an IoT based access control security design are as follows:

- Indoor wayfinding: Indoor wayfinding is the system's ability to allow accessibility options to users before entering via providing UI on mobile devices.
- Secure access control: Building secure access credentials to users. For example, smart lock doors which could be opened from distance.
- Instant authorisation: IoT based authentication that is fast and efficient in managing authentications instantly are an important feature of access controlling.
- Convenient interaction: The communication between devices in the network is seamless and enables users to engage with one another more easily, as well as providing location information.
- No Physical ID: Reducing the amount of physical ID generated allows the risk of it losing or getting stolen.
- Interoperability: Software and APIs, which have grown more open than ever in the IoT era, are the key to interacting with other devices and ensuring seamless interoperability.
- User Experience: Designing the node network with simple yet powerful user interface (UI) and making installation and deployment simple is essential. Mobility is an aspect of this, as it allows users to control access and other networked equipment from their mobile device.

3.11 IoT Data Storage

The principal purpose of an IoT device is to sense and provide appropriate data. With the increase in demand for access to multiple information, the concept of big data was introduced. Nowadays, each device however small it may be generating data that needs to be stored. Consider an example of a IoT based toothbrush. This smart toothbrush is Bluetooth enabled and connected to the phone. The toothbrush intelligent system can tell the time of use, activities performed with the brush, duration of usage among other data. Such cases show how important data storage is.

Devices create data that is transmitted to the primary application to be forwarded on, consumed, and used across the Internet of Things. Data can be delivered in real time or in batches at any time, depending on the device, network, and power consumption constraints. The true value, however, is determined by the order in which data sets are generated.

There are many services available commercially that can be utilised to store the IoT node data. The application area, range of node, and data type are among the few design considerations to keep in mind. The existing IoT infrastructure is divided into three levels:

1. Low-power IoT devices at the node level.
2. A hypothetical middle-tier gateway that connects IoT devices to back-end Database server.
3. The back end where IoT data is stored and analysed to improve IoT applications.

Figure 3.4 shows the storage services for IoT Data.

Storage services with different platforms have been shown in Fig. 3.4. At the centre of Fig. 3.4 is the 'Internet' from IoT that allows the 'Data' transfer to different storage servers.

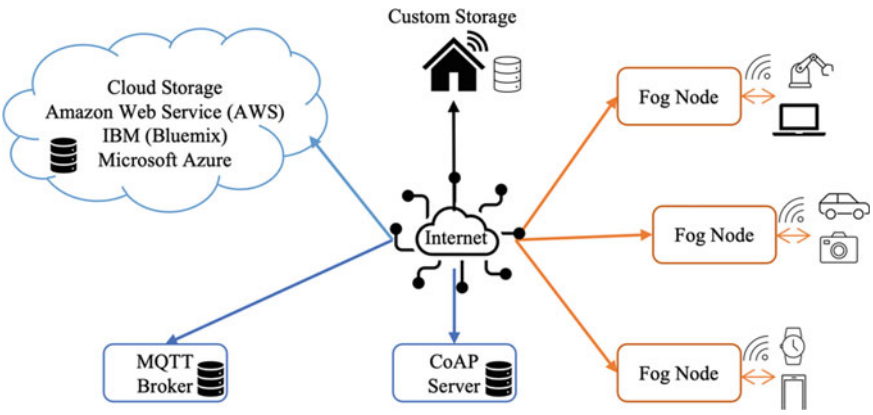


Fig. 3.4 Storage services for IoT data

- Cloud storage: These are commercially available cloud storage platforms that allow various features and plans depending upon the usage and data storage. These storages are used extensively by developers and companies and require their separate protocols and programming to communicate between the devices.
- MQTT broker: MQTT stands for Message Queuing Telemetry Transport. An MQTT broker is a server that collects incoming messages from clients and forwards them to the correct destination clients. An MQTT client is any component that runs a MQTT library and connects to a MQTT broker across a network (anything from an embedded controller to a full-fledged server).
- CoAP Server: CoAP stands for Constrained Application Protocol. CoAP is a protocol that allows basic, restricted devices to connect to the Internet of Things, even across constrained networks with poor bandwidth and availability. Machine-to-machine (M2M) applications such as wearables and supply chain management are common uses.
- Fog node: The fog nodes provide fog networking services which are distributed fog computing entities. They are present in between the IoT node and the cloud storage system. In many cases, sending data to the main cloud consumes too much power and data bytes. It allows edge devices to perform powerful computing, storage, and storage locally with mobility.

3.12 IoT Data Analytics

Data Analytics has the potential to unlocking the true power of IoT. In simple terms, IoT Data Analytics is analysing huge volumes of data produced by IoT devices. Analysing data allows to understand and study the systems in great detail and optimise it. Recently, Artificial Intelligence has played a huge role in how data is analysed and perceived.

The first step to data analysis is retrieving the necessary data from the node level or data base cloud server that is storing the data. The data available needs to properly be formatted into the appropriate data point file before it can be analysed. This step is called processing and formatting of data. The following points show the steps one can take before analysing the data:

- Ensuring that the data is formatted or converted into a standardized format that is consistent with the system is helpful.
- Making a copy of the newly changed file or produce a backup of it is important.
- To enhance the efficacy, filtering away any data that is repeated, old, or unnecessary helps.
- To help improve the present data set, incorporate extra organized (or unorganized) data from different sources.

On IoT data, there are various forms of data analytics that can be implemented:

- Prescriptive analytics: Prescriptive analytics is used to determine which actions may be taken in a given situation. It is suitable when aiding in deciphering of enormous amounts of data in order to arrive at more exact conclusions.
- Spatial Analytics: This method is used to examine IoT data and apps that are based on location. Such analysis is useful in applications like smart parking, autonomous vehicles, and also agriculture farming.
- Streaming Analytics: Streaming analytics, also known as event stream processing, makes it easier to analyse large "real-time" data sets. These are useful in emergency and urgent situation type application areas. IoT based applications that would use such services are fire detection nodes, flood management devices, and also health monitoring.
- Time series analytics: Time series analytics uses time-based data to determine inconsistencies, correlations, and insights. Fitness and ambient monitoring, and health tracking systems are three applications that tremendously apply to time series analytics.

Suggested Reading

1. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement **178**, 109424 (2021)
2. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
3. D. Thomas, R. Shankaran, M. Orgun, S. Mukhopadhyay, A secure barrier coverage scheduling framework for WSN-based IoT applications. IEEE Trans. Green Commun. Netw. (2021)
4. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the Internet of Things and serious games. Internet Things **13**, 100332 (2021)
5. S.C. Mukhopadhyay, S.K. Sah Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
6. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
7. S. Pal, M. Hitchens, T. Rabehaja, S. Mukhopadhyay, Security requirements for the internet of things: a systematic approach. MDPI Sens. 5897 (2020)
8. F. Akhter, S. Khadivizand, H.R. Siddiquei, M.E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
9. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using Graphite/PDMS sensors. Sens. Actuators, A **286**, 43–50 (2019)
10. M.E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417 (2018)
11. N.K. Suryadevara, S.C. Mukhopadhyay, Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sen-s. J. **12**(6), 1965–1972 (2012)

12. S.C. Mukhopadhyay, S.K.S. Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
13. S.C. Mukhopadhyay, J.A. Jiang (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 3, Wireless Sensor Networks and Ecological Monitoring* (Springer, 2013). ISBN: 978-3-642-36364-1
14. S.C. Mukhopadhyay (ed.), *Smart Sensors, Measurement and Instrumentation, Vol. 9, Internet of Things: Challenges and Opportunities* (Springer, 2014). ISBN: 978-3-319-04222-0
15. H. Leung, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 13, Intelligent Environmental Monitoring* (Springer, 2015). ISBN: 978-3-319-12891-7
16. S.C. Mukhopadhyay (ed.), *Smart Sensors, Measurement and Instrumentation, Vol. 15, Wearable Electronics Sensors: For Safe and Healthy Living* (Springer, 2015). ISBN: 978-3-319-18190-5
17. B. George, V.J. Kumar, J.K. Chowdhury, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 25, Advanced Interfacing Techniques for Sensors Measurement Circuits and Systems for Intelligent Sensors* (Springer, 2017). ISBN: 978-3-319-55368-9
18. R. Yan, X. Chen, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 26, Structural Health Monitoring: An Advanced Signal Processing Perspective* (Springer, 2017). ISBN: 978-3-319-56125-7
19. S.C. Mukhopadhyay, T. Islam (eds.), *Wearable Sensors: Applications, design and implementation, IOP Series on Sensors and Sensor System* (2017). ISBN: 978-0-7503-1505-0 (ebook), ISBN: 978-0-7503-1503-6 (print), ISBN: 978-0-7503-1504-3 (mobi). https://doi.org/10.1088/978-0-7503-1505-0
20. S. Mukhopadhyay, T. Islam (eds.), *Innovative Technologies and Services for Smart Cities*, Special issues in MDPI Electronics (2019). www.mdpi.com/journal/electronics. ISBN: 978-3-03921-181-4
21. O.A. Postolache, E. Sazonov, S.C. Mukhopadhyay (eds.), *Sensors in the Age of the Internet of Things: Technologies and applications* (IET Book, 2019). https://digital-library.theiet.org/content/books/ce/pbce122e. ISBN: 9781785616341, e-ISBN: 9781785616358
22. N.K. Suryadevara, S.C. Mukhopadhyay (eds.), *Assistive Technology for the Elderly* (Elsevier, 2020). ISBN: 978-0-12-818546-9
23. S.C. Mukhopadhyay, B. George, J.K. Roy, T. Islam (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 36, Interdigital Sensors: Progress over the Last Two Decades* (Springer, 2021). ISBN: 978-3-030-62683-9
24. A.K. Sangaiah, S.C. Mukhopadhyay (eds.), *Intelligent IOT Systems in Personalized Health Care* (Elsevier, 2021). ISBN: 978-0-12-821187-8
25. N.K. Suryadevara, S.C. Mukhopadhyay, Smart homes: design, implementation and issues. Smart Sens. Meas. Instrum. **14** (2015). 978-3-319-13556-4
26. S.C. Mukhopadhyay, A. Mason (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 4, Smart Sensors for Real-Time Water Quality Monitoring* (Springer, 2013). ISBN 978-3-642-37005-2-1
27. S.C. Mukhopadhyay (ed.) *Lecture Notes in Electrical Engineering Vol. 96, New Developments in Sensing Technology for Structural Health Monitoring* (Springer, 2011). ISBN 978-3-642-21098-3
28. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350
29. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045

Chapter 4
Programming Raspberry Pi for IoT System

4.1 Introduction

The invention of low-cost computers based on integrated circuits revolutionised modern society. The 'Microprocessors (μP)' can now be found in complex systems like spacecraft electronics to household devices such as smart refrigerators and to a simple toy.

A microprocessor plays a vital role in programming an IoT based system. It is the core central unit responsible for computing and processing the on-board tasks. Microprocessors are usually embedded on a single integrated circuit or a small number of ICs. Combinational and sequential logic, both can be computed by a microprocessor. It performs operations with an ALU (Arithmetic Logical Unit) and also runs communication with multiple devices in a connected network.

It is very useful to know the fundamentals of a microprocessor functionality to program it effectively towards making an IOT system. The process of fetching, decoding and executing is performed sequentially. Primarily, the instructions are saved in the memory in sequence. These instructions are read/fetched by the μP, decoded and executed till the request is completed. Lastly, the binary formatted data is fed into the output port. The process also involves the ALU which performs the operations and registers which store data temporarily. The block diagram of this working is shown in Fig. 4.1.

Thus, a μP is a fundamental unit of an IoT based system. Further, this chapter will take a deep look into understanding the software design methods and language used to program the microprocessor for IoT based applications. In this chapter, the Raspberry Pi SBC has been explained to understand these functionalities.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_4

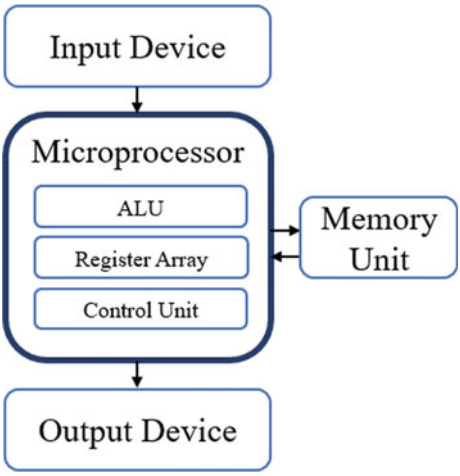


Fig. 4.1 The fundamental block diagram of a microprocessor unit

4.2 Understanding Single Board Computers (SBCs)

A single board computer consists of a memory unit, microprocessor and input/output functions all on a single circuit board making it a fully functional computing system. It also has multiple features that allow it to communicate and connect to other peripheral devices, although rarely rely on expansion slots for peripheral functions or expansion. Static RAM with low-cost 8- or 16-bit processors are frequently used in simple systems, such as those created by computer hobbyists.

Compared to personal computers, the SBC's simple design architecture relies on its ability to function as an embedded computing controller to operate complex devices. The SBCs are dependable devices that emphasize on adaptability to the application specific usage which results in faster processing and reduced bugs in the software system. Single-board computers are frequently used as demonstration or development computers, educational systems, or embedded computer controllers. The open source SBCs are popular with hobbyists and people who are getting started into the world of embedded computers.

In an IoT based system, an SBC is used as the central brain of the system. Depending on the application, there are several types of SBCs available in the market. Commercial businesses that need to design a complete IoT based device, often develop their own SBC from scratch, with the necessary microprocessor system on chip (SoC), memory elements and other connectivity hardware and software features. However, for the purpose of research and rapid prototyping, engineers often make use of commercially available SBCs for creating IoT based devices.

These usually offer open source software and resources to implement basic functionalities. Also, it is significant to understand the functional requirements of the system design to identify the type of board needed.

An overview block diagram of an IoT based system using an SBC is shown in Fig. 4.2. For prototyping an IoT based system with an SBC it is important to first

define the goal of the project. Depending on the functional requirements, the sensors and actuators are selected. Next, different types of SBCs are compared and the most suitable board is selected. This board will have installation processes for the operating system (OS). There are various programming software/languages one can choose to write the instruction set for the device. For an IoT device, wireless communication through WiFi, Bluetooth, etc. to the cloud storage gateway is necessary. Data can then be shared bi-directionally through the gateway. Finally, the data can be visualised on a Mobile App or a Desktop Website. Setup can also be used from these devices to control the IoT node. The uploaded data can be used to do analysis and apply machine learning algorithms to find predictions/anomalies in the data. The data analytics can either be performed on node level (Edge Computing) or downloaded from the cloud server first before performing any analysis. There are many other elements in an IoT system, however, for the purpose of this chapter, only the ones concerned with SBCs and associated hardware have been shown. Thus, the SBC hardware boards play a crucial role in connecting and processing the various elements.

So far, it has been seen that SBCs play a critical role in any IoT based system. Some of the most popular SBCs are listed below:

- Raspberry Pi-4 (RPi)
- NVIDIA Jetson Nano
- BeagleBone Boards
- Odroid-XU4
- Google Coral Dev Board
- Asus Tinker Board

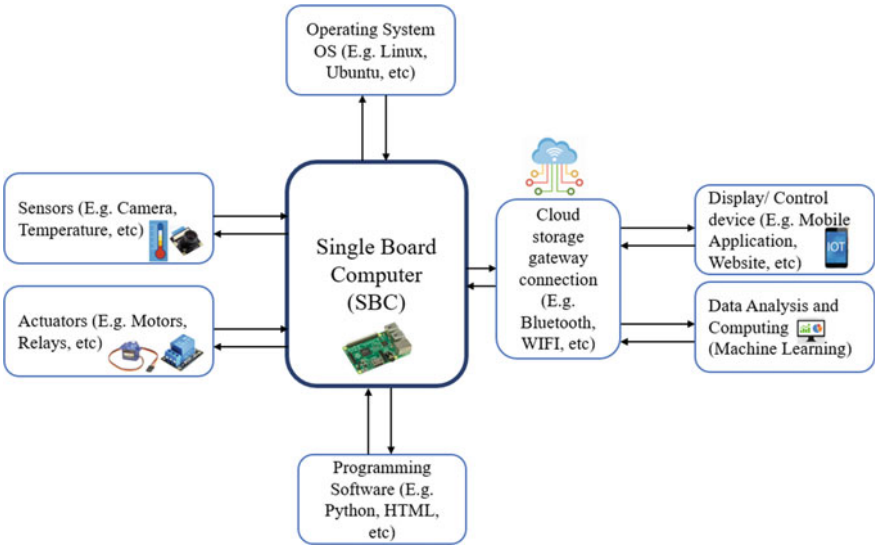


Fig. 4.2 An overview block diagram of an IoT based system using an SBC

- LattePanda
- Ultra96-v2 Board
- RockPi N10

Most of these SBCs have a few common features. Most have WiFi, Bluetooth connectivity. Most of them can be powered via a 5 V power supply. All of them have ARM based processors, with ethernet, USB, display port, GPIO with I2C, and SPI functionalities.

These boards were distinguished against each other and the top 3 boards compared are as shown in Table 4.1. NVIDIA Jetson Nano and Google Coral Dev Board, both have very similar features, the GPU performance is subpar, and the

Table 4.1 Comparison between different SBCs for IoT System prototyping

Attribute	Google Coral Dev Board	Raspberry Pi-4	NVIDIA Jetson Nano
CPU	NXP i.MX 8 M SoC (quad Cortex-A53, Cortex-M4F)	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz	Quad-Core ARM Cortex-A57 64-bit @ 1.42 GHz
GPU (AI Accelerator)	Integrated GC7000 Lite Graphics	Broadcom VideoCore VI (32-bit)	NVIDIA Maxwell w/128 CUDA cores @ 921 MHz
Memory	4 GB LPDDR4	4 GB LPDDR4	4 GB LPDDR4
Connectivity	Gigabit Ethernet/ WIFI 802.11ac/ Bluetooth	Gigabit Ethernet/ WIFI802.11ac/ Bluetooth	Gigabit Ethernet/M.2 Key E (for WIFI support)/ Bluetooth
Display	HDMI 2.0a	2x micro-HDMI (up to 4Kp60)	HDMI 2.0 and eDP 1.4
USB	Type-C OTG; Type-C power; Type-A 3.0 host; Micro-B serial console	2x USB 3.0, 2x USB 2.0	4x USB 3.0, USB 2.0 Micro-B
Pins	40-pin GPIO	40-pin GPIO	40-pin GPIO
Camera	MIPI-CSI2 camera support, MIPI-DSI display	MIPI CSI port, Video Encode: H264(1080p30)	12 lanes (3 × 4 or 4 × 2) MIPI CSI-2, DPHY 1.1 (1.5 Gbps), Video Encode: H.264/H.265 (4Kp30)
Storage	Micro-SD upto 256 GB	Micro-SD upto 256 GB	Micro-SD upto 64 GB
Input Power	5 V DC (USB Type-C)	USB C-Power Supply	12 V barrel jack connector
TensorFlow OS	Supports TF Lite	Supports TF Lite	Supports
Community Support	Medium	High	Medium
Cost	~Au$170	~Au$109	~Au$165

higher processor gives ease of use while multiple programs are running. A higher GPU also means that the AI accelerator has a better benchmark when running ML Models for TensorFlow, PyTorch, or any other framework. The Raspberry Pi-4 on the other hand has some minor differences in features. Although the RPi can run TF Lite, its GPU is not as good as its competitors. The RPi was chosen due to its ease of access, cost, community support and basically popularity among people who are just starting out into IoT System prototyping. For more advanced users looking to get deep into the AI and Edge Computing section, the Dev board and Jetson Nano are great choices.

4.3 Setup and Installation

For the purpose of this book, the Raspberry Pi-4 Model B (4 GB) is selected as the computing board. The following sections will go into the detail of using this SBC and writing programs to perform lab work for IoT based applications.

4.3.1 Learning Objectives

The major aim of this chapter is to give a general idea on how to setup, install and run basic programming codes on an SBC. The following points describe the objectives in detail:

- Understanding the key components involved in working with SBCs.
- Setting up and writing basic codes with simple algorithms.
- Experimenting with sensors, and camera modules to interface them with SBCs.
- Using hardware and software interfacing to program models.
- Learning about Operating Systems and Programming Languages.

4.3.2 Hardware Requirements

The hardware components used in this section mainly involve the sensors, camera and microprocessor unit. Figure 4.3 shows the Raspberry pi-4 board with its major I/O ports.

The following components are required to build this system:

- Microprocessor SBC: Raspberry Pi-4 Model B
- Display: A Monitor with HDMI Support
- Input: USB Keyboard and USB Mouse

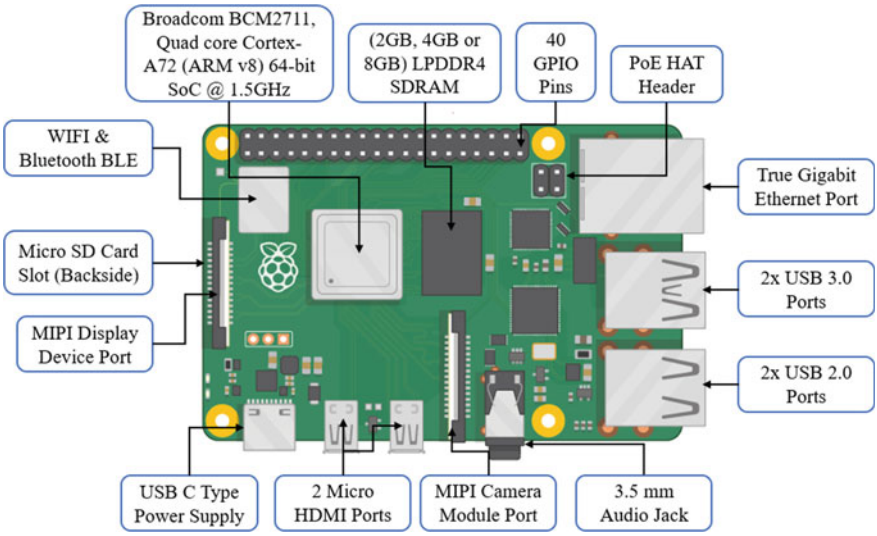


Fig. 4.3 An overview of the Raspberry Pi-4 Model B SBC

- Cables: 5 V USB C for Power (or Official Power Adapter), Micro HDMI to HDMI for Display, USB hub (optional)
- Connectivity: LAN Cable or WIFI Connection for Internet
- Storage: MicroSD Card 8 GB minimum and card reader
- Computer: A Mac, Linux or Windows computer
- Camera: Raspberry Pi Camera Module
- PIR Sensor Module
- LEDs, Push Buttons, Resistors (Multiple Values)
- Prototyping Solderless Breadboard
- Jumper Cables.

4.3.3 Software/Applications Requirements

The following are the software/applications used in this project:

- Raspberry Pi-4
- Operating System: Linux based Raspberry Pi OS
- Microprocessor Programming: Python Programming Language
- IDE: Thonny Python, IDLE, or Geany
- Imager: Raspberry Pi Imager (Install on Desktop PC).

4.3.4 Initial Setup and Installation on SBC

The initial setting up of an SBC is most often similar between different boards. It usually involves Flashing an SD Card with suitable OS on a desktop PC before inserting it in the board. The memory card, once flashed, cannot be used to store files via transfer (e.g. using this micro SD card in your mobile phone will show a drastically reduced memory size and limitations in usage). It can only be used as a storage inside the SBC. To use it in other applications or flash another OS, it has to be reformatted appropriately. This is the normal setup requiring a separate monitor, keyboard and mouse connected to the board to function. The following points describe the step-by-step initialisation process.

4.3.4.1 SD Card Flash

Before starting to use the RPi, one needs to flash the OS on the SD card.

- Install the RPi Imager on a desktop PC.
- Insert the micro SD card on a desktop/laptop via a USB Hub.
- Run the program and choose the OS as (Raspberry Pi OS-32bit) which is a Linux based OS.
- Select the correct micro SD card from the imager you want to write the OS on.
- Click 'WRITE' and wait for the program to be installed.
- After the process is done, remove the SD card from the desktop/laptop.

Figure 4.4 shows the RPi Imager running on a Windows PC.

4.3.4.2 RPi Connection

After the flashed SD card with the required OS, the SD card is inserted into the RPi. Refer to Fig. 4.3 to understand the ports and connections. The following are the guidelines for RPi Connection:

- To start the flashed SD card is inserted into the RPi.
- The RPi is powered up by connecting its power supply provided or the USB-C cable.
- The Keyboard and Mouse are connected to the USB ports on the RPi.
- The RPi is connected to the desktop monitor via a micro HDMI cable.
- The RPi does not have a power switching button. The red LED light turns on as soon the power supply is connected.
- One can also connect an audio input to the RPi for sound.

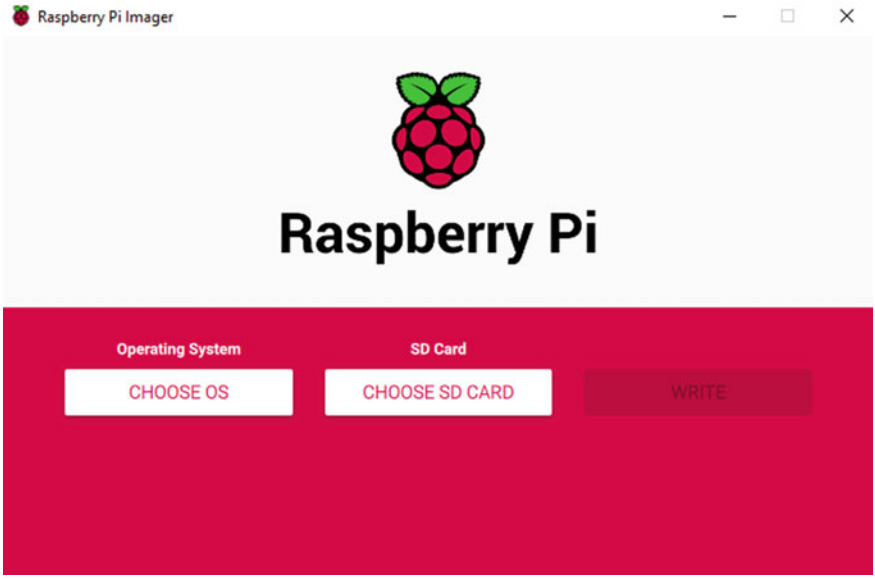


Fig. 4.4 A screenshot of Raspberry Pi Imager

4.3.4.3 Start up the RPi

After the necessary hardware is connected, the RPi start up is configured as follows:

- After powering up the RPi and connecting it to the display, the OS desktop is visible on the screen. It will take a few minutes to load the Desktop screen on the monitor.
- Before using the desktop, it will be useful to finish the set-up, by selecting the language, country, etc.
- A preferred password may be used for the RPi.
- Next, to connect to the internet, either use an ethernet cable or connect to a WiFi from settings. Figure 4.5 shows how the desktop looks like on the monitor.
- The RPi is restarted to finish the updates and setup.
- For the updates, go to 'Terminal' from the top-left corner and in the 'Terminal' window type these two commands and press enter after each one:

```
sudo apt update
sudo apt upgrade
```

- Now, we go to the RPi configuration from the top left corner > preferences > Raspberry pi configuration and change the hostname as the preferred name.

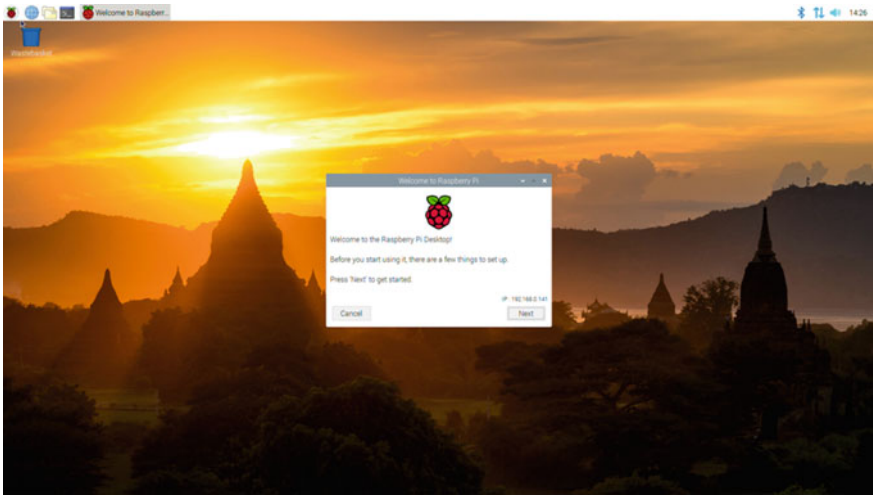


Fig. 4.5 A screenshot of Raspberry Pi Desktop with RPi OS

- Another important tool is the Rpi Configuration Tool. We can run the following command to open the Config tool:

```
sudo raspi-config
```

- This shows the configuration options for the Rpi. This is important when it is required to change password, access network options, interfacing options etc.

4.3.4.4 RPi Camera Setup

After the setup is complete, we will investigate the camera setup. Once the initial configuration was completed, the camera was physically installed into the Raspberry Pi. The camera was activated within the "Interfaces" Tab in the Raspberry Pi configuration utility.

- To plug in the camera, first THE RASPBERRY PI is TURNED OFF, as it is not recommended to attach sensitive peripherals while the device is powered on.
- The black connector next to the 3.5 mm audio jack to insert the camera ribbon cable in lifted up.
- The ribbon cable with the blue side facing the ethernet and USB ports is plugged in.
- The Camera Module ribbon cable is inserted; make sure the cable is the right way.

- The black connector to secure the ribbon cable as shown in Fig. 4.6 is pushed down.
- Start the RPi and Go to RPi configuration.
- In the 'Interfaces' tab, Enable the camera.
- Reboot/Restart the RPi.
- Now that the RPi camera is connected and the software is enabled, we can start using it. For most Raspbian distros the python libraries are already installed. If not, they can be installed by running the following commands:

```
sudo apt-get install python-picamera python3-picamera
```

- Open the terminal window in the RPi.
- To take a picture/video from this camera, type:

```
raspistill -o Desktop/image.jpg #For Image
raspivid -o Desktop/video.h264 #For Video
```

- Press enter to take the picture/video.
- We will see the image and/or video saved on the desktop.

With this, the setup and installation process are complete. For the most part, similar instructions are performed with most of the SBCs. In the next section, a

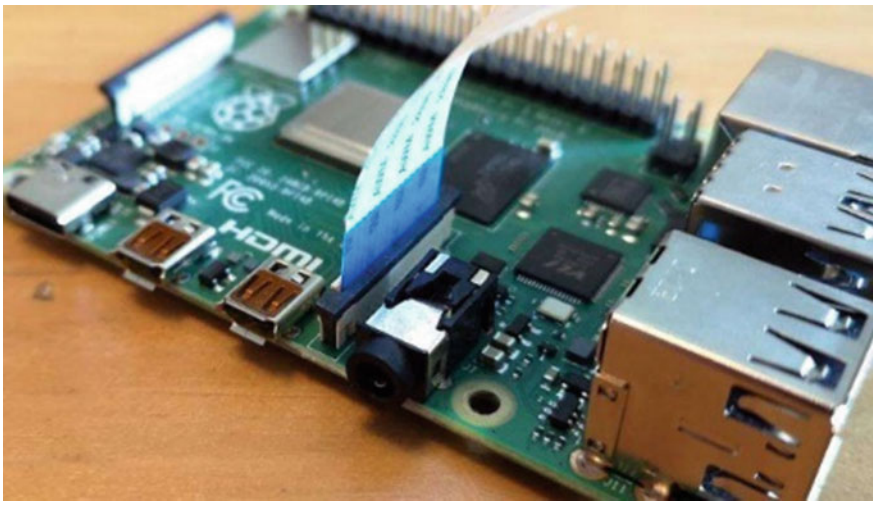


Fig. 4.6 A picture of Raspberry Pi with a camera module attached

detailed guide to programming basics of a microcontroller will be shown using this setup.

4.4 Programming in Python

Machine language, assembly language, and high-level language are the three basic forms of microprocessor programming languages. Programming a microprocessor is technically done in assembly language which provides the set of instructions to the computer (CPU) circuitry to perform tasks in binary format. Hence, each script written in any language is finally converted to binary or hexadecimal format. There are many different languages which can be used to code a microprocessor. Among many, the most important for IoT development are C++ & C, Java, JavaScript, HTML, CSS, Swift, Ruby, Rust and Python. This book covers Python, C++ & C, HTML, and CSS.

Python is a powerful and interpreted high-level general-purpose programming language that was invented in 1991 by Guido van Rossum. Python has become very popular nowadays due to its highly productive usage compared to earlier languages such as C++ and Java. It also succeeds in readability, simple syntax and commands that are close to the natural English language to code in an efficient manner. Python is used mainly for data analytics and machine learning applications apart from running web and internet application scripts. Python is also used by engineers and software developers in major companies like Google, Facebook, and even NASA.

Further, most of the SBCs have Python language usage available, hence it is mostly recommended to start off with learning Python language. The best way to learn a programming language is by understanding syntax and practising simple code lines before moving into complex and extensive codes. For this section, the camera and RPi will be used to touch on the basics of coding using hardware elements.

4.4.1 Fundamentals of Python Programming

4.4.1.1 Installations

Now, before starting to code on the Raspberry Pi in Python language, some of the basics of this language are discussed in this section. This chapter assumes that the reader has some basic knowledge of programming and Linux commands. There are many versions of Python Language since the time of its release in 1991. The most notable are Python 2.7 released in July 2010 and Python 3.9.6 as of 28 June 2021.

- Open the terminal and type the following command to check the version of Python:

```
python --version or
python3 --version
```

The default version installed on the Rpi may be shown here. It may be Python 3.7 or higher. Some older versions might be installed on the board that might be set as default, for e.g., `v2.7.13`. This will need to be changed to Python 3 installation setup as default.

- Enter the following command in the terminal window:

```
nano ~/.bashrc
```

The file '`.bashrc`' is located in the user's home directory (the user pi in this case). The file is a shell script that is executed every time the user opens a terminal (or logs in over SSH, Serial, etc.). It can assist to personalise the user environment, and we will find a few additional commands there already.

- Now, at the end of the last line, enter the command:

```
alias python='/usr/bin/python3'
```

The commands we can use are given below with the appropriate keys. For exiting the nano GNU, press '`ctrl + x`'. Save the file by pressing '`y`' when prompted. Now check the python version again by running '`python--version`' as previously shown. Now, it may show the latest version of Python 3 installed. With that, commands such as pip or pip3 (preferred installer program) will work from the terminal when we install the desktop version of Raspberry Pi OS. 'pip' commands are used to install python packages from the terminal.

4.4.1.2 Python Programming with Interpreter

In the previous section, the python installations, using GNU nano and pip were shown. Now, we will learn about a powerful tool called Python Interpreter which can be used to run Python commands from the terminal. As Python is an interpreted language, it is possible to run codes without the need for a compiler step before output generation. This can be done in the 'interactive mode', which helps run each code line individually. Now, the popular 'Hello, World!' program that is used as a sanity test for the computer's basic working will be shown to run on an RPi.

- To start using the interpreter, open the terminal window and type the following command:

```
python
```

This starts the interpreter, which can be noticed by observing the three ' >>> ' signs on the terminal, also called the prompt. Lines without a prompt are said to be the output of the code line run.

- In the terminal window type the following command, and press enter:

```
print("Hello, World!")
```

So, this was the first Python program we executed! Simple as it sounds, this code line prints a python executed code. The function here is '`print`'. The parameters that are passed to the function are given inside the parenthesis. It is possible to print anything (a character string) if it is inside the quotation marks. Without the quotation marks, Python would interpret it as a variable passing to the function.

To exit the interpreter, type '`exit()`' command.

Now, some more commonly used important functions are shown below that will come handy in any type of application.

- Adding comments in a program is very crucial to understand what each line of code means or is supposed to do. Sometimes when communicating to other programmers, it helps them to understand better what the code is meant to do. Comments start with a hashtag # character followed by a string of characters. The command below shows how this works:

```
# this line is a comment
a = 1  # this also is a comment after in the same code
          # comment can also start with a space given
#print("This is a comment")
print("#This is not a comment ")
comment = "# This is not a comment as its inside quotes"
```

As seen, the lines 1, 2, 3, 4 are comments while line 5 and 6 are not comments even though they have a hashtag.

- Python code can also be used as a calculator to quickly run basic calculations on the interpreter. The following commands shows how this works with numbers. We can use simple operators (`+, -, *, or /`) and parenthesis as well as follows. It follows the BODMAS sequential rule as usual, (Bracket Open, Division, Multiplication, Addition and Subtraction):

```
>>> 5 + 2
7
>>> 5 - 2*2
1
>>> (10 - 2*2) / 2
3.0
>>> 1 / 2
0.5
```

Note that division always gives an output in a floating point number (a decimal value).

- Observe the following numeric calculations of division:

```
>>> 5 / 3  # standard division returns a float value output
1.666666666666667
>>> 5 // 3  # integer result is obtained using floor div
1
>>> 5 % 3  # modulo operator returns the remainder
2
>>> 1 * 3 + 2  # floored quotient * divisor + remainder
5
```

As seen in the example above, different techniques or operators can be used to divide two numbers. The '//' is used for **floor division** which converts the float number into an integer value discarding any value after the decimal point. The modulo '%' operator returns the remainder of the division and as seen, the floored quotient multiplied by the divisor added to the remainder gives the dividend.

- The '**' double Asterix is used to calculate raised to, exponent or power numbers as follows:

```
>>> 3 ** 2  # 3 raised to 2
9
>>> 2 ** 4  # 2 to the power of 4
16
>>> 4 ** 5  # (4)^5
1024
```

- Assigning a variable to a value is done using the ‘ = ’ equal to sign.

```
>>> a = 5
>>> b = 10
>>> c = 2
>>> a + b * c
25
>>> z = a + b + c
>>> z
17
```

Assigning number values to the three variables a, b, and c, allowed the code to recall them to use as a value inside variable in the calculation. Further, these variables can be used to calculate another variable assigned as ‘z’ which returns the value of 17 i.e., a + b + c.

- If a variable, e.g., ‘p’ is called without assigning a value, the system gives an error as follows shown in Fig. 4.7:

In addition to this, Python also supports mixed type operands, decimal, fraction and also has a built-in support for complex numbers, which use the letter ‘j’ or ‘J’ suffix to specify imaginary part e.g., `2 + 3j`. Python language is very vast with immense amount of resources available.

Further, topics such as ‘strings’, ‘lists’, and control flow tools like ‘if’, ‘for’, ‘while’ statements will be studied and explained whenever applicable in the program.

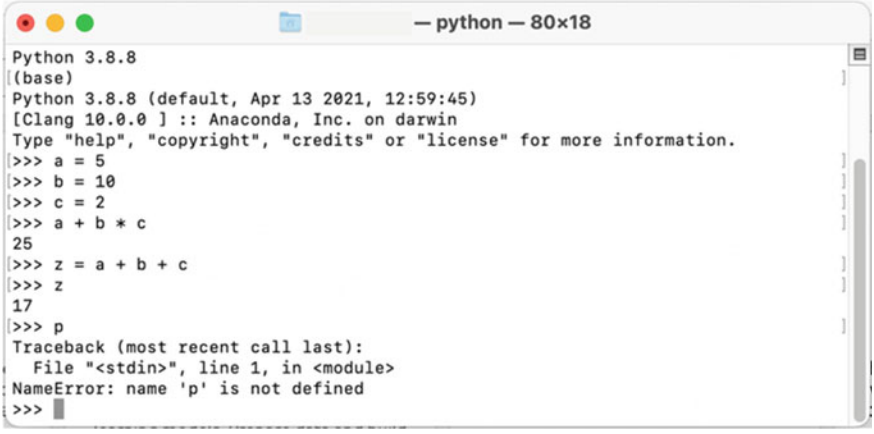


Fig. 4.7 A screenshot of python running on a console displaying the error when a undefined variable is executed

4.4.2 Implementing Python Programming with Camera on IDE

We have performed taking a picture and recording a video with the terminal window. We have also learned the fundamentals of python language and using it via the interpreter. Now, we will use a python program to use the camera functions. The 'picamera' is an inbuilt library which provides multiple functions we can use to program in python code. There are many IDEs (Integrated Development Environment) that can be used to write Python scripts. By default we can use, Thonny, Geany, Mu or IDLE as the IDE. This step used the Thonny Python IDE to access camera functions, rather than the command-line utilities used in the previous step. Thonny was pre-installed on the Raspberry Pi, as was picamera library.

- Open the Thonny Python IDE from the 'Programming' section.
- Create a new file named 'camera.py'
- Enter the following code. Save and run the program. Keep in mind to not save the file name as 'picamera.py' as saving files as existing Python modules causes errors.

```
from picamera import PiCamera # from module import function
from time import sleep #from module import function

camera = PiCamera() #initialise the camera module and its functions
camera.start_preview() #starts the camera preview mode
sleep(5) #sleep for 5 seconds
camera.capture('/home/pi/Desktop/image.jpg')
#capture image and save at the defined location
camera.stop_preview() #stops the camera preview mode
```

- This will run the camera for 5 s and then stop.
- Note: If we do not keep a set time in sleep i.e., minimum of 2, the Pi will freeze.
- Depending on the image, check if the rotation is appropriate. If it needs to be changed, use the following code (You can rotate the image by 0, 90, 180, or 270 degrees) and update the existing code:

```
camera = PiCamera()
camera.rotation = 180
```

- This will take rotate the frame of the image, by the angle we set.

- Update the code to the following: (We can see the image saved on the location given)

```
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

- This will take 1 picture within the 5 s and save it on the file path.
- Now, add a loop statement to take 5 pictures in a row:

```
camera.start_preview()
for i in range(3):
sleep(5)
camera.capture('/home/pi/Desktop/image%s.jpg' % i)
camera.stop_preview()
```

- This will take 1 picture every 5-s, 3 times.
- Now, Update your program to replace 'capture()' with 'start_recording()'.

```
camera.start_preview()
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(5)
camera.stop_recording()
camera.stop_preview()
```

- This will take 1 video for 5-s and save it on the file path.
- It is possible to change image settings and add effects to the images.
- Setting the image resolution (min: 64 $\times$ 64 and max: 2592 $\times$ 1944).

```
camera.resolution = (2592, 1944)
camera.framerate = 15
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/max.jpg')
camera.stop_preview()
```

- This will change the resolution of the picture and save it in the set folder path.

A screenshot shown in Fig. 4.8 is an example of the code running on the Thonny Python IDE RPi-4.

- We can add text to the images by running the following command:

```
camera.start_preview()
camera.annotate_background = Color('green')
camera.annotate_foreground = Color('blue')

 camera.annotate_text = " This is a text "
 sleep(6)
 camera.stop_preview()
```

The following steps were demonstrated in this section:

- Previewing the operation of the camera using the `start_preview` and `stop_preview` functions.
- Rotating an image using the `rotation` property.
- Using a `for` statement to capture consecutive images, and saving them in numbered files.
- Saving a video using the `start_recording` and `stop_recording` functions.
- Setting of the image resolution and framerate properties.
- Commenting each line of code is important to understand the flow and meaning of a code when multiple lines of code are written.
- Importing Modules and functions in a code section is crucial.

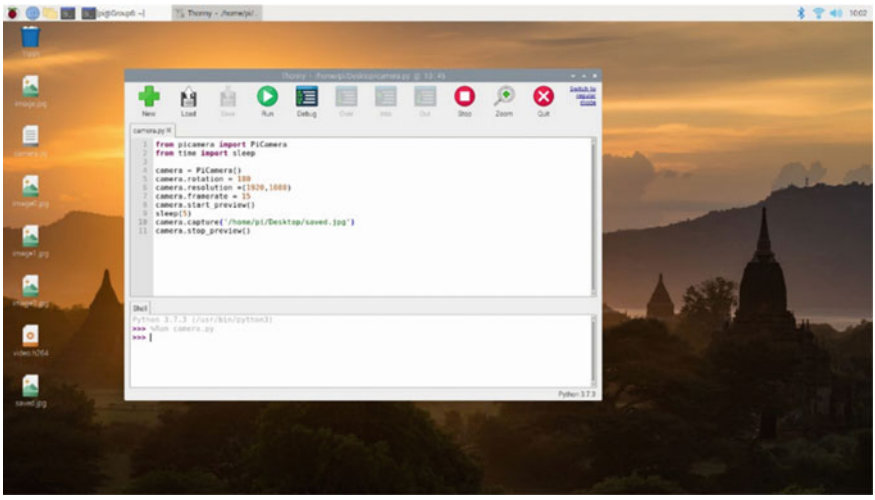


Fig. 4.8 A screenshot of python running an example of the code on the Thonny Python IDE RPi-4

- Each code may have the correct flow of writing with detailed focus on syntax and compiling for any errors.

4.4.3 *Implementing Python Programming with GPIO on IDE*

The chapter has discussed python implementation on the Interpreter using a camera module interfaced with the RPi board. Now, we will look into the working of the General Purpose Input Output (GPIO) pins that are available on the RPi-4.

The Raspberry Pi's row of GPIO (general-purpose input/output) pins along the top edge of the board is a powerful feature. All current Raspberry Pi boards use a 40-pin GPIO header (unpopulated on Pi Zero and Pi Zero W). GPIO pins can be accessed by running the following command on the terminal '`pinout`' and results as follows as shown in Fig. 4.9.

Figure 4.10 shows the pinout of the GPIO on a Raspberry Pi as a handy guide to connect multiple components to it.

The pins are explained as follows:

- Power: The colours **Red** for 5 V, **Orange** for 3.3 V, and **Black** for Ground (0 V) are available on the board as output power. The remaining pins are all 3V3 general purpose pins, which means the outputs are set to 3V3 and the inputs are 3V3-tolerant.
- Outputs: The output pin of a GPIO can be configured to high (3V3) or low (0 V).
- Inputs: The GPIO pins can be configured to input at 3V3 or 0 V (high or low). The use of internal pull-up or pull-down resistors makes this possible. Pull-up resistors are fixed on GPIO2 and GPIO3, although they can be changed in software for other ports.
- ID EEPROM (Electrically Erasable Programmable Read-Only Memory): In **Blue**, the GPIO0 and GPIO1 pins are only used in advanced settings by I2C communication.
- GPIO Pins (**Green**):
  - PWM (Pulse Width Modulation):

    GPIO12, GPIO13, GPIO18, GPIO19: are available with Hardware PWM. PWM is available on all other pins with Software configurations.
  - SPI (Serial Peripheral Interface):

    SPI0:

    - MOSI (Master Out Slave In-data output from master) (GPIO10);
    - MISO (Master In Slave Out-data output from slave) (GPIO9);

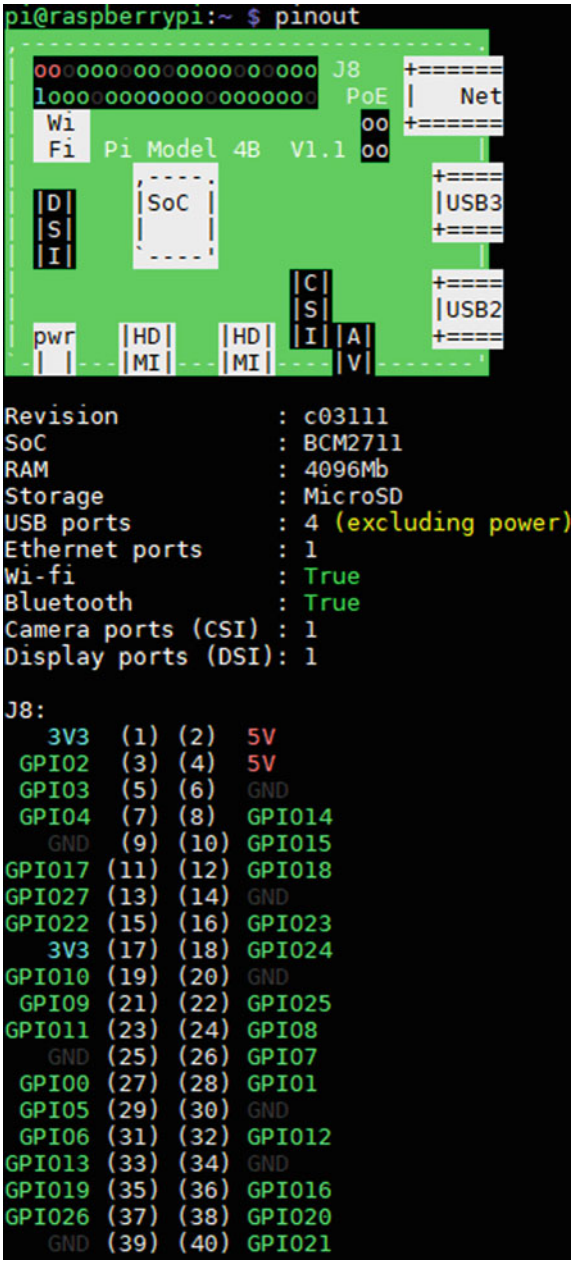


Fig. 4.9 A screenshot of the command 'pinout' displaying the output on RPi-4 terminal

- SCLK (Serial Clock-output from master) (GPIO11);
- CE0 (Chip Enable 0) (GPIO8);
- CE1 (Chip Enable 1) (GPIO7).

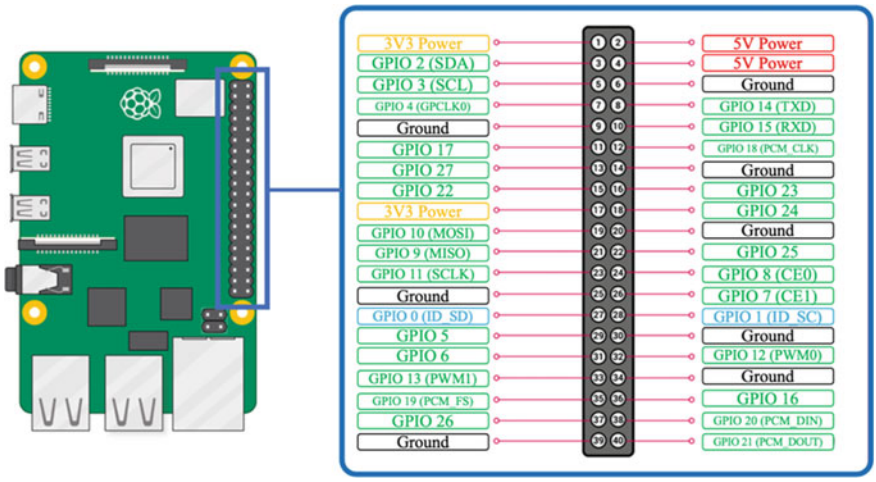


Fig. 4.10 The GPIO pins on the Raspberry Pi Model 4B

Similarly for SPI1:

- MOSI (GPIO20);
- MISO (GPIO19);
- SCLK (GPIO21);
- CE0 (GPIO18);
- CE1 (GPIO17);
- CE2 (GPIO16).

– I2C (Inter-Integrated-Circuit bus):

Data: SDA (Serial Data)(GPIO2); Clock: SCL (Serial Clock) (GPIO3)
EEPROM Data: (GPIO0); EEPROM Clock (GPIO1); SD and SC are Serial Data and Serial Clock

– Serial

TXD (Transmit Data) (GPIO14);
RX (Receive Data) (GPIO15)

Now, on the RPi desktop version, the '`gpiozero`' package is already installed. Using the pip command, install the GPIO Zero package:

```
sudo pip3 install gpiozero
```

GPIO zero has an extensive set of function interfaces to work on.

- Connect an LED to GPIO27 to control its functions as shown in Fig. 4.11.

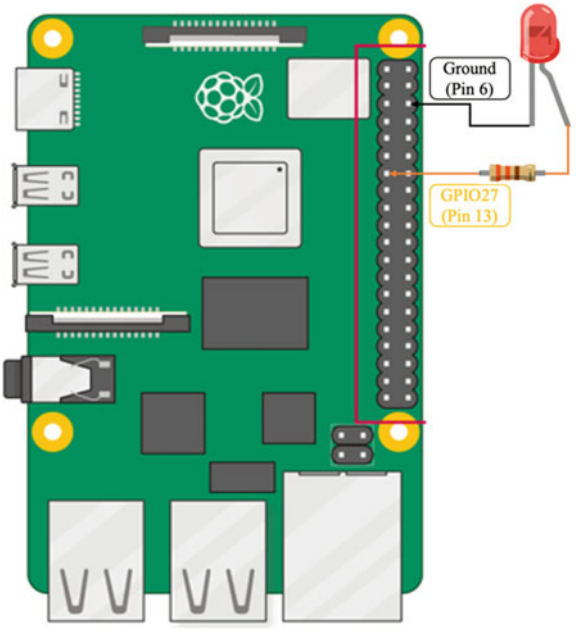


Fig. 4.11 LED to GPIO27 to control its functions on the Raspberry Pi Model 4B

- Run the following code on the Thonny Python IDE:

```
from gpiozero import LED #import LED function
from time import sleep #import sleep function

red = LED(27) #assign gpio27 to a variable

while True: #use a 'while' loop
    red.on() #switch on the led
    sleep(2) #wait for 2 seconds
    red.off()#switch off the led
    sleep(2) #wait for 2 seconds
```

There are many other functions such as `toggle()`, and `blink()` that can be used.

- Similarly, Connect a Button to GPIO2 to control its functions.
- Run the following code on the Thonny Python IDE:

```
from gpiozero import Button

button = Button(2)

while True:
    if button.is_pressed:
        print("Button on")
    else:
        print("Button off")
```

There are many other functions such as `button.when_pressed()`, and `button.when_pressed()` that can be used with similar functionality.

- Further, connect the PIR sensor to the RPi pins as follows, assuming the camera is already connected:
- (PIR) VCC Pin -> 5 V Power
- (PIR) GND Pin -> GND
- (PIR) Data Pin -> Any GPIO (e.g. GPIO 27) (Fig. 4.12)
- Test the PIR sensor working by adding the following code by running the following code on the Thonny Python IDE:

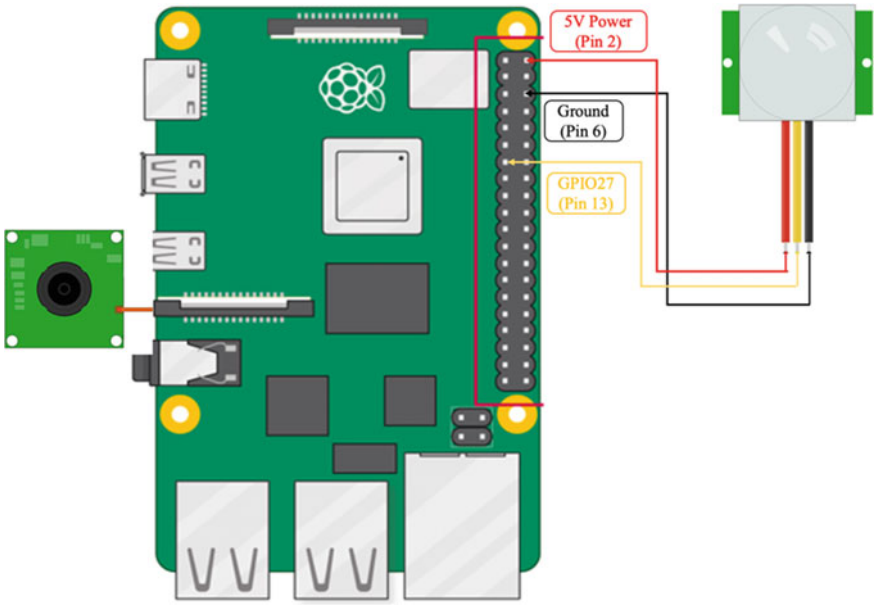


Fig. 4.12 PIR Sensor to GPIO27 and Camera Module interfaced to Raspberry Pi Model 4B

```
from gpiozero import MotionSensor

pir = MotionSensor(27)

pir.wait_for_motion()
print("Motion detected!")
```

Save the file and run the code. Whenever someone passes through the PIR Sensor's range of Infrared Area Sensing, the PIR Sensor LED will glow and the code will output 'Motion Detected!'.

- Run the next code given below and comment on the difference.

```
from gpiozero import MotionSensor

pir = MotionSensor(27)

while True:
  pir.wait_for_motion()
  print("Motion detected!")
```

- **Task 1**: Video with timestamp
- Now, as the camera is already interfaced, execute the following instructions given below:
  - Modify the code to take a photograph with the camera if a person is detected.
  - Modify the code to take a video with the camera if a person is detected.
  - Use a variable to save the video file into the RPi system if a person is detected. E.g. `variable_name="video_name.h264"`.
  - Creating Timestamps with Python: Use the '`datetime`' module in python to create a timestamp.
  - Modify the code with '`from datetime import datetime`'.
  - Assign the given code to your variable to save your video file with a timestamp. '`"{0:%Y}-{0:%m}{0:%d}".format(datetime.now())`'
  - Play the video and verify the working of the system.
- **Task 2**: Intruder detection System:
- Now that you have learnt how to use a PIR Sensor and Camera, make an intruder detection IoT system.

- You have been provided 2 PIR sensors for this task. The first part of the task is to determine the direction of motion of the person.
- For this system, assume the scenario of a University attempting social distancing due to COVID19 at the cafeteria where space is limited. They would like to ensure different doors for entry and exit to the store. Tasks assessed for the report will be:
  - There needs to be continuous video surveillance once PIR motions triggered (set a sleep time of 5–10 s, to avoid freezing the Raspberry pi).
  - And, using the algorithm for direction of person, you will trigger an image capture in case there has been an exit from the entry door (For instance if the direction of entry is Left to Right, the image will be triggered, when the direction is Right to left i.e. if someone exits instead of entering).
  - Comment on the working and installation of this IoT system in an actual practical setting.
  - Mention the drawbacks and challenges in making this system.

4.5 Outcome of Student Implementation

The following section shows the students' implementation of the tasks provided.

- **Task 1**: Video with timestamp

We then connect the RPi camera to the RPi with the PIR sensor. Using the picamera library, we use the RPi camera as an output whenever we detect motion in the PIR sensor. The three codes show different settings on how the camera will be utilized (Fig. 4.13).

In the first code, the RPi camera captures an image when a motion is detected in the PIR sensor and saves it in '/home/pi/Desktop/PIR_camera.jpg'.

On the second code, the RPi camera records a three (3) second video when motion is detected in the PIR and saves it in '/home/pi/Desktop/PIR_video.h264'.

The third code also saves a three (3) second video when motion is detected. The filename at which the video is saved is written in the format 'PIR_video<date_today>.h264', where <date_today> is in the format YYYY-MM-DD (e.g. 2021-03-20).

- **Task 2**: Intruder detection System:

After verifying the working of each of these devices, we design and implement a simple intruder detection system using two PIR motion sensors and a picamera on a Raspberry Pi. This system identifies the direction of motion of the person and if the person moves in the forward direction (Left to Right) a 5-s video is taken and if an intruder is detected (a person moving from Right to Left), an image is captured (Fig. 4.14).

motion_pir.py | camera.py | pir_camera.py *

```
from gpiozero import MotionSensor
from picamera import PiCamera
from time import sleep
camera = PiCamera()
pir = MotionSensor(4)
pir.wait_for_motion()
print("Motion detected!")
camera.start_preview()
sleep(3)
camera.capture('/home/pi/Desktop/PIR_camera.jpg')
camera.stop_preview()
```

Shell

```
KeyboardInterrupt
>>> %Run pir_camera.py
 Motion detected!
>>> %Run pir_camera.py
 Motion detected!
>>>
```

motion_pir.py | camera.py | pir_camera.py

```
from gpiozero import MotionSensor
from picamera import PiCamera
from time import sleep
camera = PiCamera()
pir = MotionSensor(4)
pir.wait_for_motion()
print("Motion detected!")
camera.start_preview()
camera.start_recording('/home/pi/Desktop/PIR_video.h264')
sleep(3)
camera.stop_recording
camera.stop_preview()
```

Shell

```
Motion detected!
>>> %Run pir_camera.py
 Motion detected!
>>> %Run pir_camera.py
 Motion detected!
>>>
```

pir_camera.py

```
from gpiozero import MotionSensor
from picamera import PiCamera
from time import sleep
from datetime import datetime
camera = PiCamera()
pir = MotionSensor(4)
pir.wait_for_motion()
print("Motion detected!")
camera.start_preview()
file1 = "PIR_video"
file2 = "{0:%Y}-{0:%m}-{0:%d}".format(datetime.now())
file3 = ".h264"
camera.start_recording(file1 + file2 + file3)
sleep(3)
camera.stop_recording
camera.stop_preview()
```

Shell

```
Python 3.7.3 (/usr/bin/python3)
>>>
```

Fig. 4.13 Python code for RPi camera with PIR (Task 1)

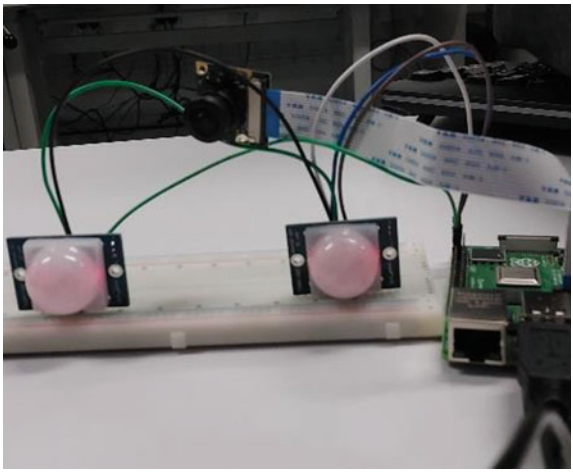


Fig. 4.14 Circuit diagram showing sensor setup

```
1 from picamera import PiCamera # camera library
2 from gpiozero import MotionSensor # PIR sensor
3 from datetime import datetime # datetime formatting lib
4 from time import sleep # library for delays
5
6 _prev_l = 0 # Previous direction Left
7 _prev_r = 0 # Previous direction Right
8 _dir = "" # Direction of motion
9
10 def getDirection():
11 global _prev_l
12 global _prev_r
13 global _dir
14 l = pirL.value # current state of LEFT PIR
15 r = pirR.value # current state of RIGHT PIR
16 if not l and not r: # No motion
17 _dir = ""
18 elif l and not _prev_l and _dir == "": # First motion
detected on RIGHT PIR
19 _dir = "Right"
20 elif r and not _prev_r and _dir == "": # First motion
detected on LEFT PIR
21 _dir = "Left"
22 _prev_l = l # Store previous state for comparison
23 _prev_r = r # Store previous state for comparison
24 return _dir # Return direction of motion detected
25
26 camera = PiCamera() # Initialise RPi camera
27 camera.resolution = (640, 480) # Reduce resolution
28 camera.rotation = 180 # Rotate camera 180 degrees
29 pirL = MotionSensor(20) # Sets LEFT PIR to pin 20 on RPi
30 pirR = MotionSensor(21) # Sets RIGHT PIR to pin on RPi
31
32 while True: # Main loop and capture image, sleep to
avoid freeze, capture video
33 if getDirection()== "Left":
34 camera.capture("/home/pi/Desktop/Intruder_im-
age%s.jpeg" %"{0:%Y}-{0:%m}-{0:%d}- {0:%S}".for-
mat(datetime. now()))
35 sleep(2)
36 camera.start_recording("/home/pi/Desktop/In-
truder_video%s.h264" %"{0:%Y}-{0:%m}-{0:%d}-{0:%S}".for-
mat(datetime.now()))
37 sleep(5)
38 camera.stop_recording()
```

Suggested Reading

1. S.C. Mukhopadhyay, N.K. Suryadevara, Internet of Things: challenges and opportunities. In: S. Mukhopadhyay (ed.), *Internet of Things. Smart Sensors, Measurement and Instrumentation*, vol. 9 (Springer, Cham, 2014) https://doi.org/10.1007/978-3-319-04223-7_1
2. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
3. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Sensing Technologies for intelligent environments: a review. In: H. Leung, S. Chandra Mukhopadhyay (eds), *Intelligent Environmental Sensing. Smart Sensors, Measurement and Instrumentation*, vol. 13 (Springer, Cham, 2015) https://doi.org/10.1007/978-3-319-12892-4_1
4. M.E.E. Alahi, S.C. Mukhopadhyay, Smart nitrate sensors, internet of things enabled real time water quality monitoring. Smart Sens. Meas. Instrum. **35** (2019). ISBN: 978-3-030-20094-7
5. [EB.7] S.C. Mukhopadhyay, A. Lay-Ekuakille (eds.), *Lecture Notes in Electrical Engineering, Advances in Biomedical Sensing, Measurements, Instrumentation and Systems* (Springer, 2010). ISSN: 1876-1100, ISBN: 978-3-642-05166-1
6. S.C. Mukhopadhyay, H. Leung (eds.), *Lecture Notes in Electrical Engineering, Advances in Wireless Sensors and Sensors Networks* (Springer, 2010). ISSN: 1876-1100, ISBN: 978-3-642-12706-9
7. S.C. Mukhopadhyay (ed.), *Smart Sensors, Measurement and Instrumentation, Vol. 9, Internet of Things: Challenges and Opportunities* (Springer, 2014). ISBN: 978-3-319-04222-0
8. S. Mukhopadhyay, T. Islam (eds.), *Innovative Technologies and Services for Smart Cities, Special issues in MDPI Electronics* (2019). www.mdpi.com/journal/electronics. ISBN: 978-3-03921-181-4
9. O.A. Postolache, E. Sazonov, S.C. Mukhopadhyay (eds.), *Sensors in the Age of the Internet of Things: Technologies and Applications* (IET Book, 2019). https://digital-library.theiet.org/content/books/ce/pbce122e. ISBN: 9781785616341, e-ISBN: 9781785616358
10. A.K. Sangaiah, S.C. Mukhopadhyay (eds.), *Intelligent IOT Systems in Personalized Health Care* (2021). ISBN: 978-0-12-821187-8
11. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement **178**, 109424 (2021)
12. [J.194] F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
13. D. Thomas, R. Shankaran, M. Orgun, S. Mukhopadhyay, A secure barrier coverage scheduling framework for WSN-based IoT applications. *IEEE Transactions on Green Communications and Networking* (2021)
14. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the Internet of Things and serious games. Internet Things **13**, 100332 (2021)
15. S.C. Mukhopadhyay, S.K.S. Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
16. S. Pal, M. Hitchens, T. Rabehaja, S. Mukhopadhyay, Security requirements for the internet of things: a systematic approach. MDPI Sens. 5897 (2020)
17. T. Islam, S.C. Mukhopadhyay, Linearization of the sensors characteristics: a review 12(1) (2019). https://www.exeley.com/articles/1/32. https://doi.org/10.21307/ijssis-2019-007
18. F. Akhter, S. Khadivizand, H.R. Siddiquei, M.E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
19. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using Graphite/PDMS sensors. Sens. Actuators, A **286**, 43–50 (2019)

20. A. Seth, A. James, S.C. Mukhopadhyay, 1/10th scale autonomous vehicle based on convolutional neural network. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–17 (2020)
21. M.E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417 (2018)
22. N.K. Suryadevara, S.C. Mukhopadhyay, Smart homes: design, implementation and issues. Smart Sens. Meas. Instrum. **14** (2015)
23. R. Yan, X. Chen, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 26, Structural Health Monitoring: An Advanced Signal Processing Perspective* (2017). ISBN: 978-3-319-56125-7
24. S.C. Mukhopadhyay, A. Mason (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 4, Smart Sensors for Real-Time Water Quality Monitoring* (2013). ISBN: 978-3-642-37005-2-1
25. S.C. Mukhopadhyay (ed.), *Lecture Notes in Electrical Engineering Vol. 96, New Developments in Sensing Technology for Structural Health Monitoring* (Springer, 2011). ISBN: 978-3-642-21098-3
26. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015), www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350
27. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045

Chapter 5
Programming Arduino for IoT System

5.1 Introduction

In an IoT system, a programmable circuit board is used to enable interfacing sensors and transmission modules to communicate among each other as well as controlling all operations. This technology works beyond connectivity of internet in typical devices such as phones, desktops, tablets etc., it qualifies everyday objects such as home appliances, work environment etc. to not only communicate using the internet, but also to be remotely supervised and controlled. This programmable circuit board is often referred to as a microcontroller. Figure 5.1 illustrates different microcontroller (μC) that are used in IoT systems based on the application and availability of the transmission medium.

There are various hardware platforms available in the IoT architecture. However, each provide slightly varying solutions based on the cost, software, or a complete end to end solutions. In this chapter, the focus will be on the IoT products of Arduino. 'Arduino' enables a simple platform to utilise the functions of the microcontroller, providing a low cost solution. The μC used in Arduino is the ATmega328 that empowers the system to have optimised power consumption along with the processing speed.

A μC is a fundamental unit of an IoT based system. Additionally, this chapter will discuss the various programming approaches for such microcontrollers.

The major focus will be on the three boards of Arduino, i.e., the UNO, Nano 33 IoT and MKR1300. The Arduino software platform uses a simplified type of C++, that can be easily learnt, programmed, and loaded on the board.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_5

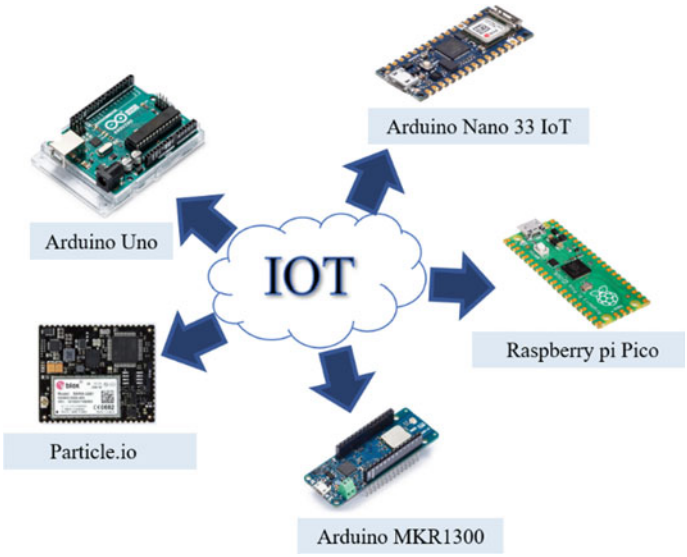


Fig. 5.1 The various promising Hardware platforms available for IoT systems

5.2 Understanding the Microcontroller

This chapter discusses and offers a comparison between the three Arduino boards used throughout the practical work conducted, i.e., Arduino UNO, Nano 33 IoT and MKR1300. The microcontroller used in Arduino UNO is based on ATMega328P 16 MHz that is a low powered, high performance microcontroller. Figure 5.2 illustrates the pin configuration of the microcontroller to the input/output pins of Arduino. There are 14 digital input/output pins (six of which can be used as PWM outputs), six analogue inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button on the board. It comes with everything that is required to get started with the microcontroller; simply plug it into a computer with a USB cable or power it with an AC-to-DC adapter or battery.

Additionally, the ATMega328P also enables 1 KB EEPROM that stores memory that is not erased while the Arduino is powered off. This will store the program uploaded to the Arduino and store it when it is powered off unless a new program has to be uploaded. Also, in the Arduino UNO board, the ATMega328P chip is not soldered it can be replaced in case it gets damaged.

In the case of Arduino MKR1300 and Nano 33 IoT the microcontroller used is the SAM D21 Cortex 32-BIT low power ARM MCU that has been shown further in this section. Furthermore, a detailed comparison has been illustrated in Table 5.1. This comparison aids in selection of the appropriate design for a defined IoT system.

The Nano 33 IoT is often chosen when wireless connectivity is necessary, it is surprising that it exceeds the Arduino Uno in general-purpose applications that do

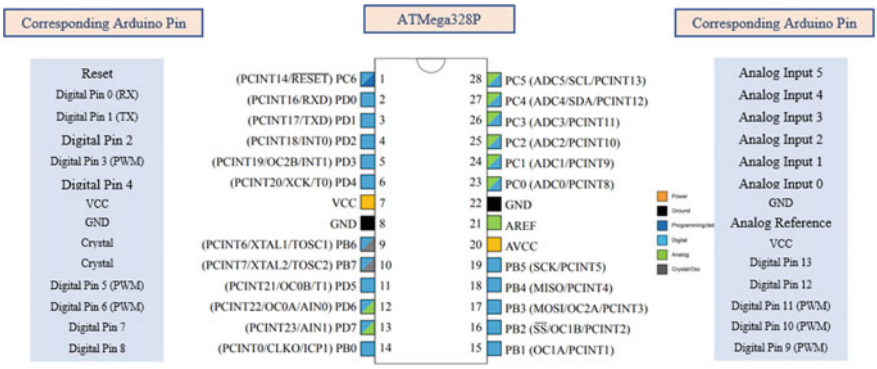


Fig. 5.2 The Pin configuration of the µC ATMega328P that corresponds to the Arduino UNO

Table 5.1 Comparison among the Arduino boards used in this chapter

Features	Boards		
	Arduino Nano 33 IoT	Arduino MKR WAN 1300	Arduino Uno Rev 3
Microcontroller	SAM D21 Cortex 32-BIT low power ARM MCU	SAM D21 Cortex 32-BIT low power ARM MCU	ATMega328P 16 MHz
WiFi/ Bluetooth/ LoRa	Nina W102 uBlox module for WiFi and BLE connectivity	Murata CMWX1ZZABZ for LoRa connectivity	Not available
Digital Input/ output	14	8	14
PWM	5	13	6
External interrupts	Every digital and analog pin	10 pins	2 pins
Analog In (ADC)	8 in (8/10/12-bit ADC)	7 in (8/10/12-bit ADC)	6 (10-bit ADC)
Analog out (DAC)	1 out (10-bit DAC)	1 out (10-bit DAC)	Not available
UART/SPI/I^2C	1/1/1	1/1/1	1/1/1
Cryptography	(ECC608 chip)	(ECC608 chip)	Not available
Additional peripherals	6-axis IMU (LSM6DS3 sensor)	Not available	Not available
Form factor	45 mm (W) x 18 mm (H)	68 mm (W) × 25 mm (H)	69 mm (W) × 53 mm (H)
Operating voltage	3.3 V	3.3 V	5 V
Flash/SRAM	256 KB/32 KB	256 KB/32 KB	32 KB/2 KB
Price (in AuD)	36$ (AuD)	~65$ (AuD)	39$ (AuD)

not require wireless connectivity, especially when cost is a factor. However, for transmission over a wide area, LoRa, i.e., the MKR1300 board will be the most favourable choice. The MKR1300 uses the independent Murata CMWX1ZZABZ to connect to a private or public LoRaWAN network.

The Nano 33 IoT board uses u-blox NINA-W102 that is a line of stand-alone multi-radio MCU modules that combine a powerful microcontroller (MCU) with a wireless communication radio. Wi-Fi and Bluetooth v4.2 communications are supported by the radio. The Nano 33 IoT and MKR1300 have the same security feature using the ATECC508 chip that keeps data private and secure using the same μC i.e. the SAM D21 Cortex-M0. This is a powerful processor with 256 KB of CPU flash memory, and it consumes less power. Figure 5.3 shows the pin configurations of SAM D21 Cortex-M0 corresponding to the Nano 33 IoT board.

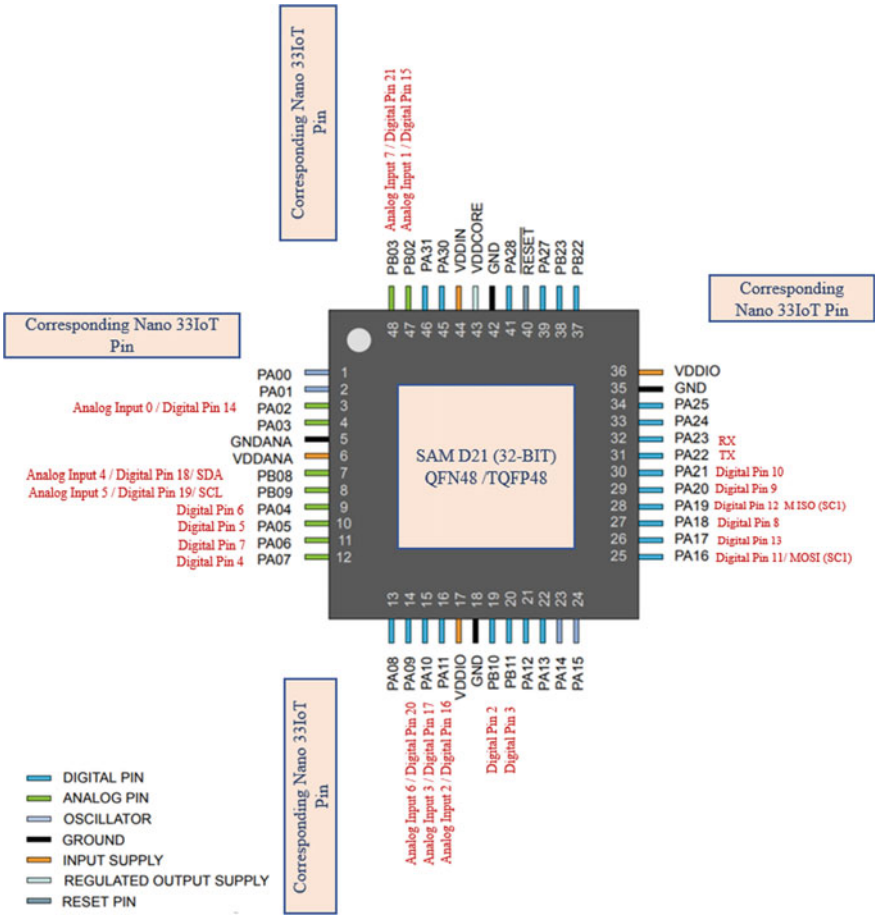


Fig. 5.3 The Pin configuration of the μC SAM D21 that corresponds to the Arduino Nano 33 IoT

An important aspect to note is that MKR1300 and the Nano 33 IoT board operate at 3.3 V, which implies that all the Digital and Analog pins to be restricted to 3.3 V to avoid damage.

In summary, a microcontroller is a little computer built onto a single integrated circuit or computer chip. The μCs are a very useful tool for programming and controlling electronic devices. A microcontroller chip, as well as other essential connectors and components, are included on microcontroller boards, allow users to connect inputs and outputs. Using the Arduino software one can program the μC to perform various tasks, which will be elaborated in the next section.

5.3 Function Definitions and Configurations

In the previous sections, a summary was given on the types of microcontrollers used in different Arduino boards Although, in this section, the basics of programming will be introduced along with their various functions.

5.3.1 Objectives

The major aim of this section is to provide a broader understanding on the μC used in Arduino and how to program it. The points further elaborate on these aims:

- Learning how to program the board using the given software platforms available for the Arduino board.
- Understanding the in-built functions and experimenting with different applications
- Developing and designing basic C++ libraries.
- Understanding the pinouts available to interface sensors and calibrate them.
- Illustrating a sample guideline sheet to implement at a workshop.

5.3.2 Hardware Used

The hardware components used in this chapter primarily involve the μC boards illustrated earlier in this chapter and the sensors interfaced. The following components are required to build this system:

- Microcontroller:
  - Arduino UNO Rev 3
  - Arduino Nano 33 IoT
  - Arduino MKR1300

- Micro USB cable:
  - Arduino MKR WAN 1300 compatible
  - Arduino Nano 33 IoT compatible
- USB cable A-B: Arduino UNO Rev3 compatible
- Sensors:
  - PIR Sensor Module-Motion Sensing
  - LEDs
- Prototyping Solderless Breadboard
- Jumper Cables
- Desktop/Laptop

5.3.3 Software/Applications

The following software can be used to program the boards:

- Arduino IDE: (Compatible with Arduino UNO Rev 3, Nano 33 IoT, MKR1300)
- Arduino CLI: (Compatible with Arduino UNO Rev 3, Nano 33 IoT, MKR1300)
- Web Editor: (Compatible with Arduino UNO Rev 3, Nano 33 IoT, MKR1300)
- IoT Cloud: (Compatible with Nano 33 IoT, MKR1300)

5.3.4 Basic Code Structure

To understand and implement basic coding in this chapter Arduino UNO Rev 3 will be implemented. To commence programming in this section, the Integrated Development Environment (IDE) has been used. This software can be installed in the PC or laptop from here: https://www.arduino.cc/en/software.

To power and program, the Arduino UNO Rev 3 board, it is connected to the PC/Laptop using the USB A/B cable. Once connected, the board will draw power and the green LED labelled as PWR will light up. To detect the board using the connected USB COM port (communication port), the board drivers need to be installed. This can be performed under Control Panel > System and Security > Device Manager. Under the port's section there will be listed "Arduino UNO (COMxx)", right click on this, and update the driver software. Once installed, the Arduino board can now be detected in the IDE. This can be found under Tools > Port > COMxx (Arduino/Genuino Uno).

The Arduino software is accessible as open-source tools for experienced programmers to extend. People who want to learn more about the language can use C++ libraries, and those who want to learn more about the technical specifics can

switch from Arduino to the AVR C programming language. Similarly, if required, one can include AVR-C code directly in the Arduino applications.

The Arduino Programming Language, or Arduino Language, is a native language supported by Arduino. This program is based on the Wiring development platform, which is based on Processing, which is what p5.js is based on i.e., a JavaScript for creative coding. The Arduino IDE is based on the Processing IDE, which in turn is based on the Wiring IDE.

While working with Arduino, the Arduino IDE as seen in Fig. 5.4 is used frequently, a piece of software available for all major desktop platforms (macOS, Linux, Windows) that provides with two features: a programming editor with built-in libraries, and a way to quickly compile and load our Arduino programs onto a computer-connected board.

The Arduino Programming Language is essentially a C++-based framework. Sketch is the name for a program created in the Arduino Programming Language. Normally, a sketch is stored with the '.ino' extension (from Arduino). The most significant distinction from "normal" C or C++ is that there are two primary functions in the code. There can be more than two, however any Arduino application may at least supply those two.

The first is setup(), and the second is loop (). The first is called once when the program starts, and the second is called frequently while it is running. The main() function, which is used as the program's starting point in C/C++, will not occur in this case. After the sketch is compiled, the IDE checks to see if the product is a valid C++ program, and then pre-processes it to glue it together.

5.3.5 Examples

There are various default examples available in the Arduino IDE. To understand the functions of the Arduino UNO board, one of the most common examples used by beginners is the LED Blink example.

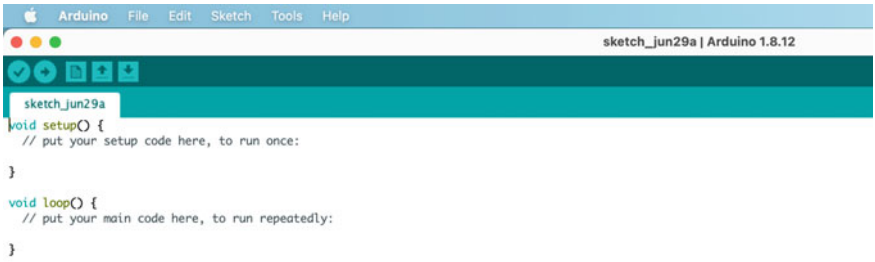


Fig. 5.4 The screenshot of the Arduino IDE for a new sketch

- Implementing Blink Example without inbuilt LED

The LED Blink example is designed for the built-in LED on the Arduino UNO board. However, this example, the LED is connected to a digital output pin i.e., Digital Pin 7.

The modification in the code will be as follows:

```
int LedPin = 7;
void setup() {
  pinMode(LedPin, OUTPUT);
}
void loop() {
  digitalWrite(LedPin, HIGH);
  delay(1000);
  digitalWrite(LedPin, LOW);
  delay(1000);
  }
```

The circuit diagram for this example is illustrated in Fig. 5.5.

5.4 Interfacing Sensors

This section illustrates the connection diagrams and interfacing pins for the sensors assigned for a project. In Fig. 5.6, the board used to interface and run this system is the Arduino UNO Rev3. In the system, two LEDs are used as indicators to determine the motion of a person across an area. For instance, if the person moves from Left to Right, then the Left PIR will detect first simultaneously glowing the corresponding Left LED and the then the Right PIR will detect, while the Right LED glows. 330 Ω resistors were used to avoid any excess current that may flow through the LEDs.

The sensors and components used were all digital, they are interfaced to the available Digital pins on the board. The left pin on the PIR sensors is connected to 5 V and the right pins are connected to a common Ground on the Arduino. It is beneficial to have the program driven by interrupts instead of looping, the PIR sensors are connected to Digital pins 2 and 3 respectively. The ATMega328P offers only two interrupt pins in Arduino UNO board.

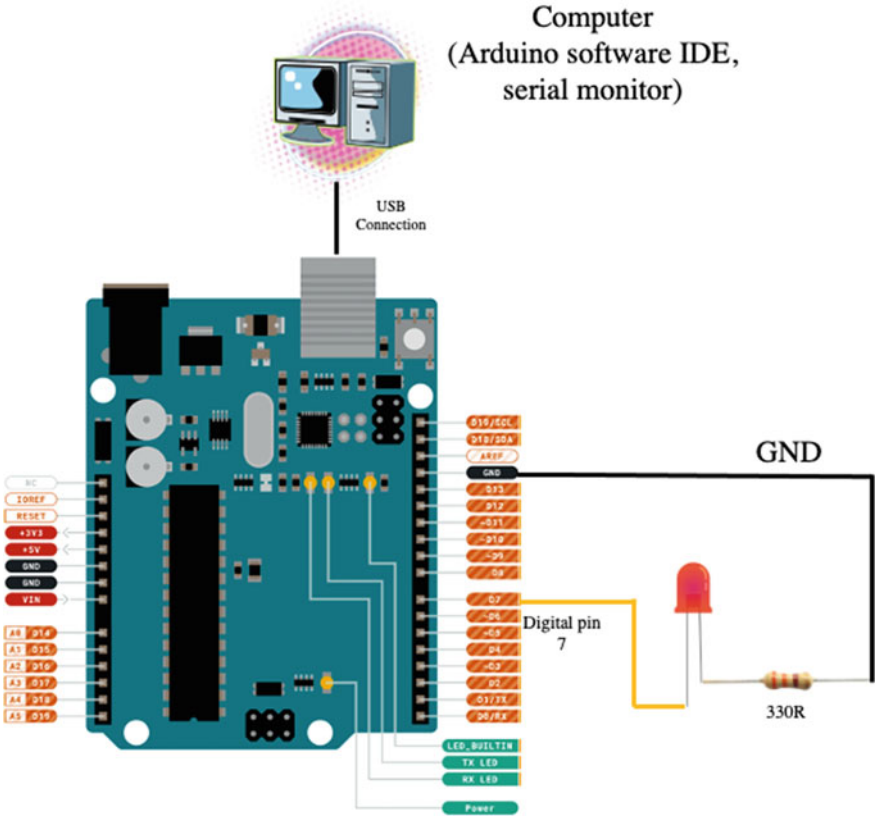


Fig. 5.5 The circuit diagram of the Arduino IDE for example sketch using a simple LED

5.5 Library Setup and Configurations

The 'Libraries' in Arduino are collections of code that make it simple to connect to sensors, displays, modules, and other devices. The built-in LiquidCrystal library, for example, makes it simple to communicate with character LCD panels. Many more additional libraries are accessible for download from the Internet. It need to be installed to use the additional libraries.

The basics of the programming are introduced along with the available built-in operations. In this section, a brief introduction is provided to library setups and configurations.

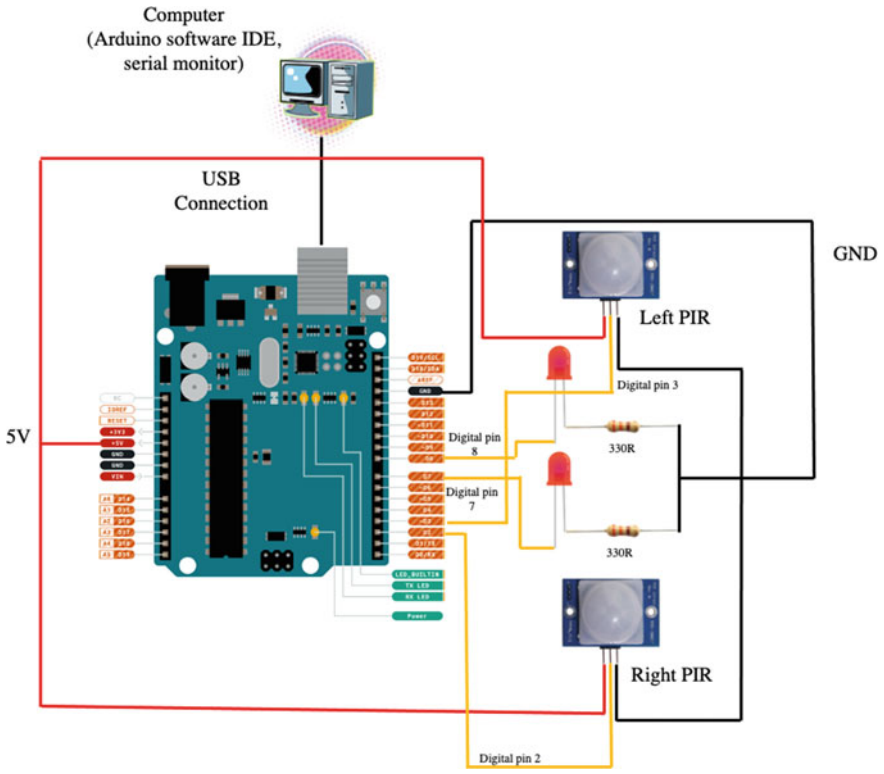


Fig. 5.6 The circuit diagram for the two PIR pedestrian direction tracking system

5.5.1 Understanding the Library and Its Installation

Library Manager is an Arduino development software feature that makes it simple to search, install, and update both official and third-party libraries. Every unique sensor or module or functions are configured with a library in Arduino by the manufacturers and/or hobbyists. This makes it easy to use it for an IoT applications. To use new uninstalled library, the user may go to Sketch > Include Library > Manage Libraries on the IDE, as seen in Fig. 5.7.

Once clicked the Library Manager will then launch, displaying a list of libraries that are either installed or ready to be installed. To find any specific library file scroll through the list or type it, click it, and then choose the library version to be installed.

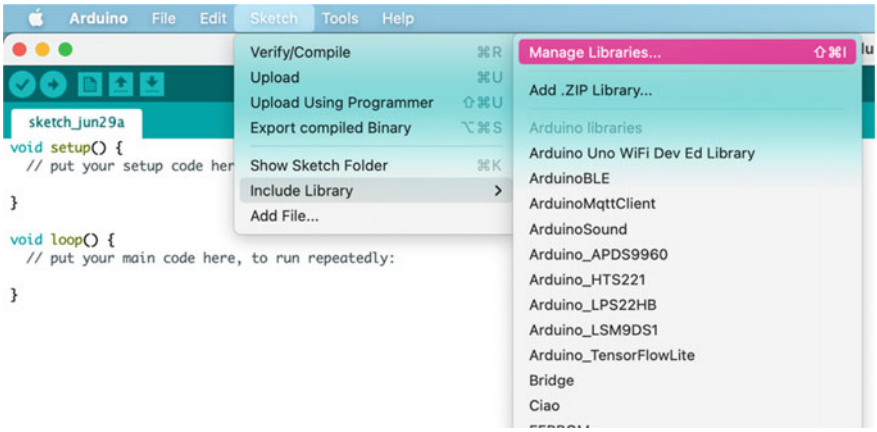


Fig. 5.7 The screenshot of the IDE that shows the path to the Library Manager

5.5.2 Writing Our Own Library

In many scenarios, the library folders may not be available in the library manager. In such cases, the third parties will configure it as a zip file or folder. The library's name is the name of the folder. A.cpp file, a.h file, and typically a keywords.txt file, examples folder, and other files required by the library will be found inside the folder.

This folder can be added by 'Sketch > Include Library > Add.ZIP Library…' as also seen in Fig. 5.7. This library will now be available for use, and it can be seen in the list of libraries. On some occasions there are many libraries available to accomplish the same task. An important thing to note while working with multiple libraries are the libraries for Arduino are stored in three locations: the IDE installation folder, the core folder, and the libraries folder within the sketchbook. The manner libraries are chosen during compilation is intended to allow the distribution's libraries to be updated. This means that adding a library to the sketchbook's "libraries" folder replaces any previous versions of libraries.

To configure our own library, the user may go through the given detailed guidelines to follow as listed on this website https://github.com/arduino/library-registry. This website provides with a list of instructions that will guide the user step by step to write and launch their own library.

An overview of the basic structure has been provided in Sect. 5.6, wherein, the user will understand how to program and design an API for the custom Arduino library.

5.5.3 Header File/Source File

This section elaborates on the two main files which are to be included in the custom library. These files are a header file (.h extension) and a source file (.cpp extension).

The header file's core consists of a line for each library function, wrapped in a class with any variables required. There is no return type, and the constructor has the same name as the class.

A class is nothing more than a collection of functions and variables kept in one place. These functions and variables can be public, which means that anybody who uses the custom library can access them, or private, which means that it can only be accessed from within the class. A constructor is a particular function in each class that is used to generate an instance of the class. Figure 5.8 shows the source folder for the ThingSpeak library used in Chap. 2, as it can be seen, there is a source and header file.

Once this library is added, an `#include` statement will be added for every header in this library's folder.

For the source file, there are a few #include statements. These provide access to the basic Arduino functions as well as the definitions in the header file to the remainder of the code. After that, there's the constructor. The user chooses the pin they want to use in this situation. This pin is then configured as an output that saves it as a private variable that can potentially be used for other functions.

5.5.4 Example Sketch and Final Library Folder

Once compiled, the header and source file are added to a directory with the 'class name' that is listed. In this directory, it is imperative, that both these files are saved under these extensions i.e. '.cpp' or '.h' and no other alternate extensions.

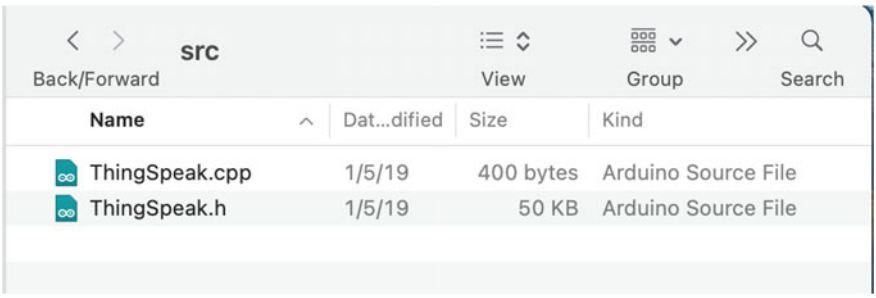


Fig. 5.8 The screenshot of header and source file in the ThingSpeak library source folder

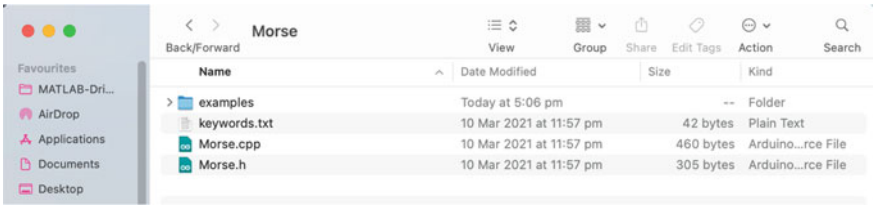


Fig. 5.9 The screenshot of custom Library folder contents

Also, it is essential to add an example to enable the users to understand how the library functions. This code will be saved under '.pde' extension and stored inside a folder labelled 'examples' as seen in Fig. 5.9.

Once this file is compiled and saved it, the user may add the library in the Arduino IDE Library Manager. Finally, the user will be able to see the file under the custom library under the library name that has the example file name. Figure 5.10 shows the example file listed and explained under Sect. 5.6.1.

5.6 Guidelines to Implementation of Project Activity

This section of the chapter discusses the guidelines provided to the students to implement the laboratory work based on library configurations and basic coding. This laboratory instruction manual explains only the necessary instructions required to complete the project while giving a basic example in detail and then putting forth the challenge for the class by interfacing sensors and creating a library. Thus, giving students the opportunity to learn by doing their research and practising their skill to write and understand the code for the header files.

In this laboratory activity, the students will be learning how to write a library in Arduino, for a sensor interface in an IoT based application. For sensors that are commercially available, the libraries have been written, by the manufacturing companies or a hobbyist. However, if any person makes a new sensor and it needs to be calibrated, thus, it is imperative to know how to write a library to avoid the main sketch to be complicated.

First, the students will begin by writing a library for the LED blink. Once completed the student will write a library for PIR sensors.

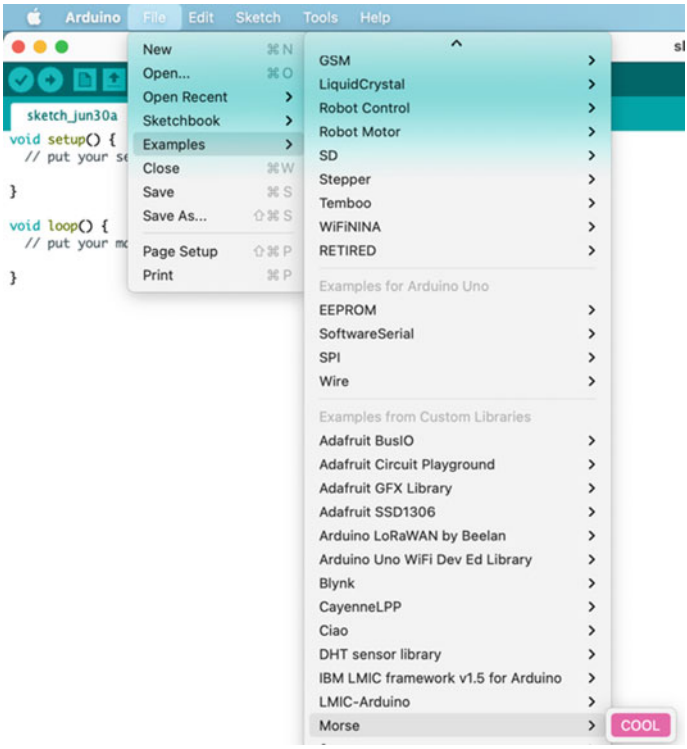


Fig. 5.10 The screenshot of Arduino IDE with the example sketch in the custom folder

5.6.1 Arduino Library for LED Blink in Morse Code

In task 1, the students will learn how to create a library for Arduino. This is done in 5 simple stages; the students may use the same logic while creating the files for the sensor interface.

The main logic of the code is to say 'COOL' using MORSE CODE by blinking the LED! 'COOL' in morse is '-.-. – –.-..'

1. Build a simple Sketch.

Before starting to write the library files, it is always easier to begin with the sketch itself. So first implement this sketch that will blink the inbuilt LED in morse code spelling 'COOL'.

```
int pin = 13;
void setup()
{
  pinMode(pin, OUTPUT);
}

void loop()
{
  dash(); dot(); dash(); dot();
  dash(); dash(); dash();
  dash(); dash(); dash();
  dot(); dash(); dot(); dot(); delay(3000);
}

void dot()
{
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
  delay(250);
}

void dash()
{
  digitalWrite(pin, HIGH); delay(1000);
  digitalWrite(pin, LOW); delay(250);
}
```

Here, dot(); and dash(); are defined as two separate functions. And the loop will run for every three seconds on the Built-in pin 13 of the microcontroller.

2. Understanding a Library

After the students run this sketch and confirm the morse code. The students can now divide the codes into separate files for the library. The primary purpose of the library is to build a resource platform that will simplify a task or process.

Create a folder named 'Morse'. In this folder the students will now save the files as the students create them in the further steps.

A typical library folder will contain:

- A header file ('.h'). // library definitions
- A source code ('.cpp') // library's code
- A keyword file ('.txt') // keywords used in the library
- A Readme file ('.txt') // information of the library, (for instance any revisions etc.)

- Example files (‘.ino’). // example codes implementing the library functions.

All these files mentioned above, may be stored in one directory/folder named with the library. This can then be added to the IDE, using the ‘Add Zip File’ function.

3. Writing the header file

The students have now understood the files that are necessary in a library, the students can now begin with the first file of the library i.e., the header file. Create a folder in the PC named as ‘Morse’ and begin saving the files as we go in this folder.

The file the students write will be a C++ source file. The students can write the file using ‘Notepad’ and save it as a ‘.h’ file.

The header file consists of the definitions for the library. For this example, the library will be called as “Morse”. Thus, the header file will be named “Morse.h”.

The header file for this example will be:

```
/*
  Morse.h - Library for flashing Morse code. Created
  for ENGG4201/8201
*/
#ifndef Morse_h
#define Morse_h
#include "Arduino.h"
class Morse
{
  public:
  Morse(int pin); void dot(); void dash();
  private:
  int _pin;  //_pin can be named as any variable, the
  //underscore before it implies it is a private var-
  iable //as defined in the header.
};

#endif
```

The header file consists of a line for each function in the library, that is wrapped in a class along with the necessary variables. The #include will give the students access to the standard types and constants of the Arduino language.

4. Writing the source file

After the students write the header file for the Morse, the students now proceed to the source code for the library which is also a ‘C++’ file. The file begins with #include statements that provide access to the header files and Arduino functions. The students have done earlier, use ‘Notepad’ to write and save the code with ‘.cpp’ extension in the ‘Morse’ folder.

Next instances for the class are created, where the user will specify the pin configuration. The output pin is configured as a private variable in this source file to be used in various functions.

```
/*
  Morse.cpp - Library for flashing Morse code. Cre-
  ated for ENGG4201/8201
*/
#include "Arduino.h" #include "Morse.h"

Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin; //_pin can be named as any variable,
  //the underscore before it implies it is a private
  //variable as defined in the header.

}

void Morse::dot()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

In the code, 'Morse::' this implies that the function is part of the Morse class. To provide a variable to the user, to assign a pin number, the 'set_pin' variable is provided.

The keyword file for this example will be:

```
Morse KEYWORD1
dash  KEYWORD2
dot KEYWORD2
```

This file will be stored as a '.txt' file. This file provides information where the IDE understands which words are important and may be highlighted.

The readme file which is also a text file provides a guide to the user who is new to the library and may know what it is used for, and what are the latest updates.

5. Modifying the sketch using the new library

Since, all major files are completed. Now the next stage is to build an example to use the header and source files implementing the functionality.

Create an examples folder in the existing 'Morse' folder the students have made and write the Arduino sketch implementing the library functions as shown below.

Notice, the use of the keywords specified in the library in the modified sketch. The dot(); and dash(); functions are now prefixed with 'morse'.

```
#include <Morse.h>
Morse morse(13);
void setup()
{
}
void loop()
{
  morse.dash();morse.dot();morse.dash();morse.dot();
  morse.dash();morse.dash();morse.dash();
  morse.dash();morse.dash();morse.dash();
  morse.dot();morse.dash();morse.dot(); morse.dot();
  delay(3000);
}
```

As the sketch is now modified, the students save this in the examples folder as a 'Cool.ino' file. Now the final folder may look like this:

To manually add this to Arduino, the students copy the above in 'Morse' folder and paste the folder in the Arduino > libraries directory. The storage address will be similar to C:\Users\Documents\Arduino\libraries\Morse.

Alternatively, the students can Save this folder and compress it into a zip folder. Now add this zip file to the IDE, go to sketch > include library > add. .zip library > choose 'Morse.zip' and click ADD!

Once added, the students will now find the example code in the IDE under File > Examples > Morse > Cool. Click on it and upload it to the Arduino to verify the Morse code.

Once completed, the students may proceed with the next task!

5.6.2 Building a Unique Library to Configure PIR Sensors

In this task, the students will design the library to determine the direction of pedestrian motion using two PIR sensors.

For this task, the students will use two PIR sensors and two LEDs. Depending on which PIR detects first, the simultaneously LEDs will glow (for instance if the right PIR glows first and then left, the right LED will blink first and then left).

The main logic of the code will be to determine which direction the pedestrian moves i.e., left to right or right to left.

The direction of motion to be printed on the serial monitor. Once completed, the students may proceed with the next task!

5.7 Outcome of the Student Implementation

This section aims to illustrate the work done by the students while implementing the task 5.6.2.

Figure 5.11 shows the hardware circuit along with the implementation of the project. Some decisions were made in deciding upon a good architecture for the sketch:

- Encapsulate all the direction-sensing logic within a class, so that the user's sketch (the consumer of the library), did not need to be aware of the details of the logic used. Additionally, if that logic was improved, modification of any sketch that used it will not be required.
- Provide support for multiple instances of the PIR Direction Sensor class. If two objects are instantiated, a four-sensor configuration that can detect two separate movements simultaneously may be possible without introducing complexity for the user of the library.
- Provide a simple interface. Ideally, the library may be as easy to use as the DHT library was in Lab instructions for week 1.

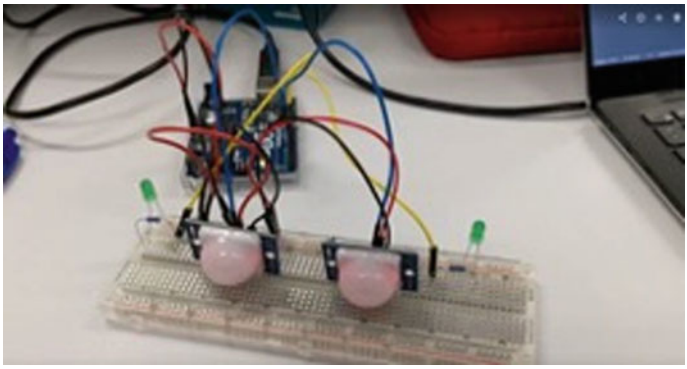


Fig. 5.11 The image of the circuit implemented by the students in the lab

- Provide flexibility to the user of the library to respond to the events generated by the library in a customisable way. Callback functions were desired, with behaviour customised within the sketch rather than hard coded into the library; and
- Avoid having the library to "take control" of the user's sketch. Ideally, to preserve the interrupt-based design developed in the previous sketch, and keep the loop() of the sketch empty to let the user include whatever additional functionality was desired.

There were two different approaches to completing the library. An approach based on interrupts is presented in the following section.

5.7.1 Interrupt-Based Library Design

5.7.1.1 Limitations of the Interrupt-Based Design

The major issue that was encountered with the migration of the sketch to a library was that it was not possible to use an instance method (non-static method) of a class as an interrupt handler with the attachInterrupt() function. Attempting to do so resulted in the following error message:

```
     C:\Users\pm_au\Documents\Arduino\librar-
     ies\PIR_Library\pir.cpp:15:71: error: invalid use of
     non-static       member       function       'void
     PIR::testInterrupt()'
  attachInterrupt(digitalPinToInterrupt(lPir),  testInter-
rupt, CHANGE);
```

As the error message suggests, the solution was to use a static method within the class rather than an instance (or non-static) method.

The static method can only use static fields within the class because there is no "current object" (or this reference) associated with a call to a static method. There is also no other mechanism to associate metadata with the interrupt, such as ID. The consequence of this limitation is that most of the data (pin numbers and state) also will be static, which leads to a design where only a single instance of the class can be instantiated by a user of the library. This means that only a single "PIR Direction Sensor" instance comprising two actual PIR sensors can be accommodated.

5.7.1.2 Design of Class

The code fragment below shows the structure of the class:

```
class PIR {
public:
PIR(byte lPir, byte rPir, byte lLed, byte rLed, void
(*cback)(int));
// return codes for callback
  static const int LEFT_TO_RIGHT = 1;
  static const int RIGHT_TO_LEFT = 2;
private:
  static void _changeLeft();
  static void _changeRight();
  static byte _pirL;
  static byte _pirR;
  static byte _ledL;
  static byte _ledR;
  static volatile byte _state; // variable for state
  machine
// callback to main sketch
  static void (*_callback)(int);
// states
  static const byte _NONE_SET = 0;
  static const byte _LEFT_SET = 1;
  static const byte _RIGHT_SET = 2;
};
```

The class provides a very simple interface to the user. A single constructor is provided, taking arguments for the pin numbers for the LED and PIR sensors. The only other public data are the codes for LEFT_TO_RIGHT and RIGHT_TO_LEFT, which need to be processed by the call-back function in the user's sketch.

All other data is private, including the interrupt handlers `_changeRight()` and `_changeLeft()`, the pin numbers for sensors and LEDs, the _state variable and its constants, and a pointer to the user's callback function `(*_callback)(int)`. The user's sketch has no direct access to private data.

The `_changeLeft()` and `_changeRight()` functions behave identically to their counterparts in the sketch, except they reference private static variables in the class. Also, rather than output to the Serial Monitor directly, they delegate that task to the callback function defined in the sketch. This makes the library more flexible, as most users would want to customise this function to do something more useful with the data, rather than simply display it. A fragment of the `_changeLeft()` function, manipulating the class' _state variable and performing the callback to the user's sketch, is shown below:

```
void PIR::_changeLeft() {
…
if (PIR::_state==PIR::_RIGHT_SET)
{
// detected right to left movement, call the sketch
_callback(PIR::RIGHT_TO_LEFT);

PIR::_state=PIR::_NONE_SET;
}
…
}
```

5.7.1.3 The Example Sketch

The benefit of the interrupt-based design is the simplicity of the interface provided to the user. There are only two steps necessary to complete in the sketch, as shown in the library's example sketch.

The first step is to define the callback function which processes the LEFT_TO_RIGHT and RIGHT_TO_LEFT events from the library. In the example, these events are output to the Serial Monitor:

```
void movement(int direction)
    {
if (direction == PIR::LEFT_TO_RIGHT)
{
Serial.println("Detected LEFT to RIGHT movement");
}
else if (direction == PIR::RIGHT_TO_LEFT)

{
Serial.println("Detected RIGHT to LEFT movement");
} }
```

The second step is to instantiate an object of the library, passing the pin numbers for the sensors and LEDs as well as the name of the callback function defined above:

```
PIR pir
(LEFT_PIR, RIGHT_PIR, LEFT_LED, RIGHT_LED, movement);
```

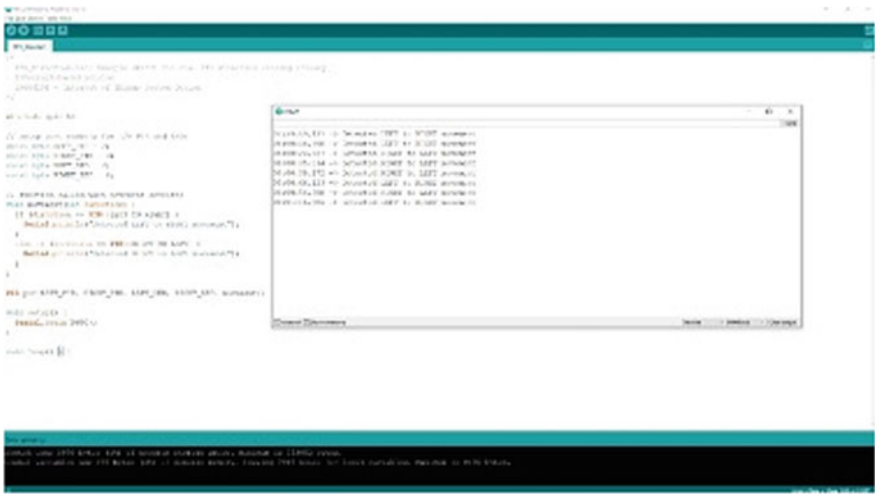


Fig. 5.12 The screenshot of the result of the custom library designed by the student

Note that the sketch's loop() function is completely empty, and can be used to perform any other function desired by the user.

The example sketch added to the library is shown in Fig. 5.12 below, along with its Serial output.

Suggested Reading

1. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
2. S.C. Mukhopadhyay (ed.), *Wearable Electronics Sensors: For Safe and Healthy Living*, vol. 15 (Springer, 2015)
3. N.K. Suryadevara, S.C. Mukhopadhyay, *Smart Homes* (Springer, Berlin, 2015)
4. S.C. Mukhopadhyay, *Intelligent Sensing, Instrumentation and Measurements*, vol. 5 (Springer, Berlin, 2013)
5. S.C. Mukhopadhyay, P.J. Krishanthi, O.A. Postolache (eds.), *Modern Sensing Technologies*, vol. 29 (Springer, 2018)
6. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement **178**, 109424 (2021)
7. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
8. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the Internet of Things and serious games. Internet Things **13**, 100332 (2021)

9. S.C. Mukhopadhyay, S.K.S. Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
10. T. Islam, S.C. Mukhopadhyay, Linearization of the sensors characteristics: a review **12**(1) (2019). https://www.exeley.com/articles/1/32. https://doi.org/10.21307/ijssis-2019-007
11. F. Akhter, S. Khadivizand, H.R. Siddiquei, M.E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
12. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using graphite/PDMS sensors. Sens. Actuators, A **286**, 43–50 (2019)
13. M.E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417 (2018)
14. N. Afsarimanesh, M.E.E. Alahi, S.C. Mukhopadhyay, M. Kruger, Development of IoT-based impedometric biosensor for point-of-care monitoring of bone loss. IEEE J. Emerg. Sel. Topics Circuits Syst. **8**(2), 211–220 (2018)
15. R. Morello, C. De Capua, G. Fulco, S.C. Mukhopadhyay, A smart power meter to monitor energy flow in smart grids: the role of advanced sensing and IoT in the electric grid of the future. IEEE Sens. J. **17**(23), 7828–7837 (2017)
16. M.E.E. Alahi, L. Xie, S. Mukhopadhyay, L. Burkitt, A temperature compensated smart nitrate-sensor for agricultural industry. IEEE Trans. Ind. Electron. **64**(9), 7333–7341 (2017)
17. S. Pirbhulal, H. Zhang, M.E.E. Alahi, H. Ghayvat, S.C. Mukhopadhyay, Y.-T. Zhang, W. Wu, A novel secure IoT-based smart home automation system using a wireless sensor network. Sensors **17**, 69 (2017). https://doi.org/10.3390/s17010069
18. T. Islam, S.C. Mukhopadhyay, N.K. Suryadevara, Smart sensors and internet of things: a postgraduate paper. IEEE Sens. J. **17**(3), 577–584 (2017)
19. S. Mukhopadhyay, Advanced sensors and instrumentation: a project based paper with emphasis on remote environmental parameters measurement. IEEE Instrum. Measur. Mag. 68–73 (2016)
20. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045
21. N.K. Suryadevara, S.C. Mukhopadhyay, Smart homes: design, implementation and issues. Smart Sens. Meas. Instrum. **14** (2015). 978-3-319-13556-4
22. R. Yan, X. Chen, S.C. Mukhopadhyay (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 26, Structural Health Monitoring: An Advanced Signal Processing Perspective*. (Springer, 2017). ISBN: 978-3-319-56125-7
23. S.C. Mukhopadhyay, A. Mason (eds.), *Smart Sensors, Measurement and Instrumentation, Vol. 4, Smart Sensors for Real-Time Water Quality Monitoring* (Springer, 2013). ISBN: 978-3-642-37005-2-1
24. S.C. Mukhopadhyay, *Lecture Notes in Electrical Engineering Vol. 96, New Developments in Sensing Technology for Structural Health Monitoring* (Springer, 2011). ISBN: 978-3-642-21098-3
25. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350

Chapter 6
WiFi Based IoT System

6.1 Introduction

IoT system applications needs a wide range of connectivity requirements in terms of range, data throughput, energy efficiency, and device cost. Small, intermittent payloads, such as utility meters, to increasing amounts of continuous data, such as real-time video surveillance, are all examples of IoT data transfer requirements.

Wireless fidelity is abbreviated as Wi-Fi. Wi-Fi is a wireless networking technology that enables computers, mobile phones, iPads, game consoles, and other devices to communicate across a wireless network. WiFi (IEEE802.11) is a wireless technology that was created with the goal of replacing Ethernet with unlicensed wireless communication. Its purpose was to provide off-the-shelf, easy-to-install, and easy-to-use short-range wireless communication that was cross-vendor compatible as shown in Fig. 6.1.

WiFi is typically not a sustainable choice for large networks of battery-operated IoT sensors, especially in industrial IoT and smart building applications, due to its high energy consumption. Instead, it refers to connecting devices that can be easily plugged into a power outlet, such as smart home appliances, digital signage, and security cameras. WiFi 6 will be the enterprise's preferred wireless access technology, with 5G primarily reserved for outdoor use with significant overlap in indoor applications as seen in Fig. 6.1. WiFi 6—the latest WiFi generation—adds significantly more network bandwidth to increase data throughput per user in heavily populated areas. The standard is expected to improve public WiFi infrastructure and revolutionize customer experiences in the retail and mass entertainment industries with new digital mobile services.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_6

Fig. 6.1 The transition for WiFi 6 as compared to 5G for IoT applications as illustrated by Cisco in 2020

6.2 Description

The objectives and components description for the project is elaborated in this section.

6.2.1 Objectives

The major aim of this project is to give the users a firsthand experience of building a complete IoT system. The following points describe the objectives in detail:

- Developing a complete working IoT system based on WiFi in a short amount of time of about 3–4 h.
- Interfacing sensors and realising their function.
- Give users an idea of the microcontroller/microprocessor hardware functionality in building the system.
- Learning about the WiFi transmission protocol used.
- Sending the data wirelessly to the cloud for visualisation and analytics.

6.2.2 Hardware Used in the Project

The hardware components used in this chapter mainly involve the sensors and microcontroller/microprocessor unit. The following components are required to build this system:

- Microcontroller: Arduino Nano 33 IoT
- Micro USB cable: Arduino Nano 33 IoT compatible
- Microprocessor: Raspberry Pi Model 4B
- Cable for Raspberry pi: HDMI Cable, USB A to C
- Sensors and Components
- PIR Sensor Module—Motion Sensing
- HC-SR04 Ultrasonic Module—Distance Measuring Sensor
- Raspberry pi Fish-eye wide angle camera
- LEDs
- Pushbutton
- Prototyping Solderless Breadboard.

6.2.3 Software/Applications Used in the Project

The following are the software/applications used in this project:

- Microcontroller Programming: Arduino IDE
- Web Framework for Raspberry WiFi connectivity: Flask
- Could data analytics and visualisation: ThingSpeak.

6.3 Motivation

WiFi plays a critical role in data transmission for home and industry applications. The most popular WiFi standard in homes and many companies right now is IEEE802.11n, which provides serious throughput in the hundreds of megabits per second, which is good for file transfers but may be more energy consumption for many IoT applications. Therefore, in the IoT domain, the technology's main limitations in coverage, scalability, and power consumption preventing it from being widely adopted.

In this chapter, the user will use WiFi to transmit data and control the interfaced peripherals, these ideas can be transferred to any project chosen to implement in WiFi applications.

6.4 WiFi Functionality on Raspberry Pi

Wireless connectivity is an important step in building any IoT system. IoT devices need different communication protocols to transmit/receive data. Enabling WiFi functionality on a SBC is the first task to accessing the boards internet connectivity.

Most of the SBCs have in-built WiFi chips that help the device to connect to the internet. The most recent technology of WiFi that these boards use is based on the IEEE 802.11 family of wireless network protocol standards.

Raspberry Pi Model 4B uses the WiFi standard IEEE 802.11 b/g/n/ac dual band 2.4/5 GHz. These standards allow the Wireless Local Area Network (WLAN) connectivity functionalities for the board. The 'b/g/n/ac' stand for different versions of the WiFi protocol and have different modulation techniques and data rates. It supports two frequency of operations i.e. 2.4 and 5 GHz operation. Most of the Raspberry Pi boards that have built-in support for WiFi accessibility, have the BCM43455/6 chip. The BCM43455 incorporates advanced enhanced collaborative coexistence hardware methods and algorithms to ensure that WLAN and Bluetooth collaboration is designed for maximum performance. To check the exact wireless chip from the Raspberry Pi, go to terminal and type the following command:

```
sudo journalctl -b | grep brcmfmac
```

This command shows the output with the chip BCM43455/6 depending on the model of the RPi one uses. Figure 6.2 shows the Functional Block Diagram of BCM43455 from Technical Datasheet.

WLAN operating in IEEE 802.11ac mode provides MCS0–MCS9 (up to 256 QAM) rates in 20, 40, and 80 MHz channels for data speeds of up to 433.3 Mbps. The IEEE 802.11a/b/g/n standard specifies all rates that are supported. 2.4 and 5 GHz transmit amplifiers, as well as receive low-noise amplifiers, are included in the chip. External PAs and LNAs are also supported as options. The BCM43455 is

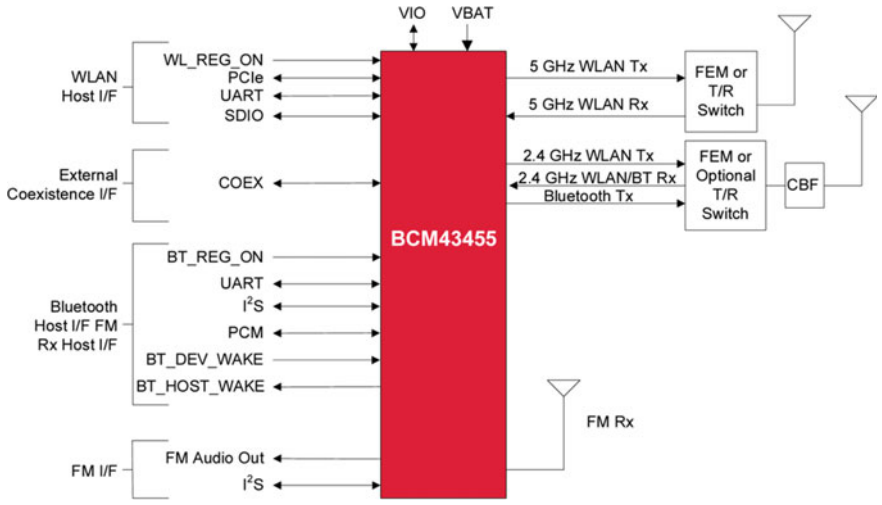


Fig. 6.2 Functional block diagram of BCM43455 from technical datasheet

developed to suit the needs of mobile devices that require minimal power consumption and compact dimensions by utilising innovative design methodologies and process technology to reduce active and idle power.

6.5 Setup and Installations of WiFi on Raspberry Pi

In the last section, a deep insight to the WiFi chip and function was discussed briefly. Further, this section explains how to enable and use WiFi with internet on the RPi.

- Ethernet for Internet

Firstly, before understanding the wireless connectivity setup, the easiest and fastest way to use internet is through the Gigabit Ethernet port. A LAN cable is required to connect it to the desired network router and RPi's port with a ethernet or RJ45 cable. Once plugged in, the network LED on the RPi will start blinking. Theoretically, it supports uplink/downlink speeds up to 1 GB/s. Practical benchmarking has shown uplink/downlink speeds up to 786.26/837.22 Mbps. The network settings of the device can be checked using the following command:

```
sudo apt-get install net-tools
```

```
ifconfig eth0
```

The command gives the output as shown in Fig. 6.3.

It is seen, the IP address of users device is shown after '`inet addr`' and will be something like 192.168.1.120. A device on the internet or a local network is identified by its IP address, which is a unique address. The Internet Protocol (IP) is a collection of rules that regulate the format of data transferred over the internet or a local network. The IP address numbers are from 0.0.0.0 to 255.255.255.255.

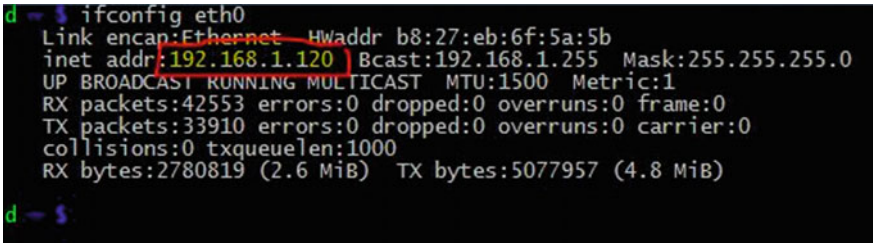


Fig. 6.3 Screenshot of running ifconfig eth0 command on terminal

Some other commands that one can use from the terminal to directly show the IP address are:

```
ip addr show
```

or

```
ip a s
```

or

```
hostname --ip-address
```

These are the methods to see the network IP address of the RPi with ethernet enabled. To check the correct operation of the network, use 'ping' command e.g.;

```
sudo ping  www.raspberrypi.org
```

This returns with data packets send/receive that will confirm the internet access.

- WiFi with Imager (For SSH)

The simple way to enable WiFi settings on the RPi whether one is using the desktop/headless version is shown as follows:

While installing the OS on the SD card as seen in Sect. 4.3.4, Press Ctrl + Shift + x to open the advanced menu (CMD + Shift + x for Mac OSX). In the drop down menu, the user can setup the WiFi credentials according to the SSID and password of the network. Save the settings and flash the micro SD card. This allows the system to connect to the RPi network via SSH (Secure Shell).

- WiFi with Desktop

One of the direct ways to connect to the WiFi on the RPi is to connect to the network from the desktop GUI (Graphical User Interface) version of the RPi OS. First, is the normal way to connect to a Windows/Mac PC, on the top right corner, clicking and enabling WiFi and adding the appropriate password to the SSID would enable it to use internet functions as shown in Fig. 6.4.

6.6 Guidelines for Project Implementation

In the earlier section the methods of enabling network WiFi for the RPi to access the internet were shown. In this section, an IoT project using WiFi communication will be discussed. This will allow the user to understand how WiFi technology can

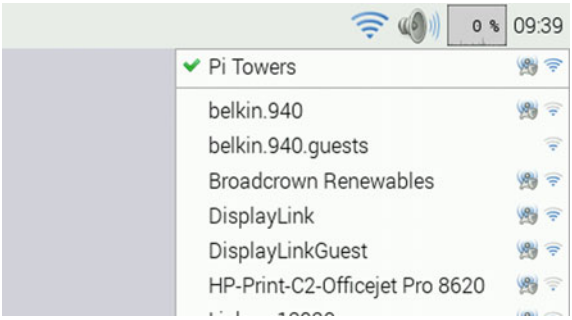


Fig. 6.4 Screenshot of WiFi on desktop version of Raspberry Pi OS

be used to send and receive data from an RPi IoT node. This section will elaborate the example codes and guidelines given to users in the lab work.

In this task, we will learn how to control different sensors and peripherals connected to a RPi using another Web based controller via WiFi communication. We can also use this to see the sensor data on a website. The web framework called 'Flask' will be used in this section.

6.6.1 Software Installations and Setup

Start by connecting the RPi as a desktop. Go to the terminal and type the following commands one after the other and press enter:

```
sudo apt install python3 idle3 #install python v3 idle
sudo pip3 install guizero #install guizero library
sudo apt-get install python3-flask
cd Documents #enter command directory 'Documents'
mkdir lab_WiFi #make a new folder inside this
cd lab_WiFi #enter command directory 'lab_WiFi'
mkdir static #make a new folder inside this
mkdir templates #make a new folder inside this
```

With these steps, the necessary packages have been installed required for the lab. The directory tree is as follows:

```
> Documents
  > lab_WiFi
    > static
    > templates
```

Flask is a Python-based microweb framework. It is referred to as a microframework because it does not necessitate the usage of any specific tools or libraries. Flask is known to be easier to understand for beginners and is not as complex and advanced as another similar web framework called 'Django'. Both Flask and Django have many open source materials and forums for understanding and debugging. Flask provides a modest collection of straightforward APIs, as well as comprehensive documentation. We start the web development with Flask as a beginner to Python, to acquire a feel for both the backend and frontend, as well as grasp the essential ideas.

The library '`guizero`' which is a Python 3 toolkit for making minimal graphical user interfaces. It is intended to help inexperienced programmers rapidly and simply construct graphical user interfaces for their programmes.

6.6.2 *The `guizero` Library*

Before starting to create a web framework on Flask, this section will elaborate and show some basic examples to get started on the fundamentals of the '`guizero`' library. Open the Thonny Python app on the RPi to save and run the codes.

- To create an app window titled 'New Window' type the following, save and run the code:

```
from guizero import App #import the function

app = App(title="New Window") #use the app function to
create a new title

app.display() #display the output
```

To add text widget command with a button to a New Window type the following:

```
from guizero import App, Text, PushButton #import the function

def change_message(): #this function is called each time
    message.value = "Button was pressed! #button pressed

app = App(title=”New Window”) #use the app function to create a new title

message = Text(app, text="This is the New App Window") #use the text widget function to display

button = PushButton(app, text="Button Here", command=change_message) #

app.display() #display the output
```

6.6.3 Python Webserver Application Using Flask on RPi

In this section the Flask web framework to build an IoT web application has been used. The Geany Python IDE has been used. Geany is an Integrated Development Environment (IDE) that is compact and lightweight. It was created with the goal of creating a minimalistic and efficient IDE with few dependencies on other libraries. The Geany app allows users to write code in HTML, CSS and Python.

- Create a new file and save it as: “`test.py`”. Save the file into the `lab_WiFi` folder. The file will contain the following code:

```
from flask import Flask #Import the Flask class

app = Flask(__name__) #Create a Flask object ‘app’

@app.route('/') #use route() decorator function
def index(): #the ‘/’ will direct to root URL
#return the text value to the client’s web browser
  return “<p>Test Successful<p>”

#The hostname is ‘0.0.0.0’ and 80 is the port number
if __name__ == '__main__':
  app.run(debug=True,port=80,host='0.0.0.0')
```

In the python file, a class is created that includes a run function inside. The parameters inside the run function include the host and the port number, which indicates the address at which the Website can be opened. The hostname is '0.0.0.0' and 80 is the port number. To open this, use chromium app and set the address to "http://127.0.0.1:80/". The "127.0.0.1" is the home address and the port number is indicated by the ":80". This code looks as shown in Fig. 6.5. To get the output as seen in the Fig. 6.5 follow the given below steps:

- Run the command: "`ifconfig`" or "`hostname -I`" to see the RPi IP address.
- It may be something like: "`192.168.149.121`" depending on the RPi's WiFi connection to the mobile hotspot/WiFi provided. It will be next to `wlan0`.
- Now go to the terminal. The correct folder is accessed by using '`cd`' command to get to the correct folder e.g., `cd Documents/lab_WiFi.`
- Run "test.py" command using (Fig. 6.6):

```
sudo python3 test.py
```

- Go to Chrome and type the correct IP Address into the space with port 80: e.g. "http://127.0.0.1:80/".
- The same can also be opened on any webserver connected on the same network as the RPi. For e.g. a mobile phone with same WiFi or mobile hotspot.

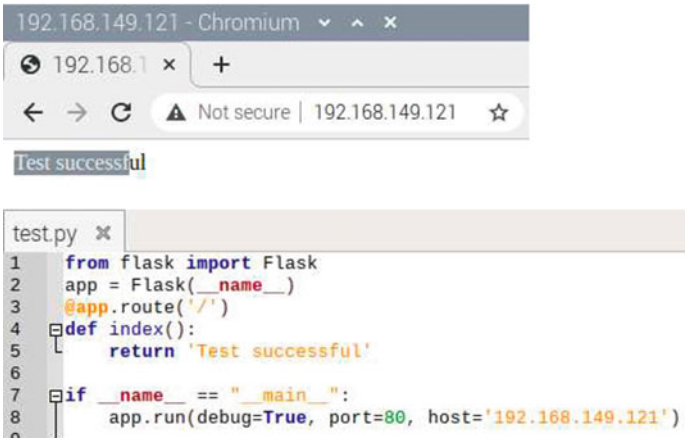


Fig. 6.5 Screenshot of running test.py code on Geany

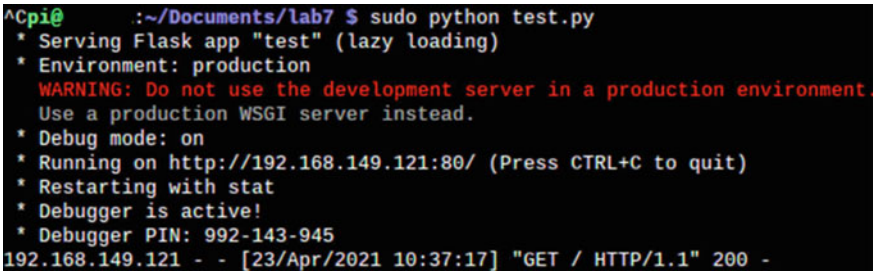


Fig. 6.6 Screenshot of running sudo python test.py command on terminal

The warning can be ignored as the code will run in such a way without any issues. First use '`pip install waitress`' from the terminal. Next, the warning can also be resolved using the following code lines replacing the last two lines:

```
if __name__ == "__main__":
  from waitress import serve
  serve(app, host="0.0.0.0", port=8080)
```

- Run "test.py" again command using:

```
sudo python3 test.py
```

6.6.4 Website Design

To use better visualisation and designing techniques, the templates for dynamic data and static for static data (colour, shape, etc.) are used. These are written in HTML and CSS language.

- Create a file named "`index.html`" from Geany and save it in "`templates`" folder.
- Create a file named "`style.css`" from Geany and save it in "`static`" folder.
- The updated "`test.py`" file:

```
from flask import Flask, render_template
import datetime

app = Flask(__name__)

@app.route("/")
def hello():
   now = datetime.datetime.now()
   timeString = now.strftime("%Y-%m-%d %H:%M")
   templateData = {
      'title' : 'HELLO!',
      'time': timeString
      }
   return render_template('index.html', **templateData)

if __name__ == "__main__":
   app.run(host='0.0.0.0', port=80, debug=True)
```

- The "`index.html`" file:

```
<!DOCTYPE html>
   <head>
      <title>{{ title }}</title>
      <link rel="stylesheet" href="../static/style.css/">
   </head>
   <body>
      <h1>Hello, World!</h1>
      <h2>The date and time on the server is: {{ time
}}</h2>
   </body>
</html>
```

- The "`style.css`" file:

```
body {
   background: grey;
   color: blue;
}
```

Running the program gives the following output as seen in Fig. 6.7:

The design of the web page was updated by adding a Cascading Style Sheet. A '`.css`' file holds colour and style information which can be used throughout

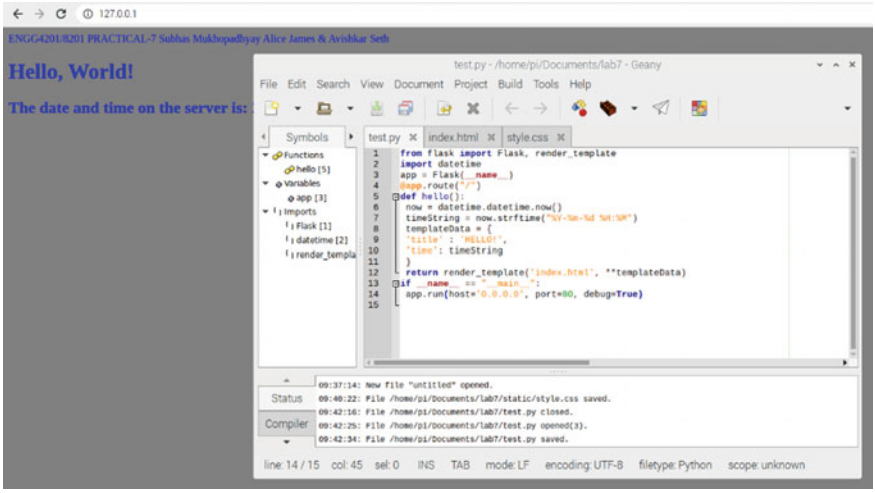


Fig. 6.7 Screenshot of running sudo python test.py command on terminal

many pages of a website. This is the equivalent of writing functions and methods for commonly used code in programming. It saves both time and resources, and keeps a consistent look to many different web pages within a website. The update also included some dynamic information, the time and date stamp of the web server, which changes on every display of the web page. This is useful as it shows the user if they are getting cached web pages or if a new web page is being generated each time.

This is important for IoT projects, as it would be difficult to debug if there is a hardware or software issue when data values are not updating, when it may simply would be the web browser is showing an old copy of the web page. The Webserver will be up when the python file is run. The python file includes "`@app.route("/")`" which searches for the HTML & CSS files inside the directory. The HTML and CSS file is linked by specifying the directory and filename using the following format:

```
<link rel="stylesheet" href ="../static/style.css">.
```

6.6.5 IoT System Design with RPi and WiFi

In this section we will design a complete IoT system using a sensor, buttons and LEDs.

- Interface a PIR sensor, a push button, and [3 LEDs (Use resistors)] with the RPi's GPIO on the breadboard.
- Fig. 6.8 is used to make the circuitry:
- Use the following codes provided:
- "*test.py*" code as shown in Fig. 6.9:
- "*index.html*" code as shown in Fig. 6.10:
- "*master.css*" code as shown in Fig. 6.11:
- Run the "*test.py*" program in the terminal.
- Run the webserver on a mobile phone browser with the IP address.
- Report the results of the program and web buttons created.

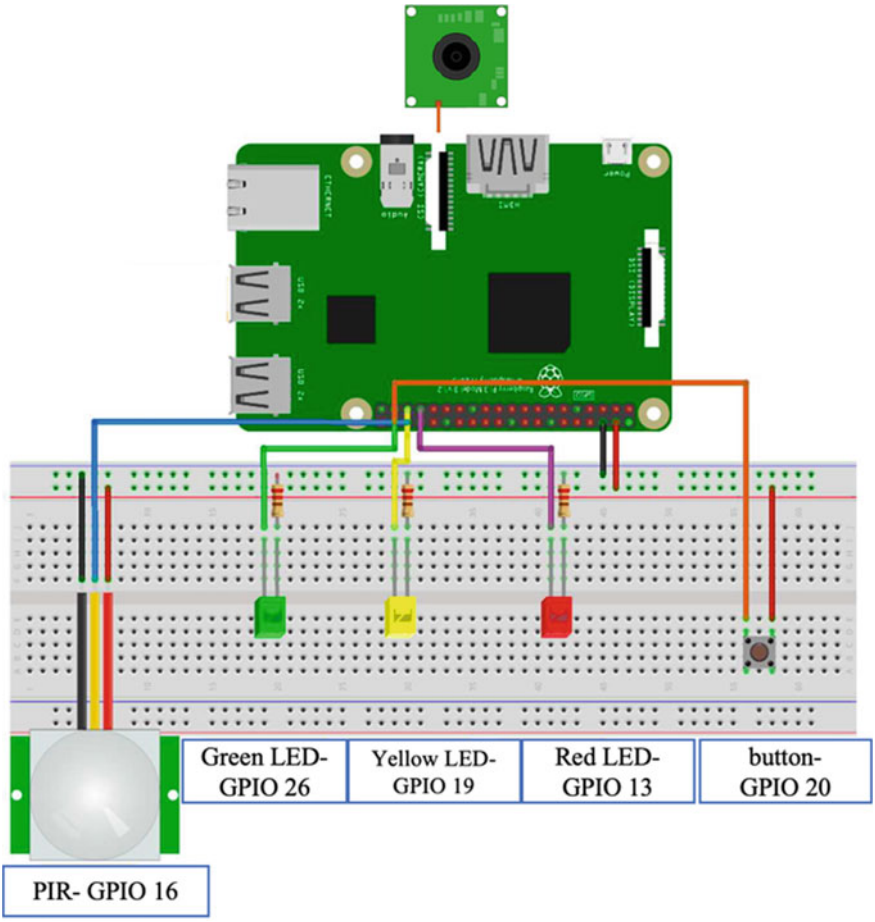
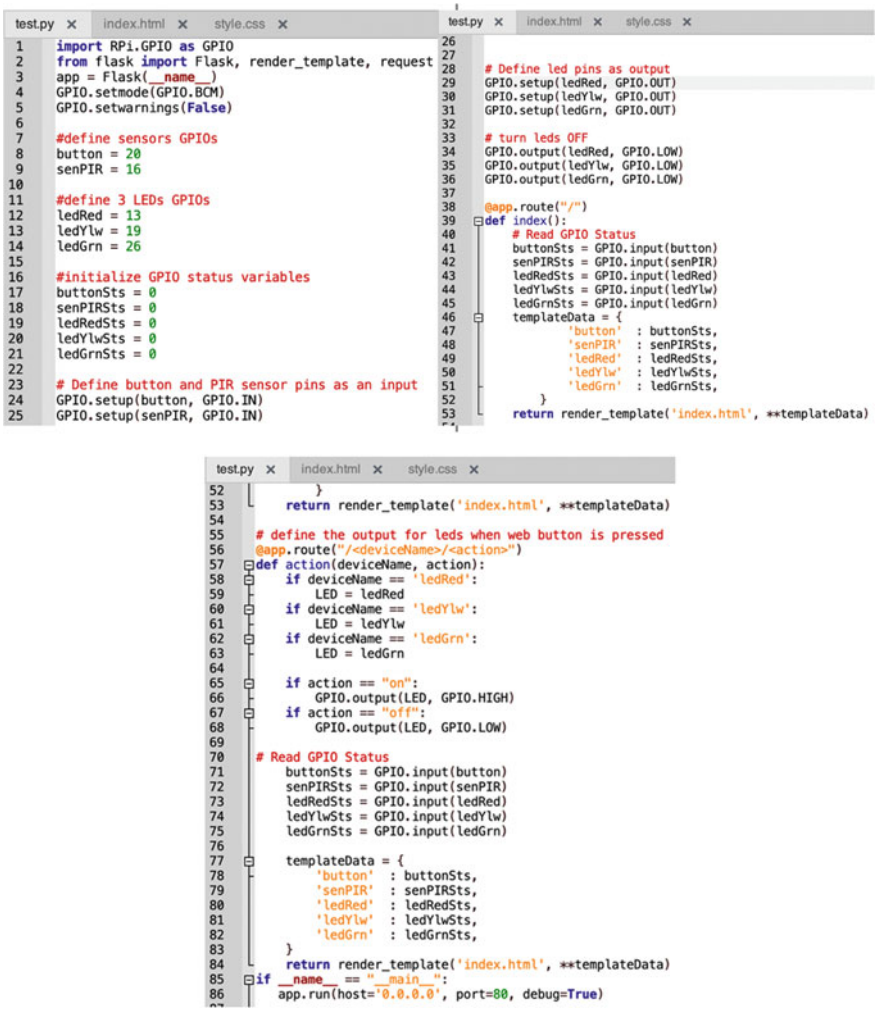


Fig. 6.8 Hardware circuitry for making RPi IoT system with Flask



```
test.py  index.html  style.css

import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#define sensors GPIOs
button = 20
senPIR = 16

#define 3 LEDs GPIOs
ledRed = 13
ledYlw = 19
ledGrn = 26

#initialize GPIO status variables
buttonSts = 0
senPIRSts = 0
ledRedSts = 0
ledYlwSts = 0
ledGrnSts = 0

# Define button and PIR sensor pins as an input
GPIO.setup(button, GPIO.IN)
GPIO.setup(senPIR, GPIO.IN)

# Define led pins as output
GPIO.setup(ledRed, GPIO.OUT)
GPIO.setup(ledYlw, GPIO.OUT)
GPIO.setup(ledGrn, GPIO.OUT)

# turn leds OFF
GPIO.output(ledRed, GPIO.LOW)
GPIO.output(ledYlw, GPIO.LOW)
GPIO.output(ledGrn, GPIO.LOW)

@app.route("/")
def index():
    # Read GPIO Status
    buttonSts = GPIO.input(button)
    senPIRSts = GPIO.input(senPIR)
    ledRedSts = GPIO.input(ledRed)
    ledYlwSts = GPIO.input(ledYlw)
    ledGrnSts = GPIO.input(ledGrn)
    templateData = {
            'button'  : buttonSts,
            'senPIR'  : senPIRSts,
            'ledRed'  : ledRedSts,
            'ledYlw'  : ledYlwSts,
            'ledGrn'  : ledGrnSts,
        }
    return render_template('index.html', **templateData)

    }
    return render_template('index.html', **templateData)

# define the output for leds when web button is pressed
@app.route("/<deviceName>/<action>")
def action(deviceName, action):
    if deviceName == 'ledRed':
        LED = ledRed
    if deviceName == 'ledYlw':
        LED = ledYlw
    if deviceName == 'ledGrn':
        LED = ledGrn

    if action == "on":
        GPIO.output(LED, GPIO.HIGH)
    if action == "off":
        GPIO.output(LED, GPIO.LOW)

# Read GPIO Status
    buttonSts = GPIO.input(button)
    senPIRSts = GPIO.input(senPIR)
    ledRedSts = GPIO.input(ledRed)
    ledYlwSts = GPIO.input(ledYlw)
    ledGrnSts = GPIO.input(ledGrn)

    templateData = {
        'button'  : buttonSts,
        'senPIR'  : senPIRSts,
        'ledRed'  : ledRedSts,
        'ledYlw'  : ledYlwSts,
        'ledGrn'  : ledGrnSts,
    }
    return render_template('index.html', **templateData)
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Fig. 6.9 Screenshot of running test.py code on Geany Python

The following output in Fig. 6.12 is an example of the above code provided:

The push-button, PIR, and LEDs are connected to the GPIO pins. The GPIO library is used to configure the states. The PIR and the push button are set to be input and the LEDs are set to be an output using `GPIO.setup` function. The `GPIO.setup` function reads the state of a GPIO pin and the `GPIO.setup` function sets the GPIO to either a HIGH or LOW state. The HTML & CSS file are both provided in this setup.

Fig. 6.10 Screenshot of running index.html code on Geany Python

test.py × index.html × style.css ×

```
<!DOCTYPE html>
 <head>
    <title>GPIO Control</title>
    <link rel="stylesheet" href='../static/master.css'/>
 </head>
 <body>
       <h1>RPi GPIO Control</h1>
       <h2> Sensor Status </h2>
       <h3> BUTTON ==>  {{ button  }}</h3>
       <h3> MOTION ==>  {{ senPIR  }}</h3>
       <br>
       <h2> Actuator Status & Control </h2>
       <h3> RED LED ==>  {{ ledRed  }}  ==>
           {% if  ledRed   == 1 %}
              <a href="/ledRed/off"class="button">TURN OFF</a>
           {% else %}
              <a href="/ledRed/on" class="button">TURN ON</a>
           {% endif %}
       </h3>
       <h3> YLW LED ==>  {{ ledYlw  }}  ==>
           {% if  ledYlw   == 1 %}
              <a href="/ledYlw/off"class="button">TURN OFF</a>
           {% else %}
              <a href="/ledYlw/on" class="button">TURN ON</a>
           {% endif %}
       </h3>
       <h3> GRN LED ==>  {{ ledGrn  }}  ==>
           {% if  ledGrn   == 1 %}
              <a href="/ledGrn/off"class="button">TURN OFF</a>
           {% else %}
              <a href="/ledGrn/on" class="button">TURN ON</a>
           {% endif %}
       </h3>
 </body>
</html>
```

Fig. 6.11 Screenshot of running style.css code on Geany Python

test.py × index.html × master.css ×

```
body {
    background: grey;
    color: blue;
}
.button {
  font: bold 15px Georgia;
  text-decoration: none;
  background-color: #EEEEEE;
  color: #333333;
  padding: 2px 6px 2px 6px;
  border-top: 1px solid #CCCCCC;
  border-right: 1px solid #333333;
  border-bottom: 1px solid #333333;
  border-left: 1px solid #CCCCCC;
}
```

6.7 Designing an IoT Project Using WiFi

The following section is a task provided that uses the above examples of using Flask to build a more complex system as an IoT project using WiFi.

- Use the above interfaced hardware to show the data of the following on ThingSpeak/Flask Webserver **graphically**
- Push Button Condition: 0 or 1 (High or Low)
- PIR Status: 0 or 1 (High or Low)
- LEDs: 0 or 1 (High or Low)
- Interface the RPi camera and send live video feed to a flask webserver.

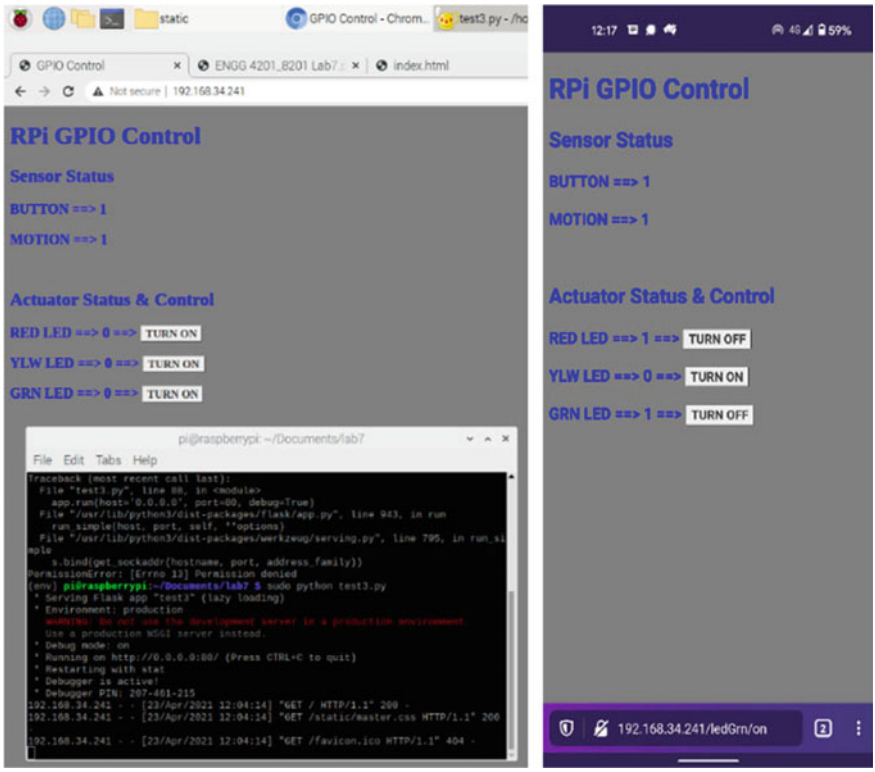


Fig. 6.12 Screenshot of running style.css code on Geany Python

6.8 Outcome of Student Implementation—Raspberry Pi

The following section shows the students implementations on the tasks given on the IoT project custom build using WiFi and Raspberry Pi4.

- IoT Custom Design on ThingSpeak:

A new python script named sending_to_thingspeak.py to read the GPIO state and send it to ThingSpeak as shown was created as shown in Fig. 6.13.

To send the data to ThingSpeak, the urllib2 library is used. The base URL and the write API Key from the ThingSpeak Channel are provided. The DHT22_data function is used to read the sensor data and return the values, that will be sent using the urllib2.urlopen function. The sensor data is sent every 20 s. The Webserver created previously and the sending_to_thingspeak.py run in parallel with each other. The Webserver serves as the control for the GPIO pins. The created ThingSpeak Channel has five channels ready to accept five sensor data from the Red LED, Yellow LED, Green LED, Push Button and the PIR sensor. The graphical results are shown below. (*Note initial values of 1, 2, 3, 4 and 5 were

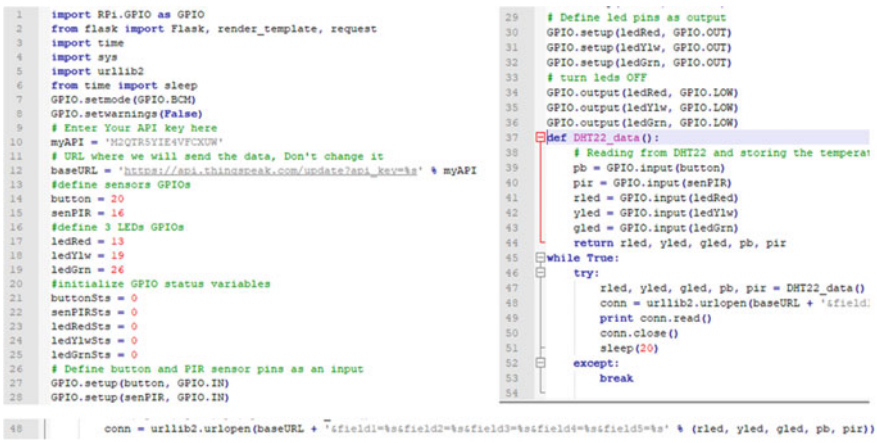


Fig. 6.13 Screenshot of running sending_to_thingspeak.py code on Geany Python

initially sent to test if the ThingSpeak channel which does not reflect the GPIO readings can be sent) (Fig. 6.14).

- IoT Custom Design on Flask Web Server:

Two alternatives were explored for displaying the sensor data: one which used a local database on the Raspberry Pi, the other used the ThingSpeak service.

Local Database Storing Sensor Data: Two examples were found which provided the recipes for displaying sensor data and live video within our web interface, similar to these links: https://www.instructables.com/From-Data-to-Graph-a-Web-Jorney-With-Flask-and-SQL/ and https://www.instructables.com/Video-Streaming-Web-Server/. Although large parts of the code were used without modification, the code was simplified to allow all data to be displayed within a single window.

Improvement to GUI using Bootstrap Framework: Firstly, Bootstrap was configured to provide both a more polished "look and feel" to our application, as shown in Fig. 6.15. Bootstrap is a popular framework for creating responsive web applications, which means that an app will size its elements appropriately for both larger screens and mobile devices.

Use of a SQLite 3 Database to Store Sensor Data: Following in the first link, an SQLite 3 database was setup to store PIR and breadboard button values.

The python index() function was modified to insert values into the database each time the page was rendered:

```
curs.execute("INSERT INTO PIR_data values(datetime('now'), (?), (?))",
(senPIRSts, buttonSts))
conn.commit()
```

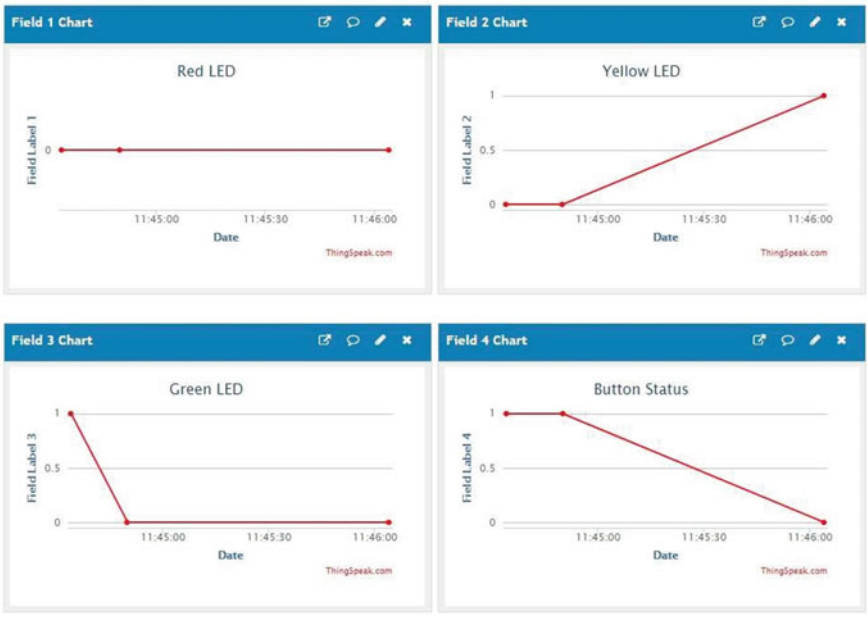


Fig. 6.14 Screenshot of ThingSpeak channel graphical results

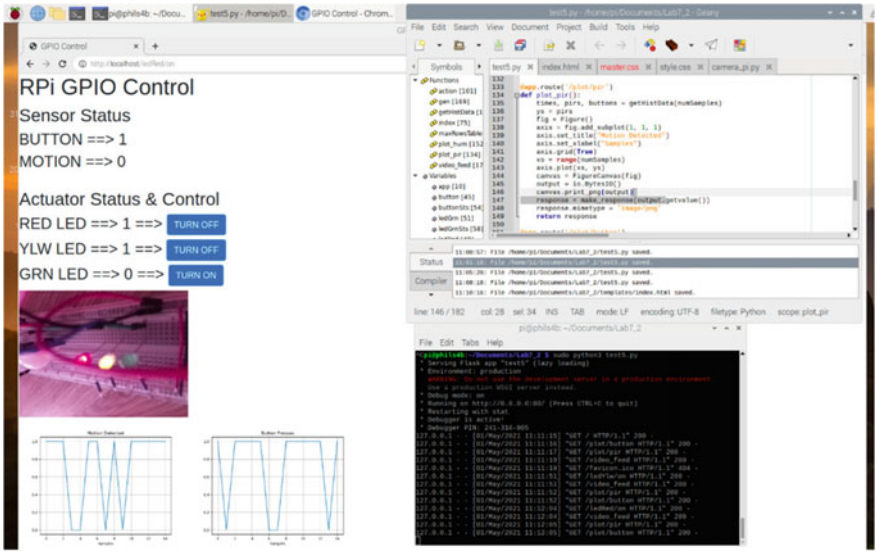


Fig. 6.15 Screenshot of running sending_to_thingspeak.py code on Geany Python

Using Matplotlib Module to Display Graphs of Sensor Data: The Matplotlib routines in the. SQL instructable guide were modified slightly to display sensor data for PIR and button status. Each routine retrieved a list of values from the database, and rendered the data within the web interface, as shown in Fig. 6.15. A section of the Matplotlib code is shown within the IDE in that Fig. 6.15.

Display of Live Video Data: The code in Webserver and video streaming was used with very minor modifications to take image data from the Raspberry Pi camera and to display it within the web interface. Figure 6.15 shows the live video feed, with the camera pointed at the breadboard, and showing the LEDs illuminated per their ON/OFF setting:

6.9 Implementation of WiFi Using Arduino

In this book, the Arduino Nano 33 IoT board has been used for WiFi projects. This chapter will further illustrate the use of Arduino Nano 33 IoT and how it can be configured with WiFi.

To begin with, a brief introduction to the board and the features of the chip are illustrated in this section. The Arduino Nano 33 IoT board is intended for creators who want to add Wi-Fi connectivity to their projects but don't have much experience with networking. The NINA-W10 that is used in Arduino Nano 33 IoT is a stand-alone multiradio MCU module that combines a powerful microcontroller (MCU) with a wireless communication radio. WiFi IEEE802.11 b/g/n in the 2.4 GHz ISM band is supported by the radio. The wireless MCU, flash memory, crystal, and matching, filtering, antenna, and decoupling components are all included in the NINA-W10, making it an extremely compact stand-alone multiradio module. Due to inbuilt cryptographic hardware accelerators, the module can be utilized to develop solutions with the highest level of security. The Fig. 6.16. illustrates the Arduino Nano 33 IoT board with the pin assignment of the WiFi module used in it.

The next section provides a set of guidelines that can be given to any student or user to commence their project with the board.

6.10 Guidelines Given to Users

This section elaborates on the guidelines given to the users.

Task 1: Implementing WiFi using Arduino Nano 33 IoT board.

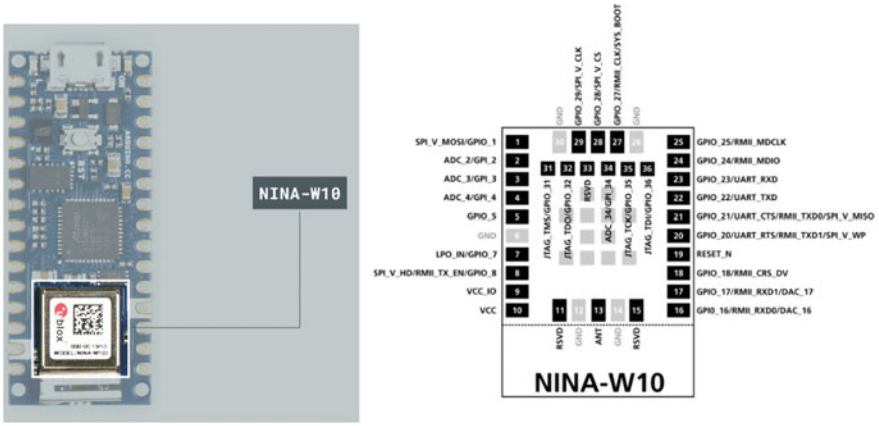


Fig. 6.16 The NINA-W10 chip module used in Arduino Nano 33 IoT and the pin-assignment for the NINA W10 module

1. Connecting Nano 33 IoT to WiFi

To configure the Bluetooth in this board, the user will use WiFiNINA library. It can be downloaded as seen in Fig. 6.17. from the Arduino IDE > Manage Libraries > WiFiNINA click install!

Please write the code 'WiFi_nano_test' given in the GitHub repository (https://github.com/AliceJames-1/IoT_Systems_Design_Book) and run it, the results are illustrated in Fig. 6.18. Enter the details of the WiFi credentials to ensure connection.

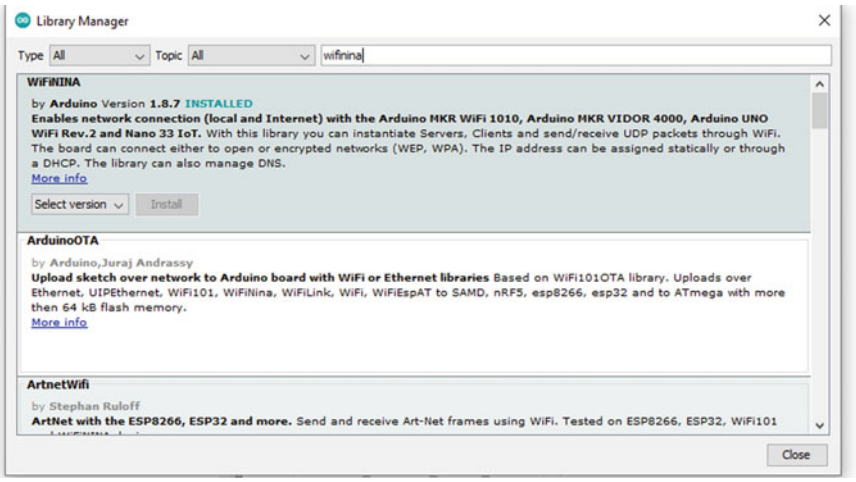


Fig. 6.17 The screenshot of the library manager to download the WiFiNINA library

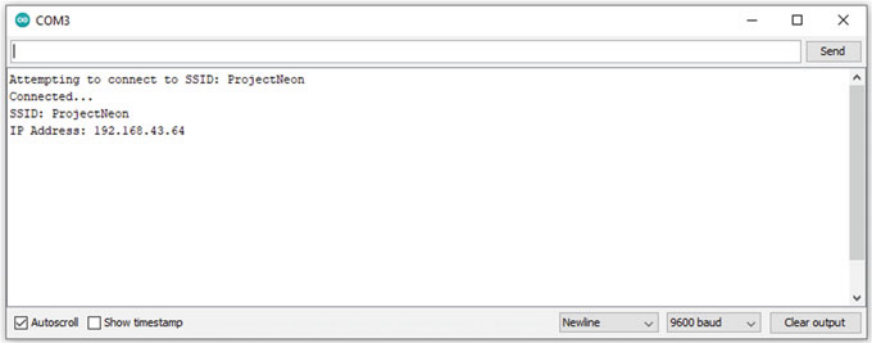


Fig. 6.18 The screenshot of the serial monitor of the IDE to illustrate connection to WiFi

2. Updating the Firmware

To update firmware, in WiFiNINA examples as seen in Fig. 6.19 and go to FirmwareUpdater and upload the code to the nano.

The next step is to click on Tools > WiFi101/WiFiNINA Firmware > and update firmware as illustrated in Fig. 6.20.

3. First connection with ThingSpeak (Send Random Numbers)

If not already, please download the ThingSpeak Library from the library manager as seen in Fig. 6.21.

Once downloaded, go to examples, and run the WriteMultipleFields from the Arduino MKR WiFi 1010 as shown in Fig. 6.22.

If the users have inserted their channel details correctly along with the WiFi details, the below results will be shown in the serial monitor as seen in Fig. 6.23 and the ThingSpeak channel will be updated.

4. Interface sensor with Nano 33 IoT

For the next step, interface the sensors with the given board and view results on serial monitor. Sensors: Ultrasonic, PIR. The same code can be followed as given in Chap. 2 of this book.

5. Transmit Data to ThingSpeak Dashboard using WiFi on Arduino Nano 33 IoT

Now, modify the transmission code and transmit the data of the sensors to the ThingSpeak Dashboard. The user's serial monitor results will be like the Fig. 6.24.

The custom ThingSpeak Dashboard will look like the following Fig. 6.25:

This concludes the guidelines for interfacing using WiFi on the nano 33 IoT, the students may explore further implementations for an IoT system.

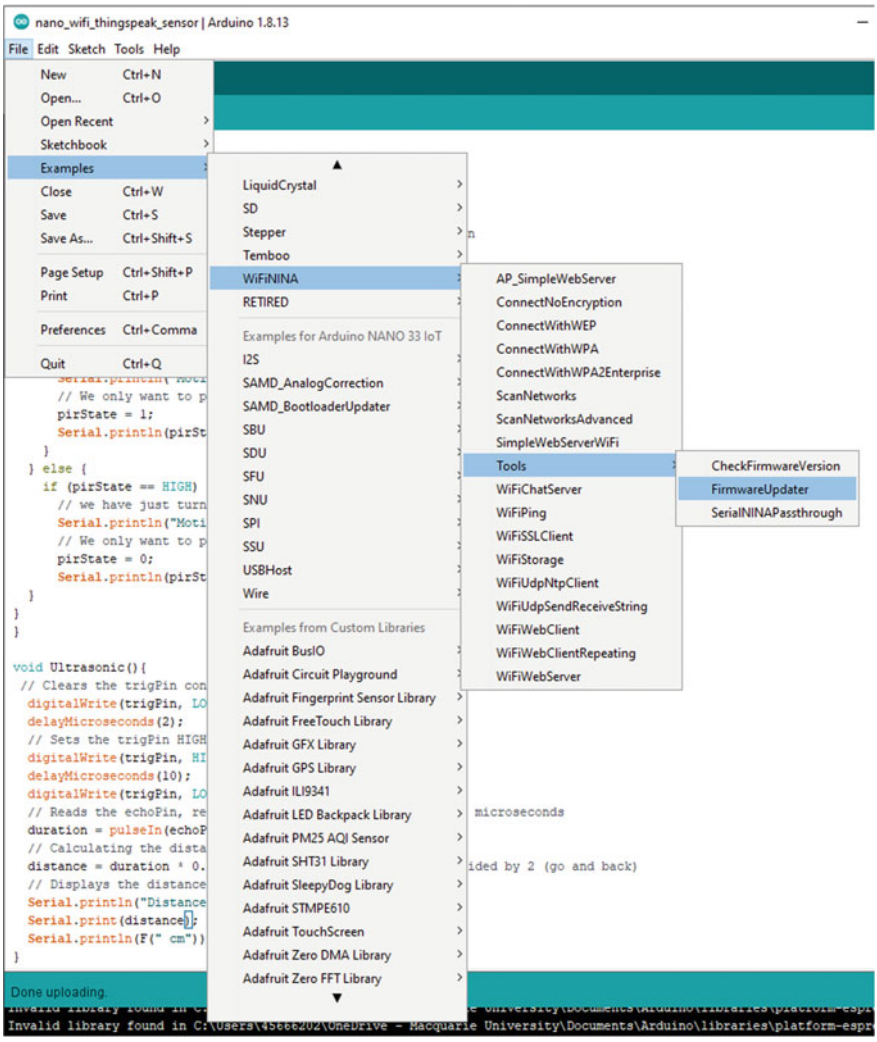


Fig. 6.19 The screenshot of the IDE to show the step to select the code for the FirmWareUpdater

6.11 Outcome of Student Challenges and Implementation —Arduino Nano 33

The following section illustrates and elaborates on the work done by the students in the lab session following the guidelines given above.

The Arduino part of this student report will cover how the group of students used an Arduino to send sensor data over WiFi to a ThingSpeak channel. It sent four fields every 20 s and ThingSpeak dynamically graphed all the data it received. The

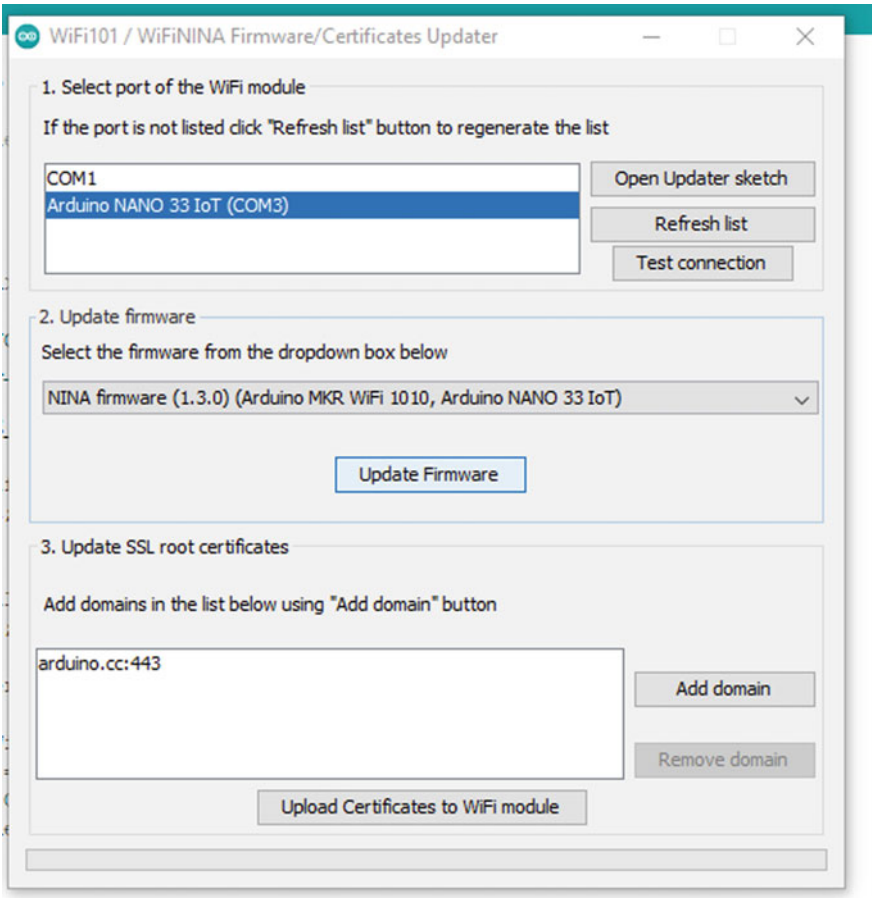


Fig. 6.20 The screenshot of the Firmware updater on Arduino IDE

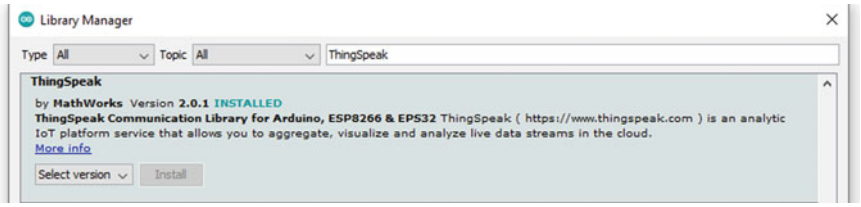


Fig. 6.21 The screenshot of the library manager to download the ThingSpeak library

first field was a counter which kept track of the number of data points. Second was a distance measurement from a HCSR04 Ultrasonic Distance Sensor. Third was a Boolean measurement from a Passive Infrared Sensor (PIR) where '1' was sent if

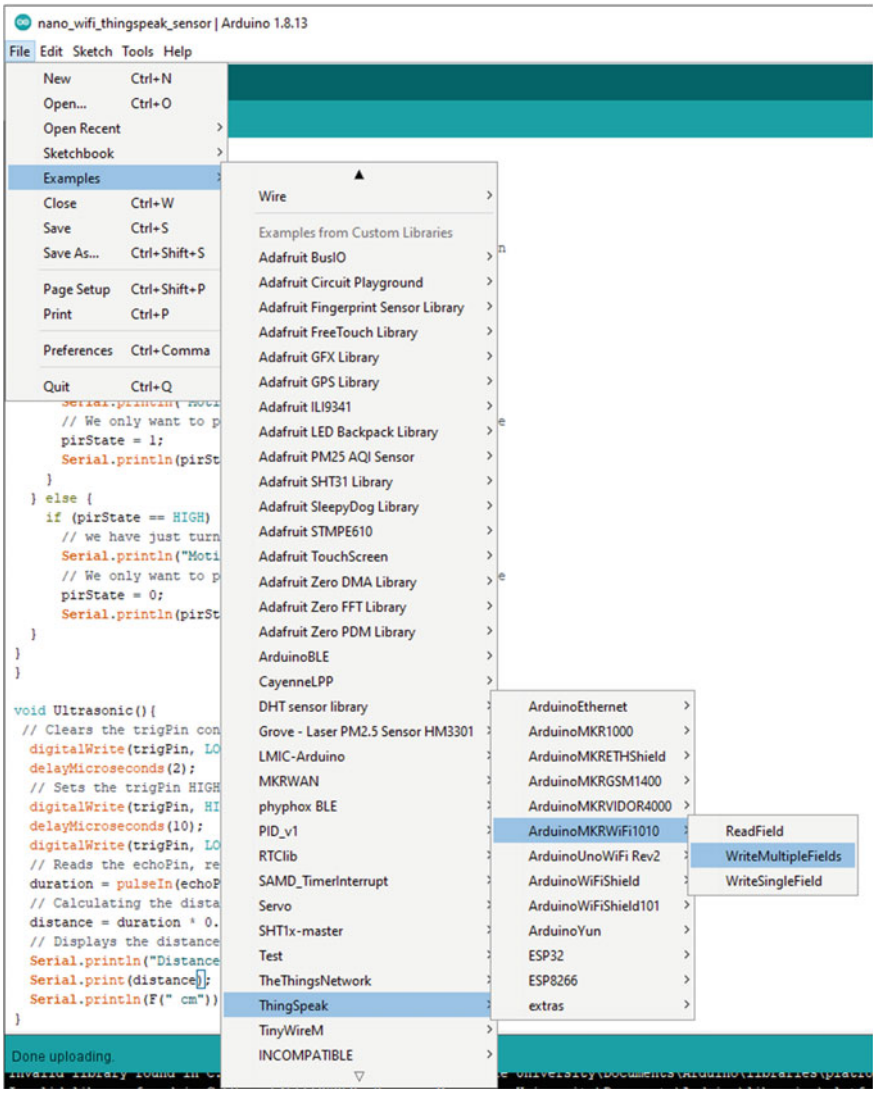


Fig. 6.22 The screenshot of the IDE to show the step to select the code for the WriteMultipleFields example code

motion was detected and '0' sent if no motion was detected. Fourth was a random number from 0 to 100 to simulate another sensor. They encountered a few challenges with the setup which will be explained along with how they overcame them.

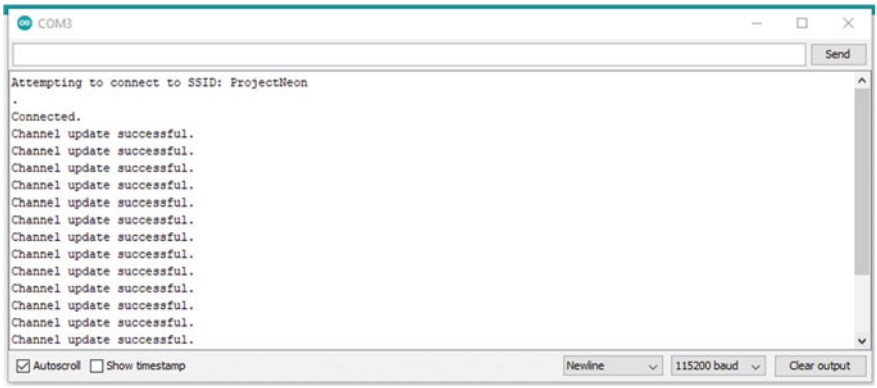


Fig. 6.23 The screenshot of the serial monitor of the IDE to illustrate ThingSpeak channel update

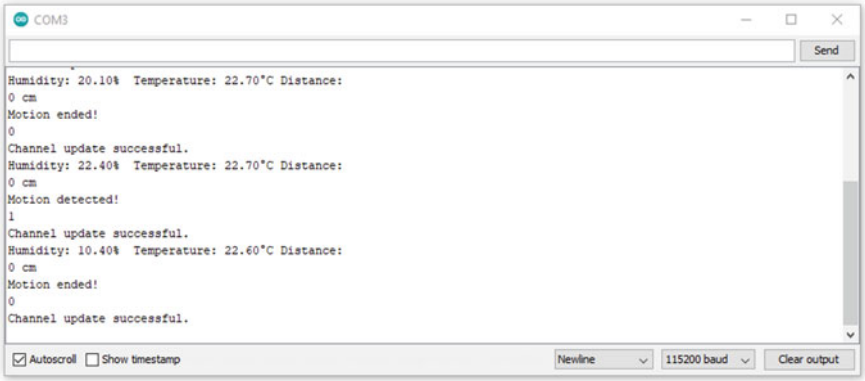


Fig. 6.24 The screenshot of the serial monitor of the IDE to illustrate ThingSpeak channel update with sensor data

Part 1—Implementing WiFi on an Arduino Nano 33 IoT

The initial step was to design and wire up the Arduino Nano 33 with the Ultrasonic sensor and the PIR sensor. The below Fig. 6.26 shows the design and the implementation:

Their first challenge came with installing the WiFiNINA library and uploading code to the Arduino. They encountered a fatal error—'core_cm0plus.h' file not found as seen in Fig. 6.27:

They tried numerous ideas and no other group was having the same issue. They looked at the header file and how it was being referenced. They made sure the header file existed and downloaded the newest version. What fixed it for them was connecting the Arduino to a different PC and running the Arduino IDE. Then they resumed to the previous PC and this error no longer appeared.

From here a new challenge appeared, the Arduino IDE would 'freeze' when attempting to upload a sketch. It would compile, begin uploading, but never finish.

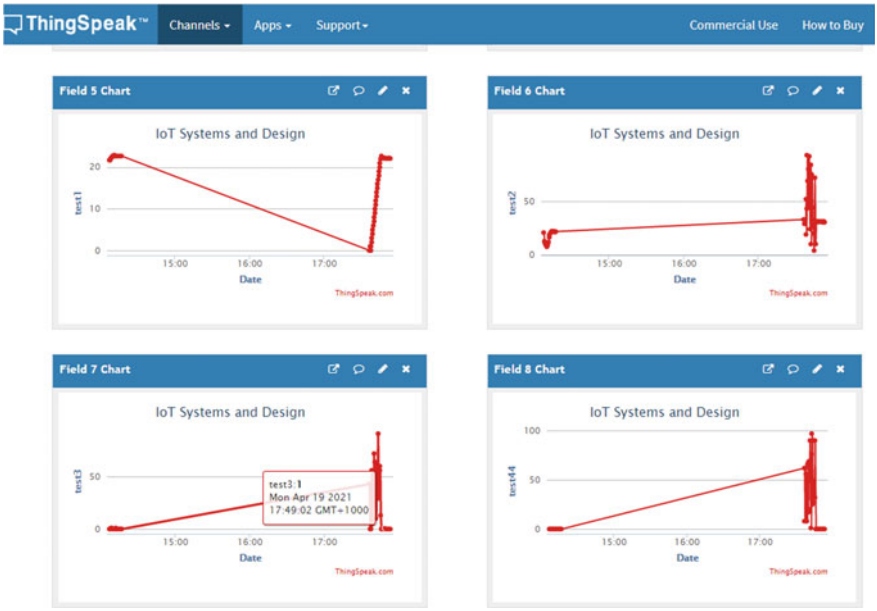


Fig. 6.25 The screenshot of the ThingSpeak channel with graphical representation of the transmitted sensor data

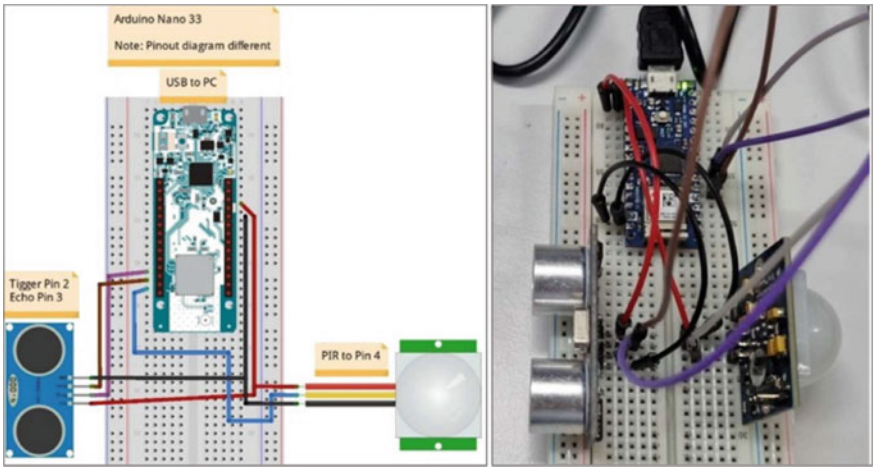


Fig. 6.26 (From left to right) The sensor and Arduino circuit diagram and the circuit implemented on the breadboard

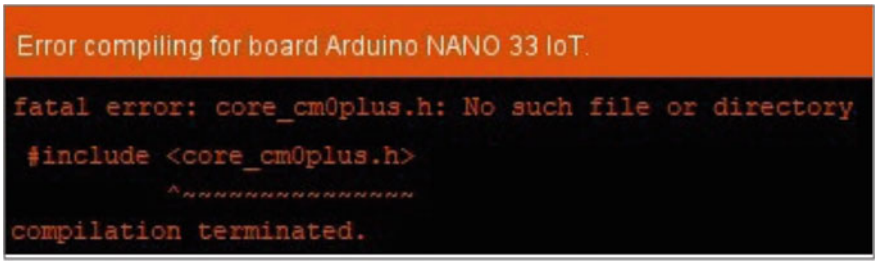


Fig. 6.27 The screenshot of the Arduino compilation error

Again, they tried many ways to solve the issue. After about 30 min of trying to get the basic blink sketch uploaded, it suddenly began working for no apparent reason. Once they were able to upload sketches again, they updated the firmware of the Arduino as seen in Fig. 6.28:

The Arduino was now ready to connect to a WiFi signal. After creating a WiFi hotspot from a mobile phone, the Arduino was able to connect and obtain an IP Address as seen in Fig. 6.29:

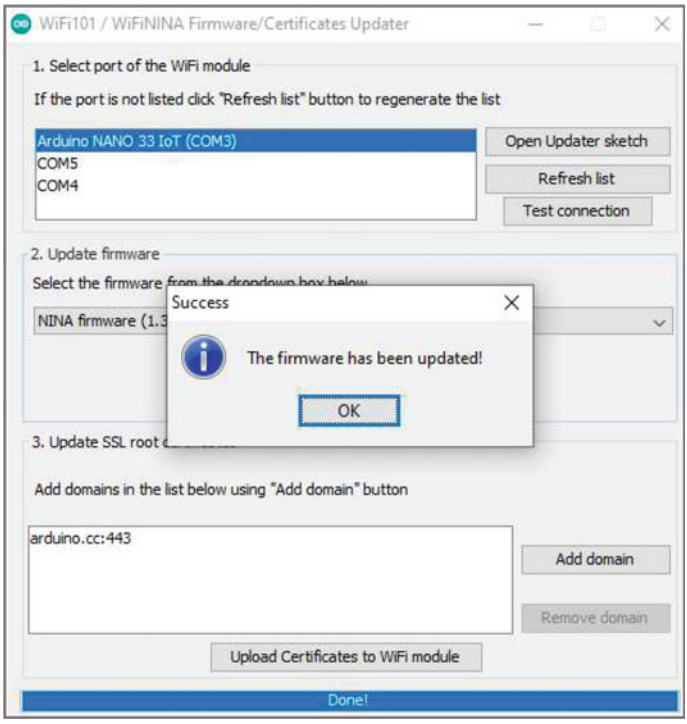


Fig. 6.28 The screenshot of the updated firmware by students

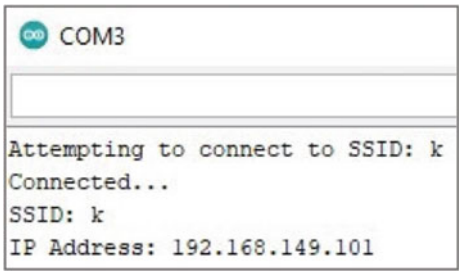


Fig. 6.29 The screenshot of the serial monitor of the IDE that depicts connection to the WiFi

They uploaded the 'writeMultipleFields' Arduino example sketch from the ThingSpeak library. After setting the correct Channel ID and Write API Key obtained from the Channel settings at ThingSpeak.com, they were able to publish some randomised values to test it was working illustrated in Fig. 6.30:

The last step was to send actual data measurements from the Ultrasonic and PIR sensors. Here they encountered multiple problems preventing a non-zero measurement from Ultrasonic sensor. An Arduino 33 Nano IoT board has traded size for ease-of-use, while it is very small which is good, the pins are labelled on the underside, and once plugged into a breadboard it is impossible to see the pin numbers. The wrong pin was being used and the Arduino was attempting to read a pin connected to nothing. Another problem was the sensor appeared more accurate and reliable when connected to the 3.3 v pin rather than the 5 v pin even though the datasheet states 5 v can be used. The Arduino sketch was modified to print debugging information to the serial monitor as depiceted in Fig. 6.31:

The graphs created by ThingSpeak are shown below in Fig. 6.32. and have some interesting points. The 'Number of Data Points' graphically shows that all the data packets are being sent and received as it is a linear line with no missing points and that they are all evenly spaced. The 'Distance' graph shows a large spike of over 1000 cm, which indicates an error in the sensor data as the HC-SR04 Ultrasonic Distance Sensor is rated to measure between 2 and 400 cm. Otherwise, the rest of the data appears as it may and highlights the power of IoT cloud computing as it effortlessly visualises multiple sensors.

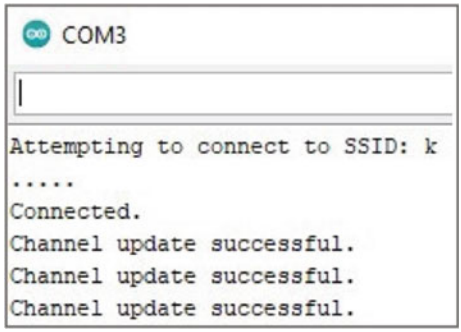


Fig. 6.30 The publishing of data confirmed on the serial monitor

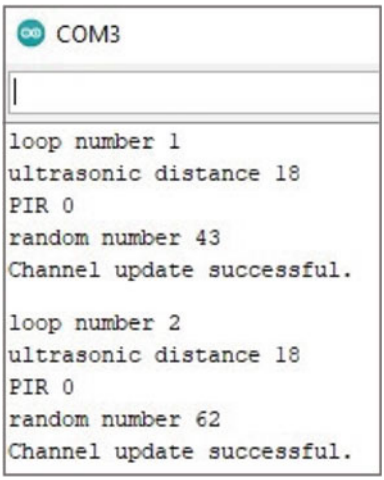


Fig. 6.31 The screenshot of the Arduino serial monitor illustrating results and channel update

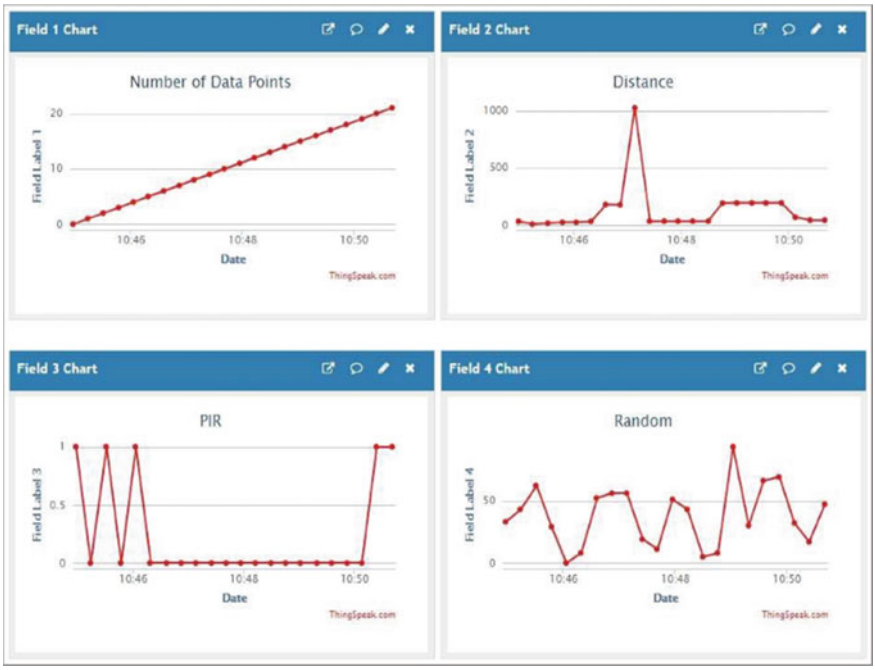


Fig. 6.32 The screenshot of ThingSpeak channel Graphing Data from Arduino

Part 1—Conclusion

The main goal of this practical was to setup and use WiFi to publish their IoT data. This is important as WiFi is ubiquitous, reliable, and fast, meaning that its use in future IoT projects is guaranteed.

ThingSpeak is free to use provided no more than 8200 messages are sent per day on average (ThingSpeak, 2021). This equates to approximately 3 million messages per year and is sufficient even if more sensors are added. For example, if a weather station were built as an IoT project with 20 sensors, it could be updated every 4 min. This would provide smooth graphs with many data points.

Suggested Reading

1. Cypress Semiconductor Datasheet: 'https://www.cypress.com/file/298786/download'
2. G.M. Mendez, M.A.M. Yunus, S.C. Mukhopadhyay, in *A WiFi based Smart Wireless Sensor Network for Monitoring an Agricultural Environment*. Proceedings of IEEE I2MTC 2012 Conference, IEEE Catalog number CFP12MT-CDR, May 13–16, 2012, Graz, Austria, pp. 2640–2645. ISBN: 978-1-4577-1771-0
3. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045
4. N.K. Suryadevara, S.C. Mukhopadhyay, R. Wang, R.K. Rayudu, Forecasting the behavior of an elderly using wireless sensors data in a smart home. Eng. Appl. Artif. Intell. **26**(10), 2641–2652 (2013). ISSN: 0952-1976, https://doi.org/10.1016/j.engappai.2013.08.004
5. S.D.T. Kelly, N.K. Suryadevara, S.C. Mukhopadhyay, Towards the implementation of IoT for environmental condition monitoring in homes. IEEE Sens. J. **13**(10), 3846–3853 (2013)
6. J.A. Nazabal, F. Falcone, C. Fernandez-Valdivielso, S.C. Mukhopadhyay, I.R. Matias, Accessing KNX devices using USB/KNX interfaces for remote monitoring and storing sensor data. Int. J. Smart Homes **7**(2), 101–110 (2013)
7. N.K. Suryadevara, S.C. Mukhopadhyay, Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sens. J. **12**(6), 1965–1972 (2012)
8. N.K. Suryadevara, A. Gaddam, R.K. Rayudu, S.C. Mukhopadhyay, Wireless sensors network based safe home to care elderly people: behaviour detection. Sens. Actuators A: Phys. **186**, 227–289 (2012). https://doi.org/10.1016/j.sna.2012.03.020
9. C. Ranhotigamage, S.C. Mukhopadhyay, Field trials and performance monitoring of distributed solar panels using a low cost wireless sensors network for domestic applications. IEEE Sens. J. **11**(10), 2583–2590 (2011)
10. K. Kaur, S.C. Mukhopadhyay, J. Schnepper, M. Haefke, H. Ewald, A zigbee based wearable physiological parameters monitoring system. IEEE Sens. J. **12**(3), 423–430 (2012)
11. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement **178**, 109424 (2021)
12. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
13. D. Thomas, R. Shankaran, M. Orgun, S. Mukhopadhyay, A secure barrier coverage scheduling framework for WSN-based IoT applications. IEEE Trans. Green Commun. Netw. (2021)

14. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the Internet of Things and serious games. Internet of Things **13**, 100332 (2021)
15. A.K. Sangaiah, S.C. Mukhopadhyay (eds.), *Intelligent IOT Systems in Personalized Health Care* (Elsevier, 2021). ISBN: 978-0-12-821187-8
16. S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 9. Internet of Things: Challenges and Opportunities (Springer, 2014). ISBN: 978-3-319-04222-0
17. N.K. Suryadevara, S.C. Mukhopadhyay, in *Smart Homes: Design, Implementation and Issues*. Smart Sensors, Meas. and Instrumentation, vol. 14 (Springer, 2015). ISBN: 978-3-319-13556-4
18. R. Yan, X. Chen, S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 26. Structural Health Monitoring: An Advanced Signal Processing Perspective (Springer, 2017). ISBN: 978-3-319-56125-7
19. S.C. Mukhopadhyay, A. Mason, in *Smart Sensors, Measurement and Instrumentation*, vol. 4. Smart Sensors for Real-Time Water Quality Monitoring (Springer, 2013). ISBN: 978-3-642-37005-2-1
20. S.C. Mukhopadhyay, in *Lecture Notes in Electrical Engineering*, vol. 96. New Developments in Sensing Technology for Structural Health Monitoring (Springer, 2011). ISBN: 978-3-642-21098-3
21. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors, https://doi.org/10.3390/s150510350

Chapter 7
Bluetooth Based IoT System

7.1 Introduction

The devices connected to the Internet of Things (IoT) can communicate and interact with each other without the need for a physical connection. Both Bluetooth and Wi-Fi are widely utilized for IoT devices, but each has advantages as well as disadvantages due to the environment in which they operate. Bluetooth, unlike WiFi was designed for portable devices and related applications, so it excels when it needs to connect two devices with minimal configuration. Furthermore, because Bluetooth uses weak signals, there is less interference, and devices may interact even in noisy surroundings.

Bluetooth was created by Ericsson in 1994 to provide wireless headsets. Bluetooth is a wireless technology standard that manages connections between devices without the use of a password and is based on physical proximity. Bluetooth employs UHF (ultra high frequency) radio waves between 2.400 and 2.485 GHz to communicate between two devices. The Bluetooth specification stipulates a minimum range of 10 m, but the maximum range is determined only by the output power of the device The hardware, communication, software system, and application layers make up a typical IoT design, with Bluetooth serving as the communication layer. The communication layer, which consists of a multi-layer stack that includes data link, network or transport, and session protocols, is a vital link between the layers. An IoT device needs have a Bluetooth-capable microprocessor as well as a second device to pair with, for it to be Bluetooth-compatible. Bluetooth Classic and Bluetooth Minimal Energy (BLE), which is designed for devices that require low power consumption, are two variants of the Bluetooth protocol that are often utilized by IoT devices that cannot communicate directly with one another, a comparison is shown in Fig. 7.1.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_7

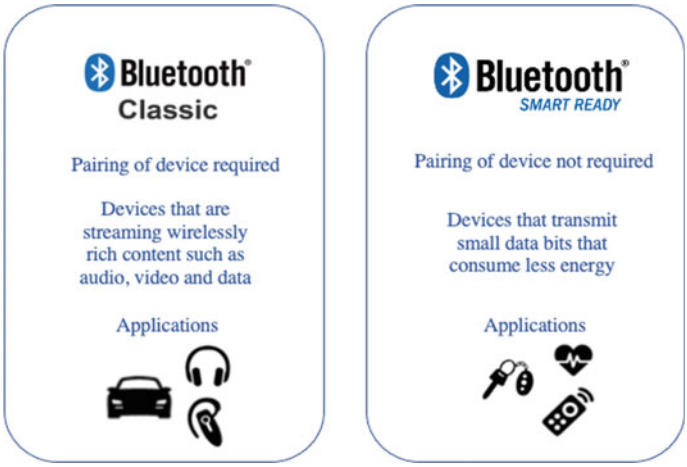


Fig. 7.1 (From left to right) The comparison between classic Bluetooth and BLE

7.2 Description

The objectives and components description given for the implementation of the project is elaborated in this section.

7.2.1 Objectives

The major aim of this project is to give the users a firsthand experience of building a complete IoT system using Bluetooth/BLE. The following points describe the objectives in detail:

- Developing a complete working IoT system based on Bluetooth/BLE in a short amount of time of about 3–4 h.
- Give users an idea of the microcontroller/microprocessor hardware functionality in building the system.
- Learning about the Bluetooth/BLE transmission protocol used.
- Sending the data wirelessly and understanding the use of various available API's and their features.

7.2.2 Hardware Used in the Project

The hardware components used in this chapter mainly involve the sensors and microcontroller/ microprocessor unit. The following components are required to build this system:

- Microcontroller: Arduino Nano 33 IoT
- Micro USB cable: Arduino Nano 33 IoT compatible
- Microprocessor: Raspberry Pi Model 4B
- Cable for Raspberry Pi: HDMI Cable, USB A to C
- Sensors and Components:
- PIR Sensor Module- Motion Sensing (×2)
- HC-SR04 Ultrasonic Module- Distance Sensor
- Raspberry Pi Fish-eye wide angle camera
- LED (×2)
- Pushbutton
- Prototyping Solderless Breadboard.

7.2.3 Software Used in the Project

The following are the software/applications used in this project:

- Microcontroller Programming: Arduino IDE
- Data Transmission and interdevice control for Arduino BLE: nRF application or LightBlue application
- Python Application for Raspberry WiFi connectivity: BlueDot.

7.3 Motivation

In the recent years, approximately, one-third of the smart devices is connected using Bluetooth. In this chapter, users will learn implementation of Bluetooth Transmission. We will utilise the Bluetooth function in the given Arduino Board as well as Raspberry Pi. Later, users will use Arduino Nano 33 IoT, the BLE function and inter-device communication using BlueDot for Raspberry Pi. There are various advantages of using Bluetooth technology such as simplification of setup, enable communications in noisy environment due to limited interference, etc.

7.4 Bluetooth Functionality on Raspberry Pi

Need of microprocessors to communicate with each other wirelessly is useful in many embedded systems. For example, an outside temperature sensor needs to communicate the temperature data to an inside display. One such wireless protocol is Bluetooth. Wireless data transfer makes IoT devices talk to each other. Bluetooth based IoT devices are one of the most commonly available technology in most of

the SBCs. Unlike WiFi, Bluetooth devices would not provide us internet access. In terms of SBCs, most of them have Bluetooth chips to enable connectivity. The most recent technology of Bluetooth that these boards use is based on the Bluetooth Special Interest Group (SIG), previously standardised by IEEE 802.15.1.

Raspberry Pi Model 4B uses the Bluetooth 5.0 (BLE) i.e., Bluetooth Low Energy. For BLE, Bluetooth 5 offers choices to twice the speed (2 Mbit/s burst) at the cost of range, or to deliver up to four times the range at the cost of data rate. The increased transmissions could be critical for Internet of Things devices, which connect multiple nodes across a home. Similar to WiFi, most of the Raspberry Pi boards that have built-in support for Bluetooth accessibility, have the BCM43455/6 chip. A high-speed 4-wire UART interface is supported by the Bluetooth component.

Since the debut of the Raspberry Pi3 in 2016, the single-board computer has featured built-in Bluetooth connectivity, allowing to connect wireless peripherals such as keyboards, game controllers, headsets, and many more devices. If the older models that don't have Bluetooth enabled on them need to access it, one can use a third party Bluetooth USB device.

7.5 Setup and Installations of Bluetooth on Raspberry Pi

This section explains how to enable and use Bluetooth with multiple devices on the RPi.

- Bluetooth with Desktop GUI

The most basic method of enabling Bluetooth on the RPi is from the taskbar on the top tight corner in the full desktop version of the RPi OS. It allows to connect to nearby devices to connect and send/receive data.

This tool is a very simple and straightforward method. However, it does not allow advanced options and device configurations for devices such as gaming controllers or wireless speakers. Installing an additional set of programmes to control the Bluetooth settings can help to solve these problems. To enable a more GUI based configuration, RPi allows multiple app installations.

- Bluetooth Manager Tool via Desktop GUI

Blueman is a full-featured Bluetooth manager developed in Python and utilising the GTK graphical user interface. Blueman is intended to make manipulating the BlueZ API and simplifying Bluetooth chores as simple as possible. To install the manager the following instructions are used:

Open the terminal and type the following command one by one and press enter:

```
sudo apt update && sudo apt upgrade
sudo apt install bluetooth pi-bluetooth bluez blueman
sudo reboot
```

The first command checks for new software packages and device firmware updates. The second command installs the necessary additional packages for the Blueman app. Lastly, the RPi is rebooted for the above changes to take effect. Once rebooted, the RPi taskbar will have the following icon to click as shown in Fig. 7.2.

Click the icon and turn on the device's 'Turn Bluetooth On'. Next, click on 'Make discoverable' to allow other devices with Bluetooth be able to see the device on their network as shown in Fig. 7.3.

When the Bluetooth is enabled, click on the option 'Setup New Device'. In this setup, a new device can be permanently added with faster access and security check. Click 'Next' on the first page. Add the appropriate device from the list or click on the search button to find a new device as shown in Fig. 7.4.

Click 'Next' after clicking on the correct device. 'Pair Device' and click 'Confirm' to pair the device. In this way two devices can be paired and added as a trusted device, so that next time it can be directly paired without doing this process.

The Raspberry Pi can connect to many other devices in a similar manner. It can also connect to mobile phones, other boards, and gaming controllers as well to send/ receive data. In the next section, we will explore using Bluetooth connectivity to implement an IoT project.

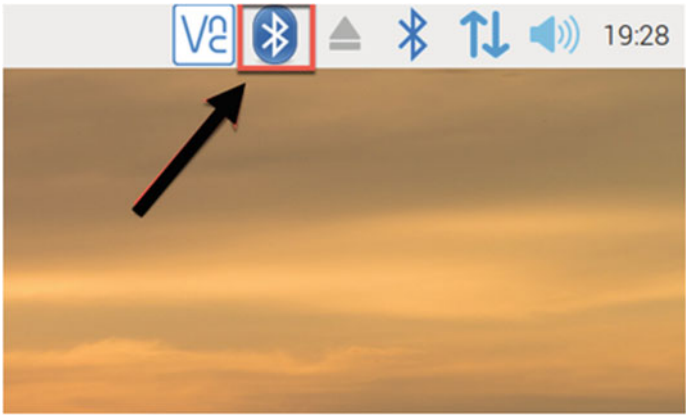


Fig. 7.2 RPi desktop menu bar icon for Blueman app

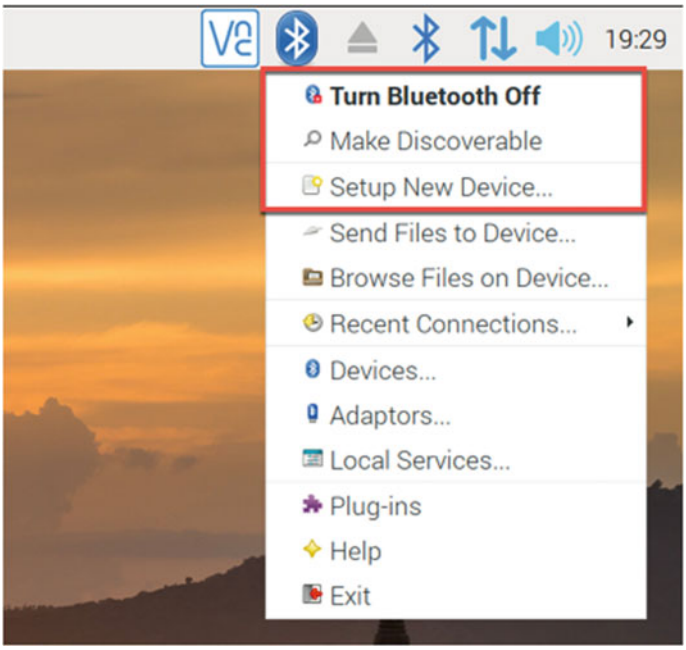


Fig. 7.3 RPi desktop for Blueman app connection process

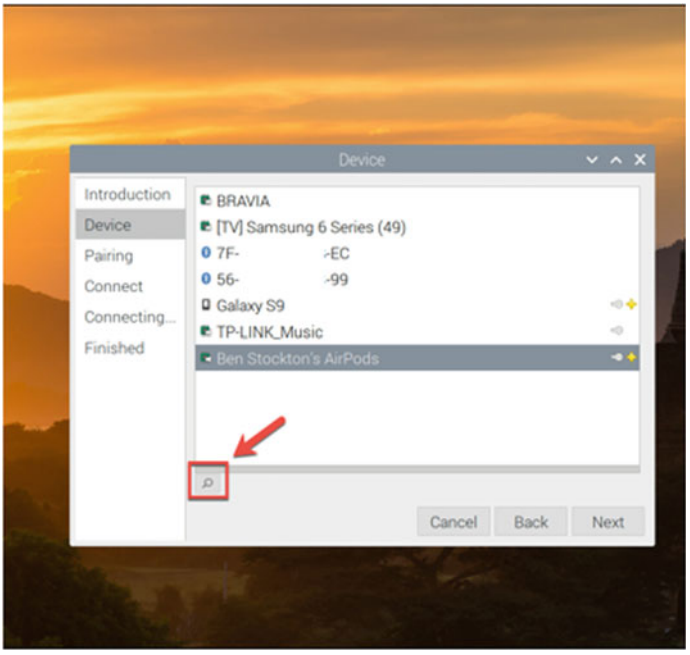


Fig. 7.4 RPi desktop for Blueman app connection process

7.6 Guidelines Given to Users

In this section, an IoT project using Bluetooth communication will be shown. This will help to understand how Bluetooth technology can be used to send and receive data from an RPi IoT node. This section will elaborate the example codes and guidelines given to users.

In this task, we will learn how to control different sensors and peripherals connected to a RPi using another Bluetooth device as a controller via Bluetooth communication. Ultimately a depth measurement system will be built between the Rpi and an Android device via Bluetooth.

7.6.1 Software Installations and Setup

Start by connecting the RPi as a desktop. Go to the terminal and type the following commands one after the other and press enter:

```
sudo apt update
sudo apt install python3-gpiozero
sudo pip3 install bluedot
sudo pip3 install bluedot -upgrade
```

Blue Dot will be the application we will be using to establish communication between 2 RPi boards and an Android Device and an RPi. Blue Dot is a Bluetooth remote and a Python module that allows us to operate the Raspberry Pi projects wirelessly.

7.6.2 Bluetooth Connections

There are two methods of controlling the RPi device that will be explained in this section. The first one is RPi to another Rpi.

To connect a RPi to another RPi, the following procedure is used. Before running the Blue Dot application, it needs to "pair" the Raspberry Pi devices using the Bluetooth menu options. After making the devices discoverable, the peer device could be added, enabling communications once the pair request was accepted at the peer. Make sure the correct RPi is selected from the list of discoverable devices. This procedure was required as a security measure to prevent unauthorised users performing operations on our devices. It was only required once.

Click 'Next' after clicking on the correct device. 'Pair Device' and click 'Confirm' to pair the device. In this way two devices can connect to each other.

7.6.3 Controlling the IoT System (Buttons)

In the IoT System as shown in the Fig. 7.5:

- Pi1: Runs the Blue Dot App only.
- Pi2: Runs the Python Programs only.
- On the Pi1 run the Blue Dot App by going to Pi Icon > Run > type the following: "`bluedotapp`" and keep it open.
- On the Pi2 go to Thonny/Geany Python. Keep it open.
- Now, a Wireless Bluetooth Button is created by entering the following code in Thonny or Geany Python IDE:

```
from bluedot import BlueDot
bd = Bluedot()
bd.wait_for_press()
print("Hello World")
```

- Save and Press Run. It will start running and wait for the connection as shown in Fig. 7.6.
- Next, the Pi1 is selected and the correct Pi2 address to connect is clicked.
- Click on the Blue Dot to see the output on Pi2.
- Stop the program before running another program on Thonny.
- Note: The procedure mentioned above is common for most of the following tasks.
- The 'when_pressed' function is also used.
- Use the "`say_goodbye()`" and "`when_released`" functions to print "goodbye" when the button is released. Update the code with these changes.
- Similarly, use the "`shout_hello()`" and "`when_double_pressed`" functions to print "goodbye" when the button is released. Update the code with these changes. The Fig. 7.7 shows the updated codes with the output.

The built-in library `BlueDot` allows us to use functions that is designed for the Blue Dot application. Calling the `BlueDot()` function from this library starts the server and waits for the client to connect in the Blue Dot application before

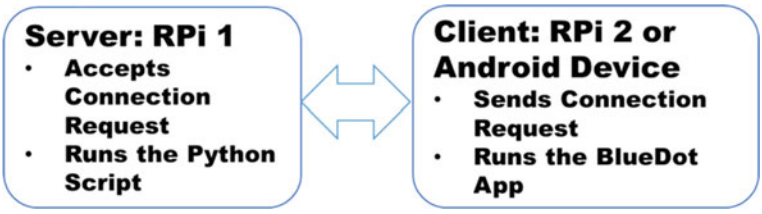


Fig. 7.5 RPi server and client

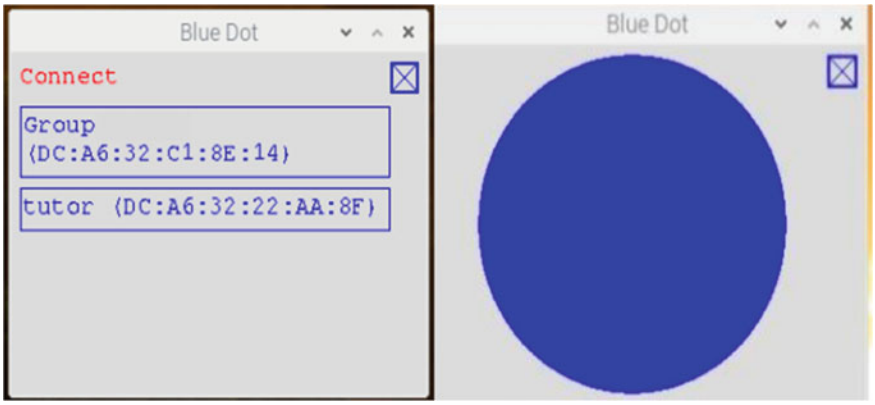


Fig. 7.6 Blue dot server and client button

commencing on the rest of the program. In the first Hello World, the `wait_for_press()` function is used to wait for the GUI in the Blue Dot application to be clicked or tapped. In the code above, whenever the GUI is tapped, it will print Hello World and end the program. In the second program. Adding a pause() at the end of the code creates an infinite loop so that the code can run multiple times. The `say_hello()` function is also created to print "Hello". The third program shows the functions like `when_pressed`, `when_released` and `when_double` pressed in the `BlueDot` library, which allows versatility and more options like pressing, releasing and double pressing for controlling the device depending on the input of the user.

7.6.4 More Example Programs with GPIO and BlueDot

The program is extended by connecting an LED connected to GPIO 17 of the RPi2. The schematic and the code is shown below in Fig. 7.8.

In the following code above, the LED stays on as long as the Blue Dot is pressed and the LED turns off when the Blue Dot is released.

A picture which was triggered remotely over Bluetooth is shown as a schematic and the code below in Fig. 7.9.

A camera is now interfaced with the RPi. The program was configured such that pressing the button would take and save a picture from the PiCamera.

A code for taking a video can also be used following the codes from Chap. 4. For the video capture, it is noted that `stop_preview` would not function when it was included in a sleep call within a function called by BlueDot. It can be resolved by moving `stop_preview` into a separate BlueDot function, thus avoiding the use of sleep.

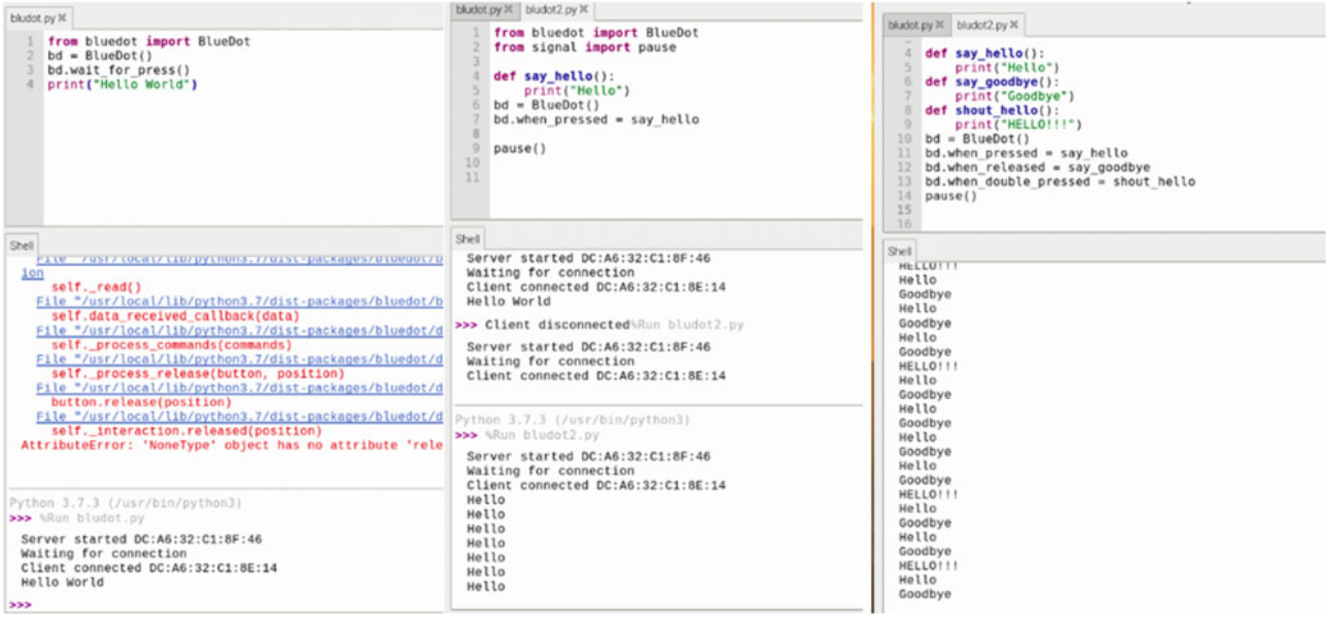


Fig. 7.7 Different basic programs with blue dot app on python

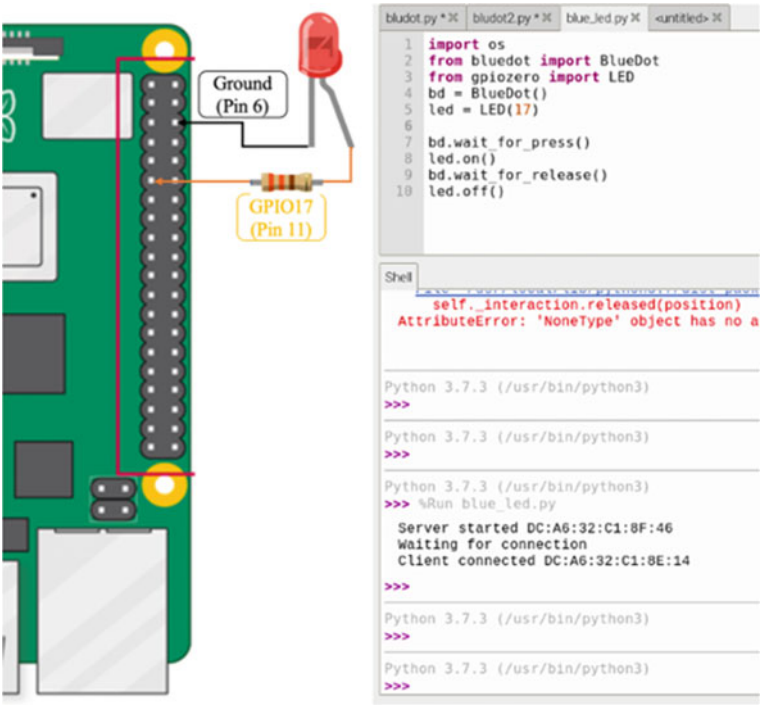


Fig. 7.8 Circuit for controlling LED with BlueDot library and code

All of the above given setup and programs are done using two RPi boards connected to each other. Further, the BlueDot library after this, struggled to communicate more information between the devices, by implementing multiple buttons on the interface as it was determined during the setup. The Python version of the BlueDot app used on the Raspberry Pi does not support multiple "dots".

7.6.5 Appearance, Layout and Adding Multiple Buttons in BlueDot

This section explores other examples from the BlueDot library. If there are many buttons, it becomes necessary to use colour to distinguish between different functions.

Figure 7.10 shows the code to change the colour of the button. The appearance can also use a hex value (#rrggbb) or a tuple (rr, gg, bb, aa) to define the colour or simply writing the colour:

bluecam.py

```
from bluedot import BlueDot
from picamera import PiCamera
from signal import pause
from time import sleep

bd = BlueDot()
cam = PiCamera()

def take_picture():
    cam.start_preview(fullscreen = False,window = (100,20,640,480))
    sleep(3)
    cam.capture("/home/pi/Desktop/task2-5.jpg")
    cam.stop_preview()

bd.when_pressed = take_picture
pause()
```

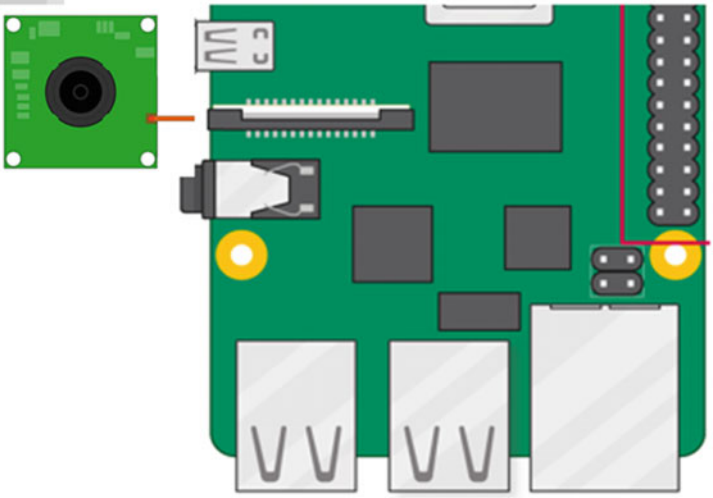


Fig. 7.9 Circuit for controlling camera with BlueDot library and code

```
bd.color = "red" or
bd.color = "#00ff00" or
bd.color = (0, 255, 0, 200)
```

To increase the functionality, multiple buttons can be added to the same program. The buttons don't have to be circle necessarily as well. The Fig. 7.11 shows the code and output of adding different buttons to make a joystick style button control:

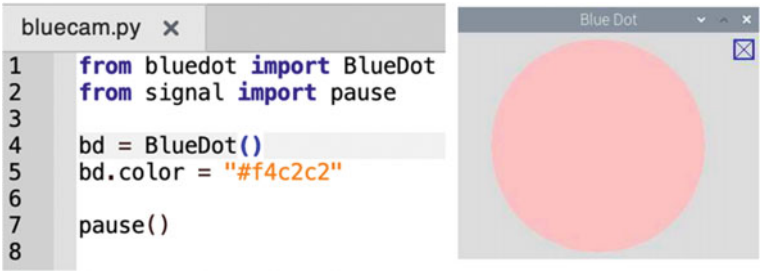


Fig. 7.10 Changing colour on BlueDot app and code

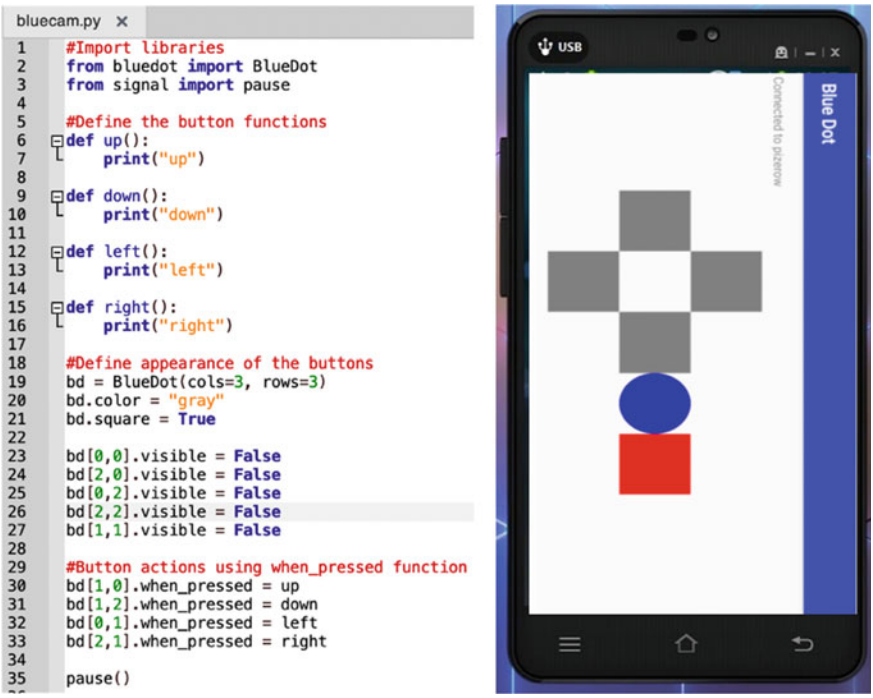


Fig. 7.11 Adding multiple buttons on BlueDot app and code using android device with RPi4

7.6.6 IoT System Design: Remote Control for Depth Measurement System

The design steps are described to implement a complete IoT system prototype of a remote control for depth measurement system.

- In this task, user develops a depth measurement IoT system prototype. (Make any assumptions needed to function for the system).
- The system will be installed inside a water tank.
- It needs to tell the distance between the sensor which is placed on the lid of the tank and the water inside the tank.
- The system will have 4 peripheral components: Camera, Ultrasonic Sensor, 2 LEDs. (Fig. 7.12).
- The components are interfaced to the Pi2. To avoid damage of any electronic components, the devices need to be powered off.
- Create 4 Buttons of different colours to control the functions of the 4 peripherals connected to the Pi2
- The button functions are as follows:
  - First button: Control the LED1 On/Off state.
  - Second button: Control the LED2 On/Off state.
  - Third button: Control the Ultrasonic Sensor On/Off state (Give Distance Output).
  - Fourth button: Control the Camera On/Off state (Take a picture and save it) .

Some useful Ultrasonic Distance Sensor programs for basic functionality:

```
from gpiozero import DistanceSensor
from time import sleep

sensor = DistanceSensor(23, 24)
while True:
  print('Distance to nearest object is', sensor.distance, 'm')
sleep(1)
```

OR

```
from gpiozero import DistanceSensor, LED
from signal import pause

sensor = DistanceSensor(23, 24, max_distance=1, threshold_distance=0.2)
led = LED(16)

sensor.when_in_range = led.on
sensor.when_out_of_range = led.off
pause()
```

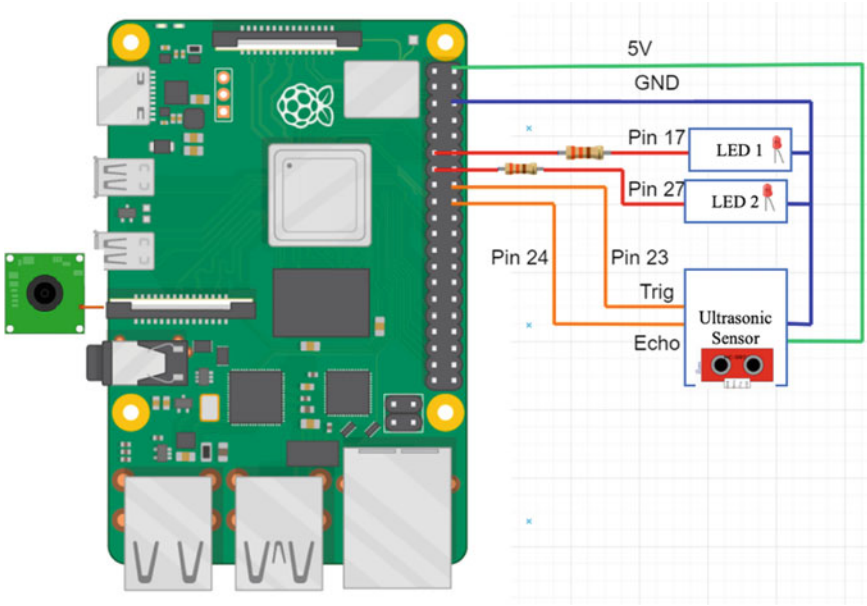


Fig. 7.12 Hardware circuitry for making RPi IoT system for depth measurement

7.7 Outcome of Project Implementation—Raspberry Pi

The following section shows the implementations on the tasks given on the IoT project custom build using Bluetooth and Raspberry Pi4.

- IoT Custom Design Depth Measurement System

For this task, an HC-SR04 sensor was interfaced to the Raspberry Pi and investigated it in the context of a hypothetical "Depth Measurement" application. One of the advantages of using the Raspberry Pi for this scenario is that there is support for the ultrasonic sensor within the "`gpiozero`" library. This library contains a `DistanceSensor` class, which provides a simple, convenient interface to the sensor functionality. With "`gpiozero`", the `DistanceSensor` class provides a distance property which allows the system to read the measured distance directly and avoid the calculation.

The code used is below in Fig. 7.13:

The `DistanceSensor` class provided two different mechanisms for obtaining a distance reading (both of which are given as sample code in the lab notes):

- A distance property that will immediately return a current distance reading from sensor; and
- `when_in_range` and `when_out_of_range` properties, that can be set to user functions that are called each time the configured `threshold_distance` property is crossed.

bluecam.py depth.py

```
from bluedot import BlueDot # Import Bluedot library
from signal import pause # Import pause function from signal
from time import sleep # Import sleep function from time
from picamera import PiCamera # Import camera library
from gpiozero import LED # Import LED library
from gpiozero import DistanceSensor # Import Ultrasonic library
from datetime import datetime # Import datetime for filename formatting

camera = PiCamera() # Create camera instance
camera.resolution = (640, 480) # Reduce resolution
camera.rotation = 180 # Rotate to correct orientation

sensor = DistanceSensor(20, 21) # Define ultrasonic pin connections (Echo, Trigger)
LED1 = LED(19); # Define LED connected to pin 19
LED2 = LED(26); # Define LED connected to pin 26

def pressed_1(pos): # Function to execute when button 1 pressed
    print("LED 1") # Print to console for feedback
    LED1.on() # LED 1 on

def pressed_2(pos): # Function to execute when button 2 pressed
    print("LED 2") # Print to console for feedback
    LED2.on() # LED 2 on

def pressed_3(pos): # Function to execute when button 3 pressed
    print('Distance to nearest object is', sensor.distance, 'm')

def pressed_4(pos): # Function to execute when button 4 pressed
    camera.start_preview(fullscreen=False, window=(100, 20, 640, 480)) #Reduce size
    sleep(2) # Sleep short time to avoid freezing camera
    print("Taking picture") # Print to console for feedback
    camera.capture("/home/pi/Desktop/Android_App%s.jpeg" %"{0:%Y}-{0:%m}-{0:%d}:{0:%S}".format(datetime.now()))
    # Save as Android_AppY-M-D-S.jpeg
    camera.stop_preview() # Close camera preview window

bd = BlueDot(rows=2, cols=2) # Setup a 2x2 table of buttons
bd[0,0].color = "red" # Top Left button
bd[0,1].color = "green" # Bottom Left button
bd[1,0].color = "yellow" # Top Right button
bd[0,0].when_pressed = pressed_1 # Define function to run for button 1
bd[0,1].when_pressed = pressed_2 # Define function to run for button 2
bd[1,0].when_pressed = pressed_3 # Define function to run for button 3
bd[1,1].when_pressed = pressed_4 # Define function to run for button 4
pause() # Process sleeps until signal is received then signal handler called

```

Fig. 7.13 Screenshot of code for depth measurement task

The final section of this chapter covers integrating a Rasberry Pi with two LED's, an Ultrasonic sensor and a camera to a mobile phone running the Bluedot application. It will setup a screen of 4 buttons: the 1st will turn on LED 1, the 2nd will turn on LED 2, the 3rd will read the Ultrasonic sensor distance and display it to the console and the 4th will capture a photograph. This system simulates a Depth Measurement System for a water tank.

The LEDs are connected to pins 19 and 26 through a resistor then to ground. The Ultrasonic sensor is connected to 5 V on the VCC pin, ground on the GND pin, RPi pin 20 to the ECHO pin, and, RPi pin 21 to the TRIG pin. The camera ribbon is plugged into the camera ribbon input on the RPi. Once the hardware and software were setup, the mobile phone application looked like the Fig. 7.14.

This system successfully implemented a water tank Depth Measurement System for remote monitoring. The LED's can indicate the depth of the tank. The Ultrasonic sensor can provide a reading of the depth of the tank. The volume of water remaining in the tank can be calculated form the measured depth. This is useful as a user can quickly interpret 5 L of water in a tank, compared to "37.5 cm" of water for example. The camera can provide further visual confirmation, for example, a user can see that the Ultrasonic sensor indicates a full tank, but the camera provides a photograph showing an object close to the sensor giving it a false reading.

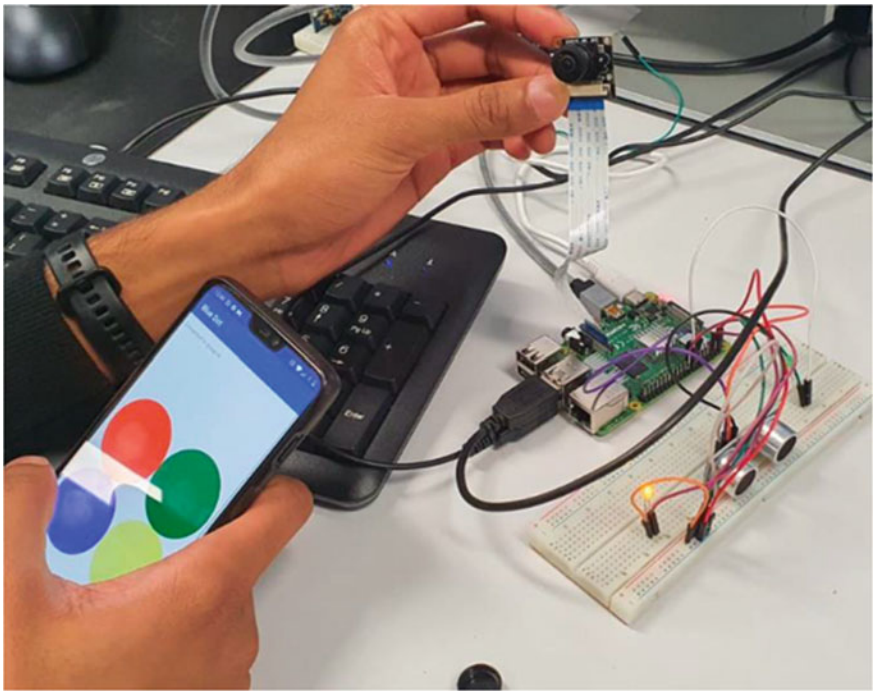


Fig. 7.14 RPi setup for depth measurement and mobile phone running BlueDot app

Further refinements of this system could be, making it actually waterproof in a casing and testing the accuracy of reading water level with an Ultrasonic sensor.

Due to the fact that this standard is operating in 2.4 GHz ISM band, interference with other communication protocol like the WiFi cannot be fully avoided which can result to random disconnection. Class 1 Bluetooth communication ranges up to 100 m and Class 2 Bluetooth ranges up to 10 m. It is relatively very slow to WiFi, as the Bluetooth 4.0 can only transfer up to 25 megabits per second. The power consumption is also an issue especially for devices relying on battery.

With this in mind, Bluetooth might not be the primary option in terms of transferring information. However, if fully utilized, Bluetooth can be used in different applications in the IoT network.

7.8 Implementation of Bluetooth Using Arduino

In this section, the use of BLE using Arduino is highlighted and explained in detail for the Arduino Nano 33 IoT board.

For a cost-effective connected project solution, the Arduino Nano 33 IoT combines Wi-Fi and BLUETOOTH® connectivity with a low-power architecture. The Arduino Nano 33 IoT and the Arduino IoT Cloud are fully compatible. TLS secure transport is fully supported by the device. The cryptographic keys are stored in hardware on the ATECC608A crypto chip, providing a very high level of security.

To utilise the board, the user will use the ArduinoBLE library provided by Arduino in this chapter. For a detailed instruction and knowledge of this library the user may read further in this link https://www.arduino.cc/en/Reference/ArduinoBLE.

In this library there are mainly two types of configurations: Central and Peripheral. In simple terms 'Clients' are central devices. They use peripheral devices to read and write data. Servers are peripheral devices. They give read/writable characteristics to operate actuators such as motors, lights, and so on, and they provide data from sensors as readable characteristics.

Services are provided by a BLE peripheral, custom services can be defined, or standard services can be used. UUIDs (universally unique identifier), or universally unique identifiers, are used to identify services. A 16-bit UUID is used for standard services, while a 128-bit UUID is used for specialized services. The radio that is being used, and its firmware determine the capacity to define services and characteristics.

The Fundamental logic of the ArduinoBLE library as illustrated in Arduino reference documents is shown in Fig. 7.15.

A central device can do four things with a characteristic: Receive the current value of the characteristic from the peripheral by using the 'Read' characteristic. Modify the value of the feature by writing by using the 'Write' characteristic.

Frequently used for tasks that are similar to instructions, such as directing the peripheral to switch on or off a interfaced component. To instruct the peripheral to communicate updated values of the characteristic without requiring the central to inquire for them on a regular basis by using the 'Indicate' or 'Notify' characteristic.

7.9 Guidelines Given to Implement the Project

In this section, the following guidelines will help to implement, Bluetooth using the nrF application for iOS and Android for Nano 33 IoT connectivity.

Task 1: Implementing BLE using Arduino Nano 33 IoT board.

In task 1, the user will commence with the first transmission module for this unit i.e. Bluetooth.

For this, the board used is Arduino Nano 33 IoT, pin configuration of the board is given in the Figs. 7.16 and 7.17. in the IDE.

For port detection of the Board, it is important to press the reset button fast to make the whole system reset. For connections it is good to check FirmwareUpdate.

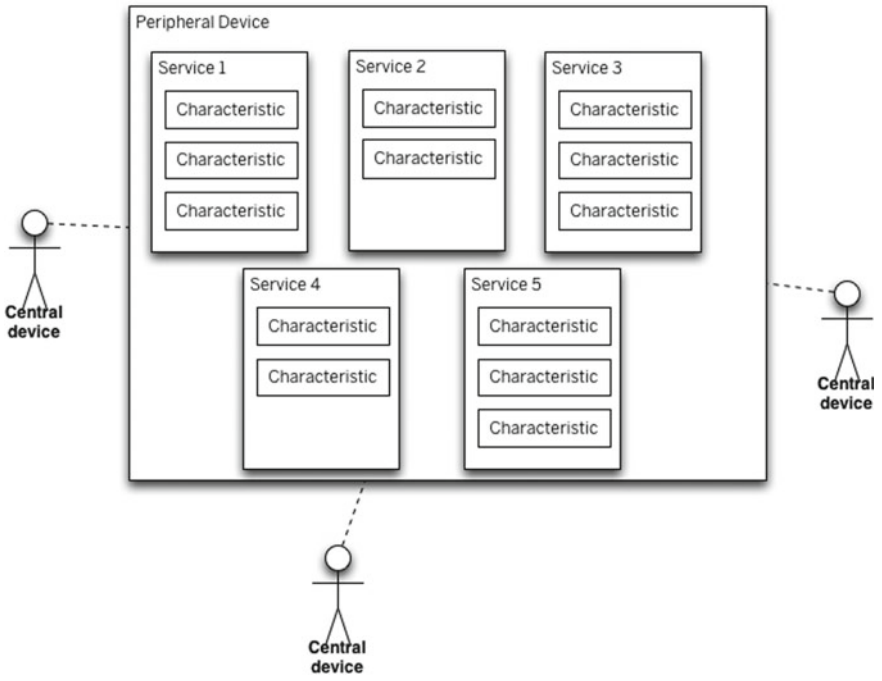


Fig. 7.15 The Fundamental logic of the ArduinoBLE library as illustrated in Arduino reference documents

1. Install and understand the library

To configure the Bluetooth in this board, the user will use ArduinoBLE library. It can be downloaded from the Arduino IDE > Manage Libraries > ArduinoBLE click install as shown in Fig. 7.18.

```
Syntax: #include <ArduinoBLE.h>
```

2. Implementing LED control using the peripherals example

Once the library is installed, select the ArduinoBLE Examples > Peripherals > LED.

In this code, the name is modified to avoid any confusion.

```
BLE.setLocalName("Your_cohort_GroupNumber");
```

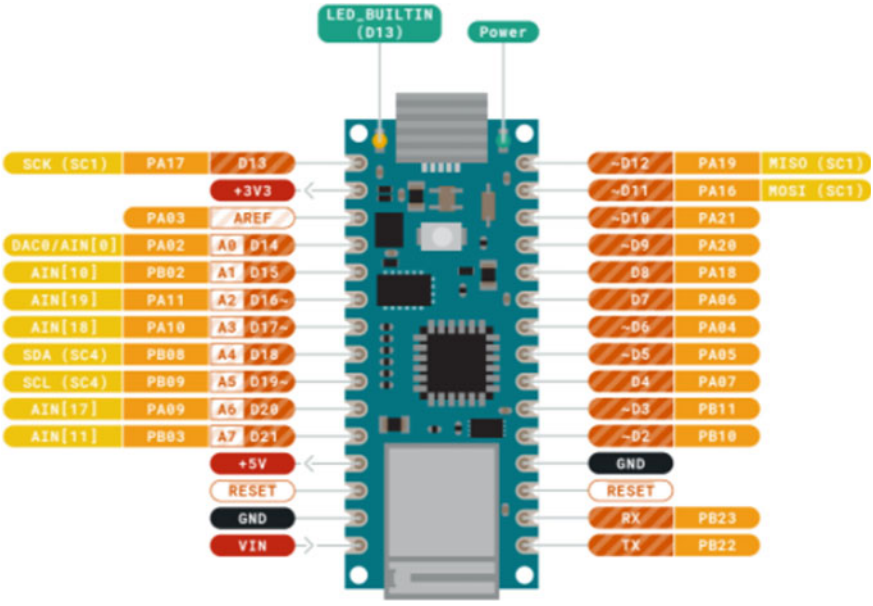


Fig. 7.16 The fundamental block diagram of a microprocessor unit

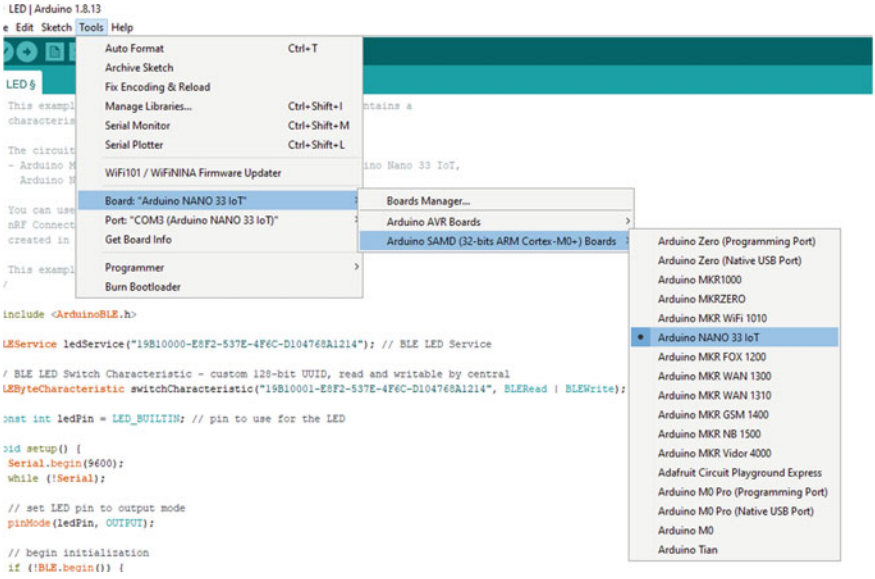


Fig. 7.17 The screenshot of the IDE providing the path to select the appropriate board for configuration

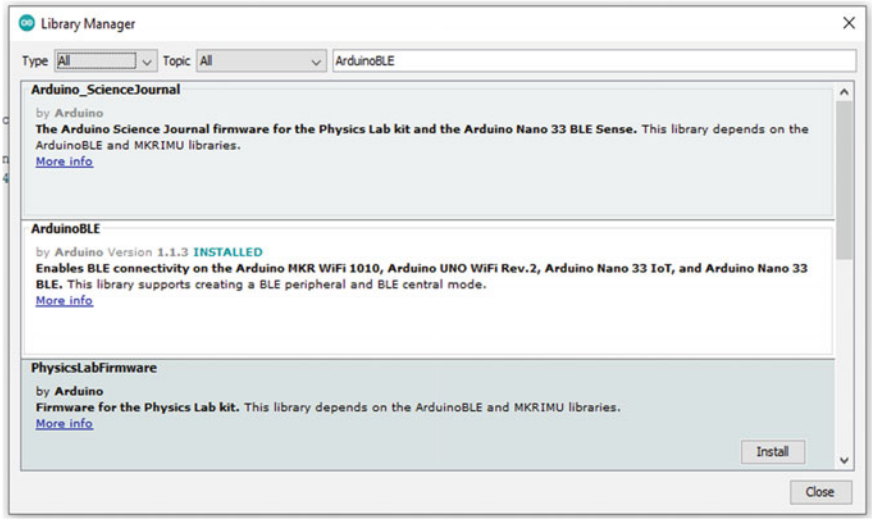


Fig. 7.18 The screenshot of the library manager illustrating the library to be downloaded

Once modified, the user can now run this code by uploading it to the Nano 33 IoT board. Next, in the phone, the user opens the nRF app (download link for iOS and Android) once Bluetooth is turned on, scan for devices as observed in Fig. 7.19.

The device will now reflect on this app like the image below as Fig. 7.20.

The user then clicks on connect, once connected, checks the serial monitor for such a result as depicted in Fig. 7.21.

The next stage is to begin to control the inbuilt LED on board. For this, the user clicks on the device name after it has been connected. The device overview will look like this Fig. 7.22 illustrated below.

The user clicks on the Client Tab on the top and will be redirected to the attribute table, click on the arrow pointing upward as illustrated in the Fig. 7.23, to start sending control commands:

The user can control the inbuilt LED on board. For this, click on the Bool and select True and click Write. Alternatively, click on Bool > False > Write, as depicted in Fig. 7.24.

The builtin LED will turn on and off respectively:

The serial monitor results are reflected in Fig. 7.25:

This is a basic control of the BLE in the Arduino Nano 33 IoT, more features can be explored and implemented.

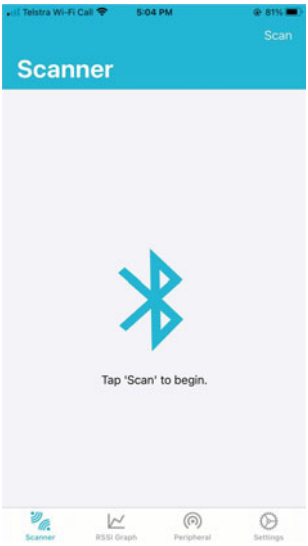


Fig. 7.19 The screenshot of the scanner on the nrF application in iPhone

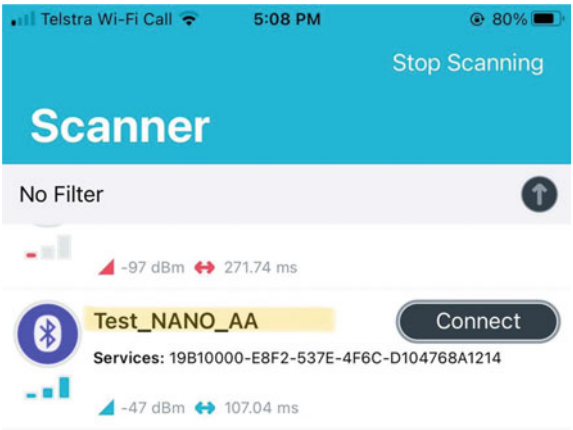


Fig. 7.20 The screenshot of the scanner on the application highlighting the desired board

7.10 Outcome of Project Implementation—Arduino Nano 33

The following section shows the implementation of the tasks provided.

The first part of this report will cover wireless Bluetooth communication between an Arduino Nano and a mobile phone. Using a mobile phone, it was possible to connect to the Arduino Nano using an application called nRF. This enabled us to turn the on-board LED on and off remotely. This simple example showcases the flexibility and power of Bluetooth and a communication protocol in IoT projects.

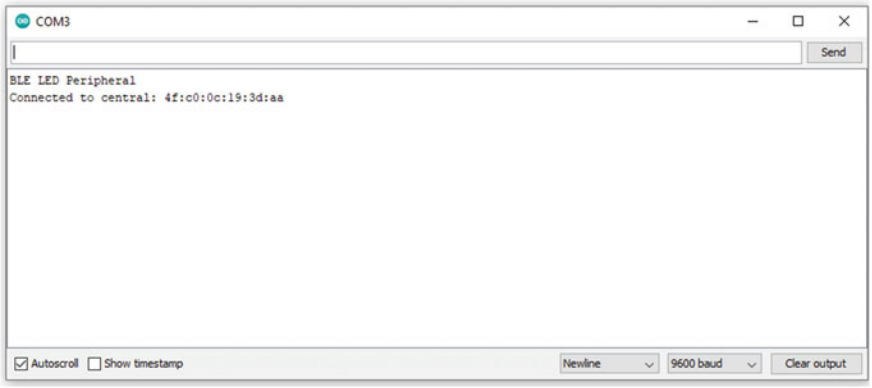
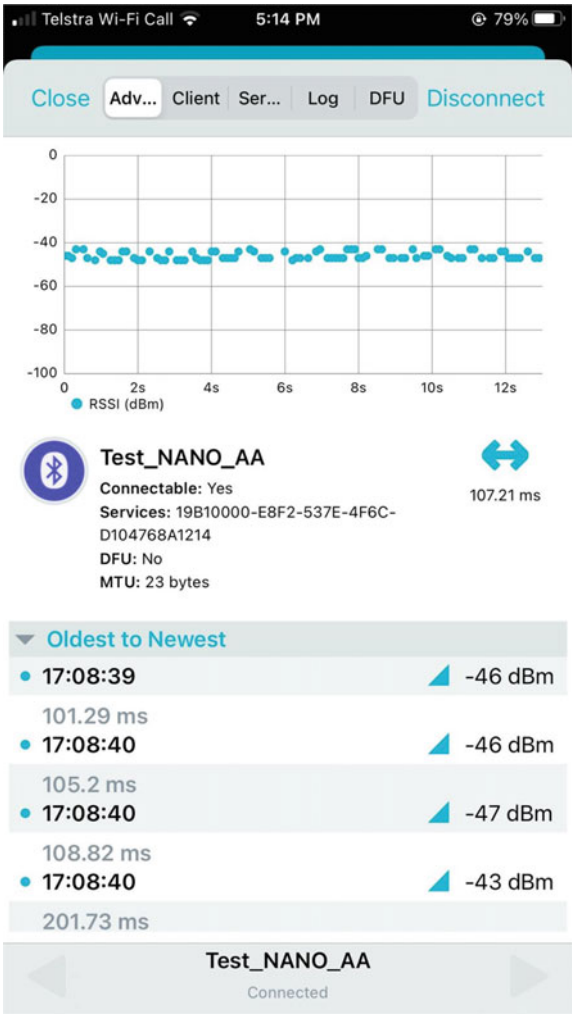


Fig. 7.21 The screenshot of the application illustrating connection to the BLE app

Fig. 7.22 The screenshot of the nRF application on iPhone showing connected board



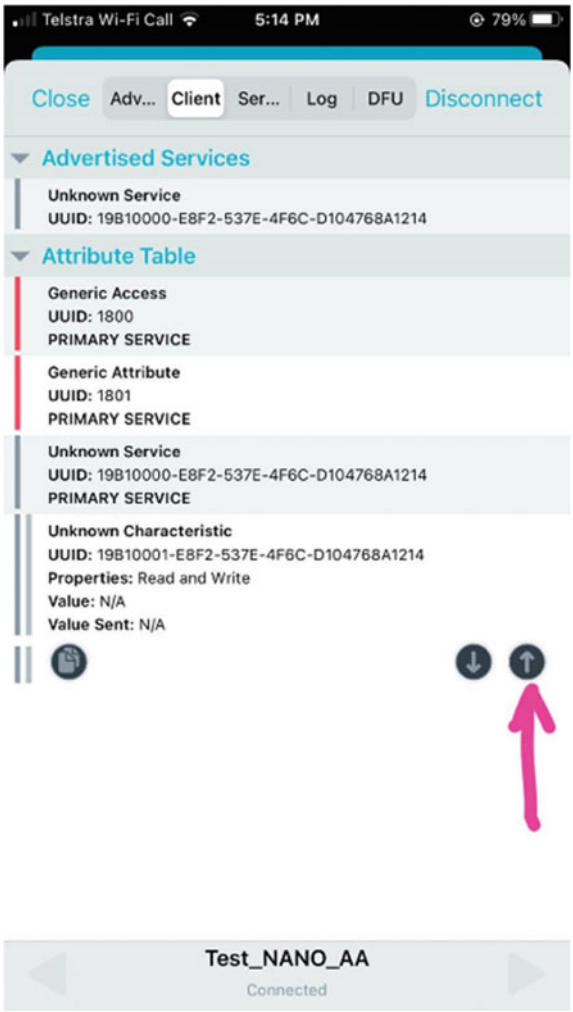


Fig. 7.23 The screenshot of the nRF application illustrating the path to send commands

Part 1—Implementing Bluetooth on an Arduino Nano 33 IoT board

To implement a Bluetooth connection between an Arduino Nano and a mobile phone, first they installed an application called 'nRF' on a phone. nRF allows the phone to scan for and connect to devices broadcasting a Bluetooth connection. They also installed ArduinoBLE in the Arduino Integrated Development Environment (IDE), which allowed us to use the Bluetooth library. Below is the Arduino code that they used:

```
#include <ArduinoBLE.h>
const int ledPin = LED_BUILTIN; // Pin to use for the LED

BLEService ledService("19B10000-E8F2-537E-4F6CD104768A1214"); // BLE
//LED Service

BLEByteCharacteristic  switchCharacteristic("19B10001-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite); // 128-bit UUID, read and writable
//by central

void setup() {
  Serial.begin(9600); // Setup serial port for communication
  while (!Serial); // Wait for serial communication
  pinMode(ledPin, OUTPUT); // set LED pin to output mode
  if (!BLE.begin()) { // Begin initialization
  Serial.println("starting BLE failed!");
  while (1);
  }
  BLE.setLocalName("4201_Group1"); // Set advertised local name
  BLE.setAdvertisedService(ledService); // Set advertised service UUID
  ledService.addCharacteristic(switchCharacteristic); // add charac-
  teristic
  BLE.addService(ledService); // add service
  switchCharacteristic.writeValue(0);   //   Initial   value   for
  characeristic:
  BLE.advertise(); // Start advertising
  Serial.println("BLE LED Peripheral"); // Print to serial monitor
  }

void loop() {
  BLEDevice central = BLE.central(); // listen for BLE peripherals to
  //connect:
  // if a central is connected to peripheral:
  if (central) {
    Serial.print("Connected to central: "); // print the central's MAC
    //address:
    Serial.println(central.address()); // print the central's MAC ad-
    dress:
    while (central.connected()) { // While central is connected to
    peripheral:
      // If remote device wrote to characteristic, use value to control
      //LED:
      if (switchCharacteristic.written()) {
        if (switchCharacteristic.value()) {
        // any value other than 0
```

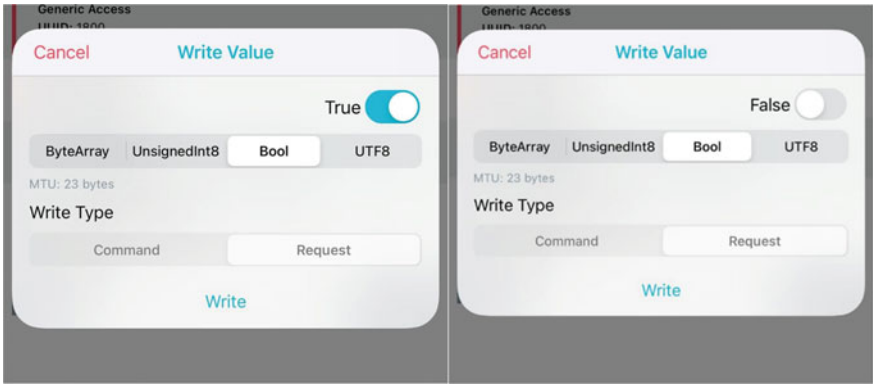


Fig. 7.24 The screenshot of the nRF application in iPhone showing the controls in Boolean form

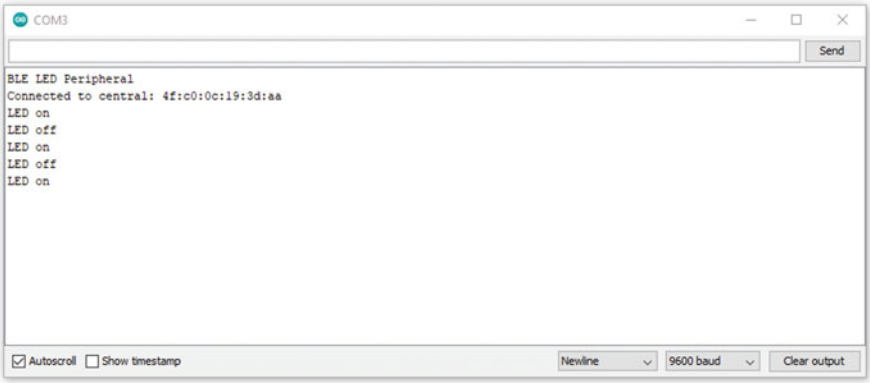


Fig. 7.25 The screenshot of the serial monitor illustrating the expected results in LED control

```
      Serial.println("LED on");
      digitalWrite(ledPin, HIGH); // will turn the LED on
      } else { // a 0 value
        Serial.println(F("LED off"));
        digitalWrite(ledPin, LOW); // will turn the LED off
        }
      }
    }
    Serial.print(F("Disconnected from central: "));
    // when central disconnects, print it
    Serial.println(central.address());
    }
}
```

Once this was running on the Arduino Nano, the user connected their phone via Bluetooth in the nRF application. Below in Fig. 7.26 is a screenshot of the connection:

There is a lot of useful information here regarding the connection status including: the Received Signal Strength Indicator (RSSI), discoverability, connection type and connection delay. This information allows for better troubleshooting, for example, if the RSSI is around -100 dBm (which indicates a very poor signal strength) there may be a physical object blocking the signal or a hardware fault in the antenna.

Using the nRF application, they sent a '1' to the Arduino Nano, this turned on the LED. Then they sent a '0', which turned off the LED. A screenshot of sending a '1' is below as illustrated in Fig. 7.27.

The serial monitor provides feedback as seen in Fig. 7.29. on what is occurring, below is the initialisation of the Bluetooth service, a connection to a device and its MAC address, receiving a '1' to turn on the LED and receiving a '0' to turn off the builtin LED as seen in Fig. 7.28.

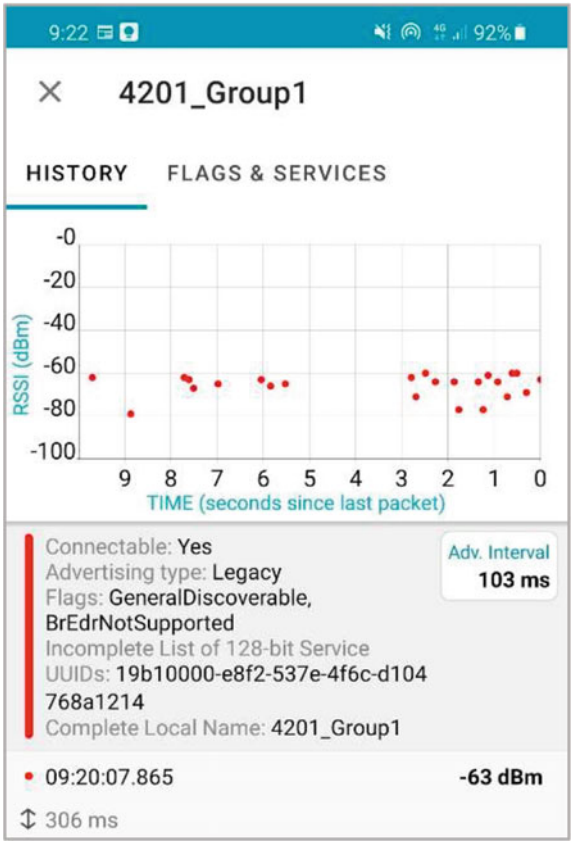


Fig. 7.26 The screenshot of the nRF application page displaying the connected board

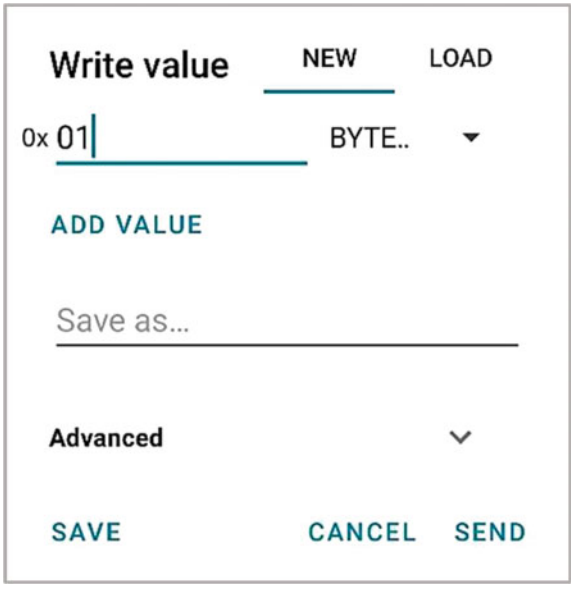


Fig. 7.27 The screenshot of the application while sending '1' to turn the LED 'on'

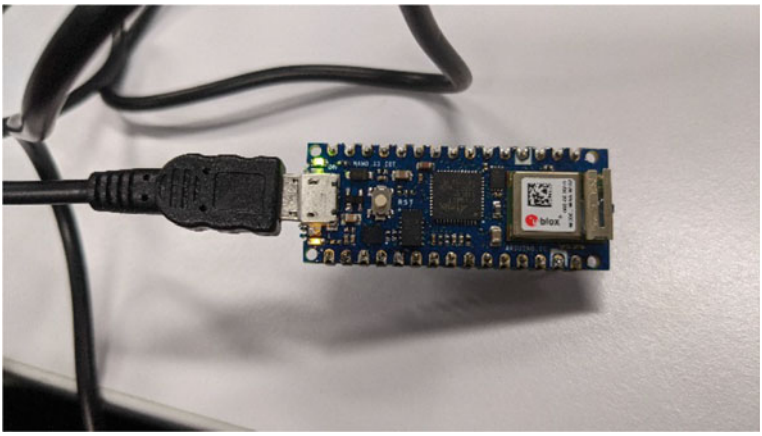


Fig. 7.28 The image of the Nano 33 board illustrating the builtin LED 'on'

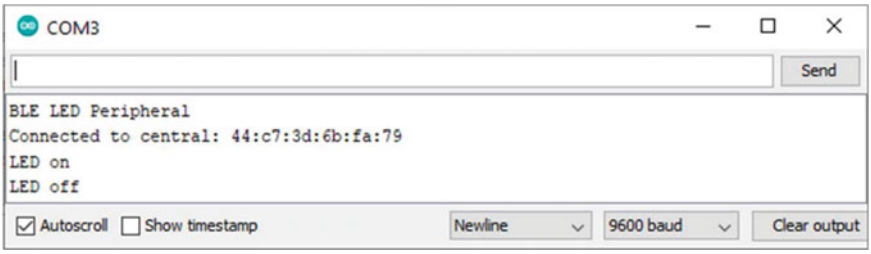


Fig. 7.29 The screenshot of the serial monitor providing feedback on the LED control

Suggested Reading

1. S.C. Mukhopadhyay, N.K. Suryadevara, in *Internet of Things: Challenges and Opportunities*, by S, Mukhopadhyay (eds.), Internet of Things. Smart Sensors, Measurement and Instrumentation, vol. 9 (Springer, Cham, 2014). https://doi.org/10.1007/978-3-319-04223-7_1
2. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement **178**, 109424 (2021)
3. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
4. D. Thomas, R. Shankaran, M. Orgun, S. Mukhopadhyay, A secure barrier coverage scheduling framework for WSN-based IoT applications. IEEE Trans. Green Commun. Netw. (2021)
5. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the Internet of Things and serious games. Internet Things **13**, 100332 (2021)

6. S.C. Mukhopadhyay, S.K.S. Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S, Zeadally, Artificial Intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
7. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
8. S. Pal, M. Hitchens, T. Rabehaja, S.C. Mukhopadhyay, Security requirements for the internet of things: a systematic approach. MDPI Sens. 5897 (2020)
9. F. Akhter, S. Khadivizand, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
10. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using Graphite/PDMS sensors. Sens. Actuators, A **286**, 43–50 (2019)
11. M.E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417
12. N.K. Suryadevara, S.C. Mukhopadhyay, in *Smart Homes: Design, Implementation and Issues*. Smart Sensors, Measurements and Instrumentation, vol. 14 (Springer, 2015). ISBN: 978-3-319-13556-4
13. R. Yan, X. Chen, S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 26. Structural Health Monitoring: An Advanced Signal Processing Perspective (Springer, 2017). ISBN: 978–3-319-56125-7
14. S.C. Mukhopadhyay, A. Mason, in *Smart Sensors, Measurement and Instrumentation*, vol. 4. Smart Sensors for Real-Time Water Quality Monitoring (Springer, 2013). ISBN: 978-3-642-37005-2-1
15. S.C. Mukhopadhyay, in *Lecture Notes in Electrical Engineering* vol 96. New Developments in Sensing Technology for Structural Health Monitoring (Springer, 2011). ISBN: 978-3-642-21098-3
16. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors, https://doi.org/10.3390/s150510350
17. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045

Chapter 8
LoRa Communication Based IoT System

8.1 Introduction

The chirp spread spectrum (CSS)-based spread spectrum modulation technology is known as LoRa (short for **Lo**ng **Ra**nge). Due to its superior energy consumption, connection durability, and long-range capabilities at the expense of low bit rates, the LoRa (Low-Power Long-Range) modulation technology is a suitable choice for many Internet-of-things (IoT) applications. The carrier frequency of a sinusoid is linearly modulated across a particular bandwidth in LoRa, which is a modified type of chirp spread spectrum (CSS) modulation. As a result, chirps are produced, which can be distinguished by their starting frequencies. The LoRaWAN ecosystem is made up of the LoRa Architecture's device and sensor chip makers, server and base station vendors, and other service providers as shown in Fig. 8.1.

These are the potential list of advantages demonstrating why LoRa and LoRaWAN are two of the most extensively used LPWAN technologies.

Energy Consumption: LoRaWAN consumes very little energy, allowing devices to last longer on their batteries.

Range: LoRaWAN has a range of about 5 km in urban areas and up to 15 km in suburban areas. LoRa radio waves can also penetrate buildings well, passing across obstacles like lifts and basements to reach sensors.

Open Source: LoRa is an open-source protocol that incorporates all community best practices. The other advantages are cost, two-way communication, localisation etc.

8.2 Description

In this section, a brief description on the objectives of this chapter and the hardware and software used for implementation of LoRa.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_8

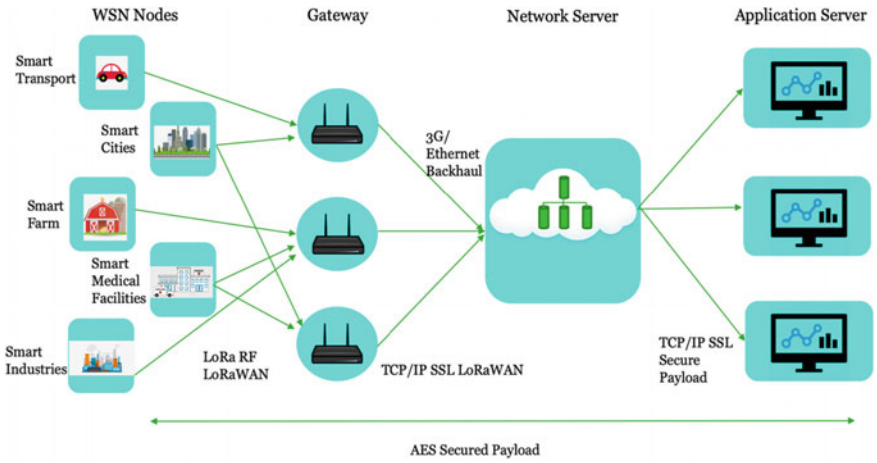


Fig. 8.1 The architecture of sensor nodes based on LoRa in an IoT system

8.2.1 *Objectives*

The major aim of this section is to provide a broader understanding of the MKR1300 Arduino board and how to program it. The points further elaborate on these aims:

- Understanding the pinouts available in the MKR1300 board and how to interface sensors and calibrate them.
- Illustrating a sample guideline sheet to implement at a workshop for peer-to-peer communication for the MKR1300 board.
- Using LoRa in an IoT system to connect to a cloud platform.

8.2.2 *Hardware*

The hardware components used in this chapter primarily involve the μC boards illustrated earlier in this chapter and the sensors interfaced. The following components are required to build this system:

- Microcontroller:
  - Arduino MKR1300
- Micro USB cable:
  - Arduino MKR WAN 1300 compatible

- Sensors:
  - PIR Sensor Module- Motion Sensing
  - DHT (temperature and humidity sensor)
  - Ultrasonic sensor
  - LEDs
- Prototyping Solderless Breadboard
- Jumper Cables
- Desktop/Laptop.

8.2.3 Software/Applications

The following software can be used to program the boards:

Arduino IDE: (Compatible with Arduino MKR1300)

8.3 LoRa Implementation Using Arduino

The Arduino MKR WAN 1300 is a learning and development board that includes the ATMEL SAMD21 microcontroller, which has been developed to combine the core's low power consumption and high performance with the Arduino's ease of use. The user installs by selecting Boards > Boards manager and search for "MKR" to install the MKR series of boards. The first given option is Arduino SAMD Boards (32-bit ARM Cortex-M0+), which includes the MKR WAN 1300. The Fig. 8.2. illustrates the sender and receiver code from the MKRWAN library.

The LoRaWAN is an intriguing Internet-of-Things (IoT) technology that is rapidly expanding our connectivity capabilities. Without the use of a cell network or a high-powered antenna array, LoRaWAN allows devices to communicate over distances of tens or even hundreds of kms. The major disadvantage of this technology is that data speeds are significantly limited—in the low kb/s range depending on the circumstances. For intermittent sensor data scattered across a large geographic area, LoRaWAN remains an effective solution, however, it might not be effective to transmit high quality video or audio streaming.

LoRa_Send

```
#include <SPI.h>
#include <LoRa.h>

int counter = 0;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("LoRa Sender");

  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
}

void loop() {
  Serial.print("Sending packet: ");
  Serial.println(counter);

  // send packet
  LoRa.beginPacket();
  LoRa.print("hello ");
  LoRa.print(counter);
  LoRa.endPacket();

  counter++;

  delay(5000);
}
```

LoRa_Receive

```
#include <SPI.h>
#include <LoRa.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("LoRa Receiver");

  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
}

void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // received a packet
    Serial.print("Received packet '");

    // read packet
    while (LoRa.available()) {
      Serial.print((char)LoRa.read());
    }

    // print RSSI of packet
    Serial.print("' with RSSI ");
    Serial.println(LoRa.packetRssi());
  }
}
```

Fig. 8.2 The IDE screenshot (from left to right) the sender and receiver code for MKR1300 boards

8.4 Guidelines to Users

This section discusses the guidelines provided to the users to implement IoT project based on library configurations and basic coding, thus enabling to understand the code for the header files. The user will use LoRa to understand inter-device communication and to transmit data and control the interfaced peripherals.

Task 1: Implementing LoRa using Arduino MK1300 board

In this task the user will investigate and configure the MKR1300 and interface sensor to display results on the serial monitor of another group using MKR1300.

To configure the MKR1300 board.

- The user selects the board type and port that is connected to the MKR1300
- Further, the user checks if the board is connected and can be configured using the IDE (use any code from the examples- basics) [optional step]
- The user then configures the board and note the value of the Device EUI and the Frequency of operation.

Once the user has configured the MKR1300 board.

The users are now required to interface the MKR1300 with the PIR sensor and transmit the data to another MKR1300 board using LoRa.

The user will need to show their data on the Serial Monitor on their PC as well as the group the user will work with, as one of the MKR1300 will be the sender and the other will be the receiver. Their output is shown in the Fig. 8.3.

The users will use the same Frequency in the sender and receiver code for the MKR1300 device amongst two groups to communicate, as seen in Fig. 8.3, else, there can be multiple receivers with the same frequency.

The user can view this sample code for LoRaSender and LoRaReciever as given in the repository in GitHub: https://github.com/arduino-LoRa/examples and has been discussed in the earlier Sect. 8.3. This repository can be used and is an interesting read to understand the different applications for LoRa communication using Arduino.

Task 2: Implementing Lab 1 Transmission to AdaFruit I/O using TTN

In this task, the user will integrate the MKR1300 board with 3 sensors, DHT22, PIR and Ultrasonic. The stages of implementation are given below.

Stage 1—Microcontroller and Sensor Interface

The first stage discusses the microcontroller and sensors interfacing. Interface the Arduino with the three sensors provided. Connect an LED to check the sensors working.

- DHT22 sensor
- PIR sensor
- Ultrasonic Sensor

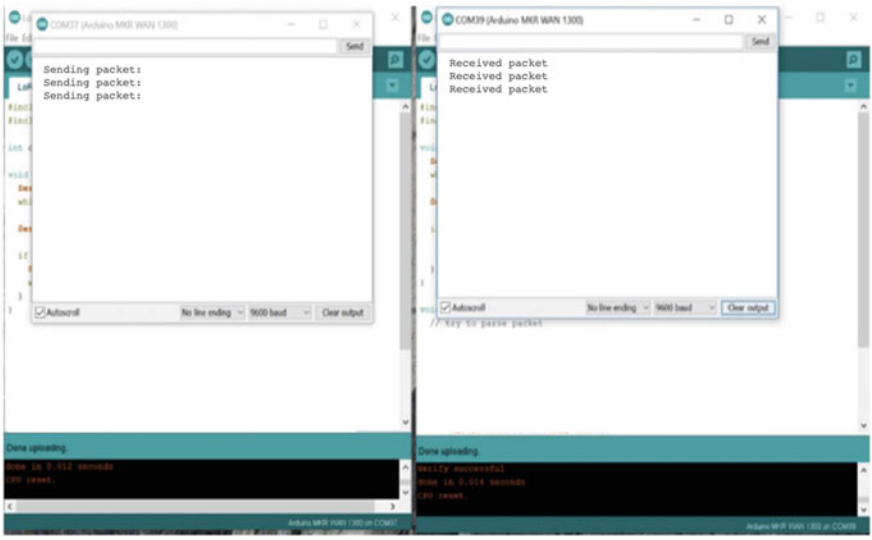


Fig. 8.3 The screenshot of the serial monitor with the sender and receiver code output for LoRa

The tasks for the design involve:

- Connect the Arduino to the PC and run the LED blink code. (Note: The user is using an MKR1300 board, therefore the user can add the necessary board packages on Arduino IDE).
- See the datasheet and connect the sensors appropriately.
- Interface the PIR Sensor to the Arduino and check its working on the serial monitor.
- Next, interface only the DHT22 sensor, add the necessary libraries, and check the results on the serial monitor.
- Finally, interface the Ultrasonic Sensor and check the results on the serial monitor.

Stage 2—Serial Monitor Results

After stage 1 is complete, we move to the next stage.

- Next, in stage 2, all the three sensors to the Arduino together using the jumper cables and breadboard are connected.
- In the Arduino program, update the code with the necessary changes to enable all three sensors working together and displaying data every 25 s.
- The outputs of the three sensors working on the serial monitor of the Arduino IDE are displayed.
- A working node will have outputs the reading of the temperature in degree Celsius/ Fahrenheit, Humidity in percentage, PIR state in binary value i.e. 0 or 1, and distance in cm or mm.

Stage 3—Configuration of the Transmission module

In this stage, the user will configure the LoRa module in the MKR1300. The user will use TTN to transmit Data and connect the gateway.

1. Getting the Device ID:

To configure the gateway settings, the user requires the unique device ID for their board.

Thus, they need to determine the DeviceEUI for the MKR1300 their group uses. This result will be displayed on the serial monitor.

2. Setting up TTN:

The following instructions are the steps to set up the TTN application:

1. The user visits https://www.thethingsnetwork.org/
2. The user logs in with the following details (Example Username: IoT_System_Design Password: Engg401/801)
3. The user clicks on console on the top-right corner
4. The user clicks on Applications
5. The user clicks on +add application
6. Then writes the group number sample format (ex: group2)

7. The Description can include: project Name
8. The user picks the handler ==> ttn-handler-asia-se (for this instance)
9. The user clicks on add application

Once completed, the application will look like the Fig. 8.4 given below.

3. Registration of The Device:

Once, every group has registered and created an application. The user now registers their device in their respective application. (i.e. 1 application has 1 registered device).

Follow these instructions:

1. The user selects set application for e.g. 'group2'
2. The user clicks on overview on the top-right corner
3. The user Scrolls down, to click on + register device
4. Then writes a unique device name e.g. node_1, and copy the DeviceEUI from the previous step on the given tab and register.

Once registered, the device overview is shown in Fig. 8.5.

4. First Communication:

After the user registers the device, they have to now check the communication. The user will use the appEUI and appKey to transmit the data from the registered device.

The user can run a 'Hello World' payload using the Arduino IDE and check via Serial monitor and the application page on TTN to validate this communication.

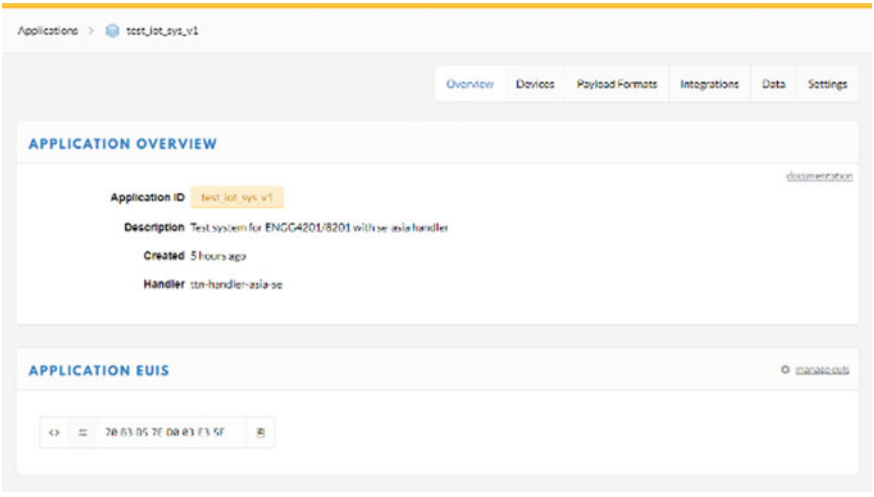


Fig. 8.4 The screenshot of the TTN console, application overview

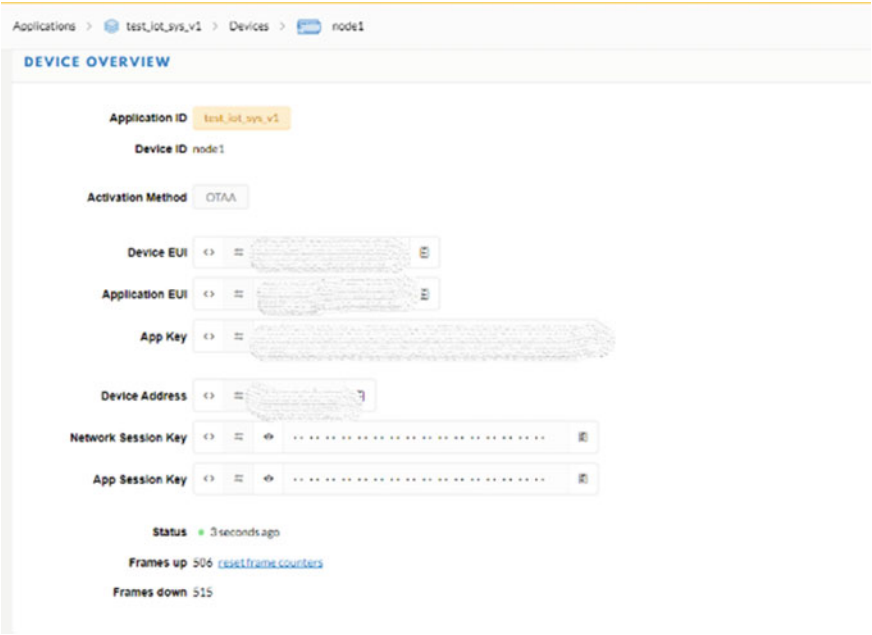


Fig. 8.5 The screenshot of TTN console for device overview

5. Payload Assignment:

This is an important task. In TTN, the users have the choice to assign a 16/32-bit payload, it is set to a default value of 32 bit.

The users need to determine which payload format they prefer to use. (Suggestion: Cayenne LPP, will be good to begin with!)

Once it is done, it can be modified in their code on the Arduino IDE as required. Once the users have assigned the sensor data to the payload they need to configure the payload decoder on TTN.

For this the user can visit the Payload formats on TTN under their application on the top right of their screen.

Further, in the decoder section, the user may now write the decoder function to interpret the data that is sent using the payload they have used.

Depending on the payload the users are using, they may have to configure their decoder bytes accordingly.

6. Sensor Data on TTN:

The users can upload the code from IDE with a set timer (i.e. how frequently they want to send the data), that will transmit the data to the TTN server, where it can be viewed in the 'Data' Tab on the top right of their application.

The user will have a result that looks like this in Fig. 8.6.

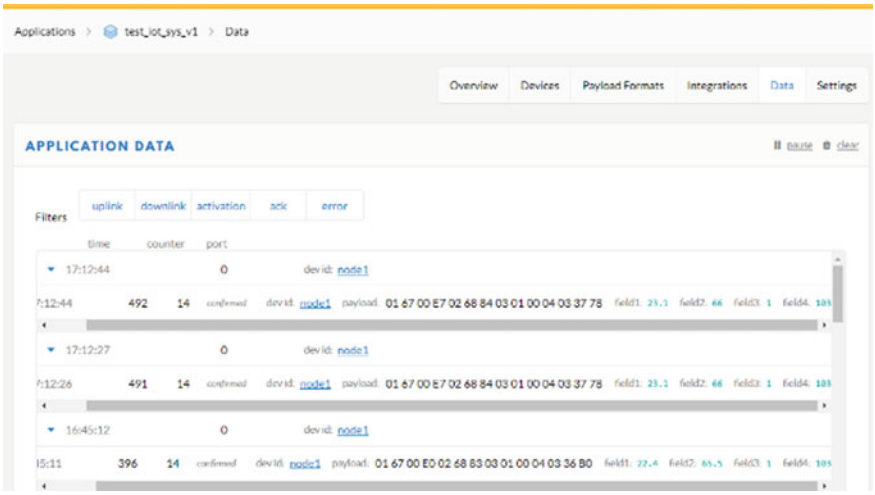


Fig. 8.6 The screenshot of the TTN console for the application data showing the payload format

Stage 4—Transmission of data to cloud and visualisation

In this stage, the user will now transmit their data from TTN to a cloud server i.e. AdaFruit I/O. This given as a bonus task.

8.5 Outcome of the Project Implementation

This section aims to illustrate the work done by the users while implementing the task in 8.4.

This first part of this report will show how to communicate between two separate Arduino MKR1300WAN boards. The communication protocol used was Long Range Wide Area Network (LoRaWAN), the payload was an integer counter for the packet ID and a Boolean value for the status of a Passive InfraRed (PIR) sensor. In this practical the student group (group 1) acted as the sender of information and group 4 acted as the receiver. They encountered a few problems with establishing a connection and then with corrupted data which will be explained.

Part 1—Implementing LoRa using an Arduino MKR1300WAN board

The student group began by designing the following circuit, a relatively simple setup with one PIR sensor connected to pin 3 on the Arduino MKR1300WAN board as shown below in Fig. 8.7.

In the Arduino IDE, they installed the MKR and LoRa libraries. They tried to run the basic example programs to test it was working, however they received error messages that LoRa failed each time as shown in Fig. 8.8.

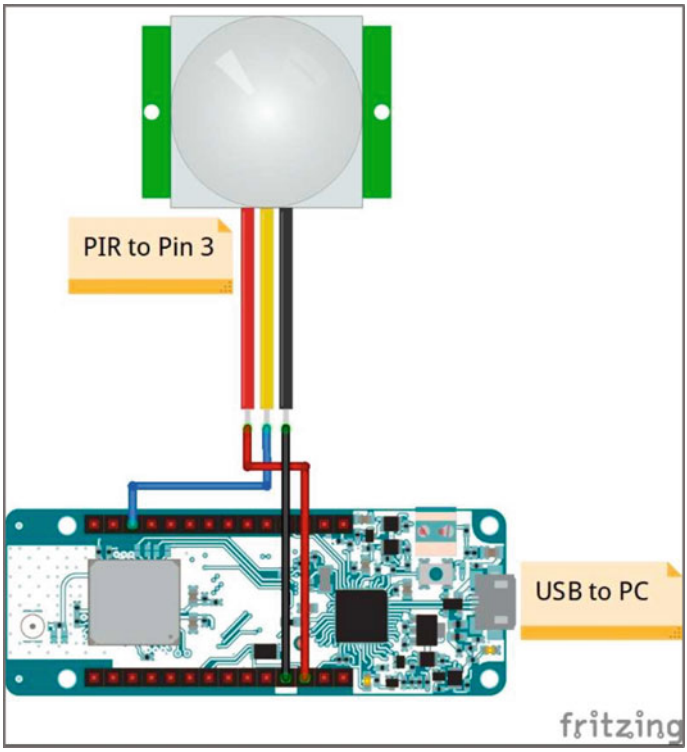


Fig. 8.7 The circuit diagram for the interface with the MKR1300 and PIR sensor

Fig. 8.8 The screenshot of Serial monitor for the connection failures with LoRa

From previous practical sessions the student group knew that this occurred sometimes when the firmware was not the latest version, problems can arise. They updated the firmware using the Arduino IDE and loaded the LoRaSender sketch. One point here is to get the frequency correct, which is done using the LoRa begin

(frequency) command. The frequency in the screenshot below is '915E6' which is scientific notation for 915 × (10^6) which equates to 915 MHz. Each country/region has a specific frequency spectrum available for use and Australia has the range 916.8 MHz through to 918.2 MHz. They began with using the 915 MHz for a quick test which is depicted below in Fig. 8.9:

Fig. 8.9 The screenshot of the Arduino sender code and results in serial monitor

They adapted the frequency to avoid collisions with other groups and packet payload slightly to make it easier to read on the receiving end, to the following:

```
#include <SPI.h>
#include <LoRa.h>
int counter = 0;
int pirSensorPin = 3;
void setup() {
pinMode(pirSensorPin, INPUT);
Serial.begin(9600);
while (!Serial);
Serial.println("LoRa Sender");
if (!LoRa.begin(916.9E6)) {
Serial.println("Starting LoRa failed!");
while (1);
}
}
void loop() {
Serial.print("Sending packet: ");
Serial.println(counter);
// send packet
LoRa.beginPacket();
LoRa.print("Hello Group 4, from Group 1, Counter = ");
LoRa.print(counter);
LoRa.print(", PIR = ");
LoRa.println(digitalRead(pirSensorPin));
LoRa.endPacket();
counter++;
delay(5000);
}
```

The student group (group 1) connected with group 4, who successfully received their messages as below in Fig. 8.10:

There a few issues with group 4 receiving their data. The first was some packets were detected multiple times. For example, packets 6, 11, 14 and 18 were ‘received’

```
LoRa Receiver
Received packet 'Hello Group 4, from Group 1, Counter = 5, PIR = 0' with RSSI -47
Received packet 'Hello Group 4, from Group 1, Counter = 6, PIR = 0' with RSSI -47
Received packet 'Hello Group 4, from Group 1, Counter = 6, PIR = 0' with RSSI -47
Received packet 'Hello Group 4, from Group 1, Counter = 7, PIR = 0' with RSSI -46
Received packet 'Hello Group 4, from Group 1, Counter = 8, PIR = 0' with RSSI -44
Received packet 'Hello Group 4, from Group 1, Counter = 9, PIR = 0' with RSSI -44
Received packet 'Hello Group 4, from Group 1, Counter = 10, PIR = 0' with RSSI -44
Received packet 'Hello Group 4, from Group 1, Counter = 11, PIR = 0' with RSSI -45
Received packet 'Hello Group 4, from Group 1, Counter = 11, PIR = 0' with RSSI -45
Received packet 'Hello Group 4, from Group 1, Counter = 12, PIR = 0' with RSSI -46
Received packet 'Hello Group 4, from Group 1, Counter = 13, PIR = 0' with RSSI -46
```

Fig. 8.10 The screenshot of Group 4 receiving their (group 1) data

twice. This is despite their serial monitor showing each packet being sent once. Here it is important to distinguish each packet with a unique identifier allowing detection and subsequent ignoring of duplicate packets. It is also noteworthy that all unique packets arrived at least once, which is far better than skipping packets leading to gaps in data. The second obvious problem is the packet received between packets 22 and 23. It's payload was corrupted, possibly due to its very weak signal strength. The RSSI (Relative Signal Strength Indicator) for that packet was −112 dB, which is extremely low. This is because RSSI is a logarithmic scale and "the closer to 0 dBm, the better the signal is" (MetaGeek 2020). This corrupted packet appears to be a duplicate as 22 was prior and 23 after, which accounts for all packets.

Part 1—Conclusion

LoRaWAN is a very powerful communication protocol for IoT projects. Its strength comes from its range, which is up 5 km in urban areas and 15 km in rural areas depending in line-of-sight and obstructions. They tested their simple setup in the lab, and I look forward to doing real world testing to see the range benefits as it can be hard to picture while communicating with another group 2 m away. The above practical has provided the key foundation to enable long range IoT projects.

This second part of this report details how sensors were integrated with a microcontroller to measure distance, motion, temperature, and humidity. The student group used a HC-SR04 for distance, a Passive InfraRed detector for motion and a DHT22 for temperature and humidity. Periodically, these sensors were sampled, and the data transmitted via LoRaWAN to TTN (The Things Network). TTN was combined with a ThingSpeak channel, which received and graphically represented the data for easy analysis. This system showcases the ease of use and power of IoT (Internet of Things) which allow remote monitoring/controlling of sensors and microcontrollers.

Part 2—Implementing Lab 1 Transmission to AdaFruit I/O using TTN

Stage 1: Sensor Interfacing

Figure 8.11 illustrates the circuit used throughout the practical:

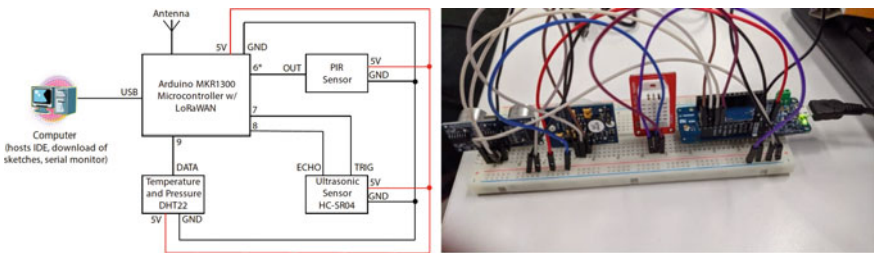


Fig. 8.11 The circuit diagram and sensor configuration

PIR Sensor Communication

The PIR sensor's value was read via the `digitalRead()` command, which returns `HIGH` when motion is detected, and `LOW` when motion is not detected:

```
void loop() {
int pirState = digitalRead(PIROUT); if (pirState == HIGH)
Serial.println("PIR: detected motion"); else
Serial.println("PIR: did not detect motion");
}
```

Ultrasonic Sensor Communication

Reading a distance value from the Ultrasonic sensor was a two-step process that involved writing data on one pin (ULTRATRIG, pin 7), and measuring the width of pulse on another (ULTRAECHO, pin 8). This pulse width was then used to calculate the distance via a simple conversion formula included within the Arduino sketch. Code for interfacing the ultrasonic sensor was taken from the Arduino Project Hub as shown in Table 8.1.

Temperature/Humidity (DHT22) Sensor Communication

The DHT22 sensor has a more complicated interface than the PIR and Ultrasonic sensors, as it presents floating point values to the microcontroller for temperature and humidity (as well as a checksum) over a single digital line. A library was available for the Arduino that provided simple API calls to retrieve the measured values for temperature and humidity without further overhead or conversions.

The methods `readTemperature()` and `readHumidity()` were invoked against the DHT object from within the loop() method in the following manner:

```
    Serial.print("AM2302: Celsius = ");
    Serial.println(dht.readTemperature());
    Serial.print("AM2302: Humidity =");
    Serial.println(dht.readHumidity());
```

Table 8.1 Distance measurements using the HC-SR04 ultrasonic sensor

Sending the pulse	Reading the distance

`digitalWrite(ULTRATRIG, LOW);`
`delayMicroseconds(2);`
`digitalWrite(ULTRATRIG, HIGH);`
`delayMicroseconds(10);`
`digitalWrite(ULTRATRIG, LOW);`
	`int duration=pulseIn(ULTRAECHO, HIGH);`
	`int distance=(duration*0.034/2);`

Stage 2: Serial Monitor Results

The PIR, Ultrasonic and DHT22 sensors were connected and all values shown on the Serial Monitor Output is shown in Fig. 8.12.

Stage 3: Configuration of the Transmission Module

Application Creation in TTN:

Creating an Application in TTN was simple, as shown in Fig. 8.13.

Register the Device

Before data could be sent from their MKR1300 device into TTN, their device needed to be registered to their TTN application. This required us to determine the EUI of their MKR1300 ("a8610a3130408112"). The "hello_world_mkr" sketch contains the code to print out the device EUI, as shown in Fig. 8.15.

After retrieving the EUI, it was a simple process to register the device with their Application via the Console, as shown in Fig. 8.14.

First Communication

After successful registration, the "appEui" and "appKey" were copied into the "hello_world_mkr" sketch, and successful packet transmission was observed, as shown in Fig. 8.15.

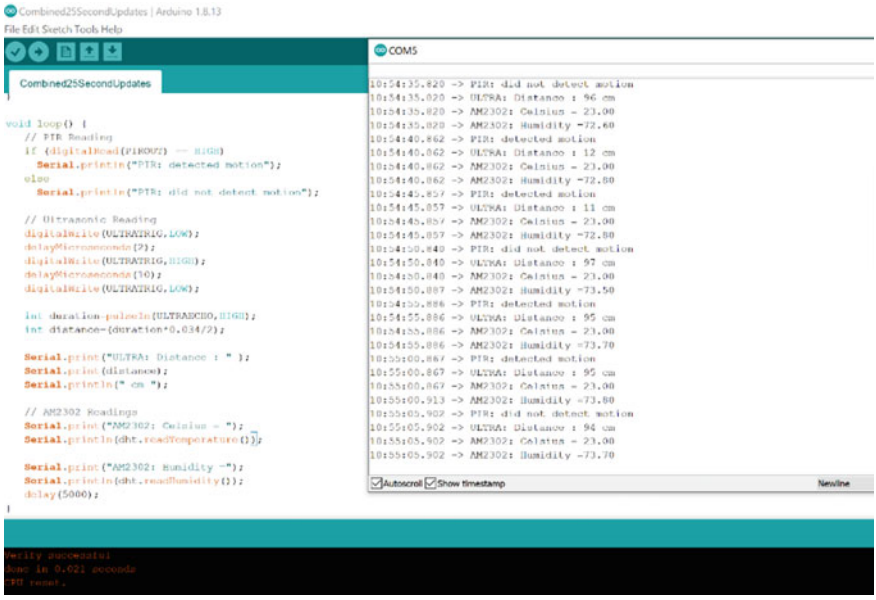


Fig. 8.12 The screenshot of combined sensor readings observed at serial monitor

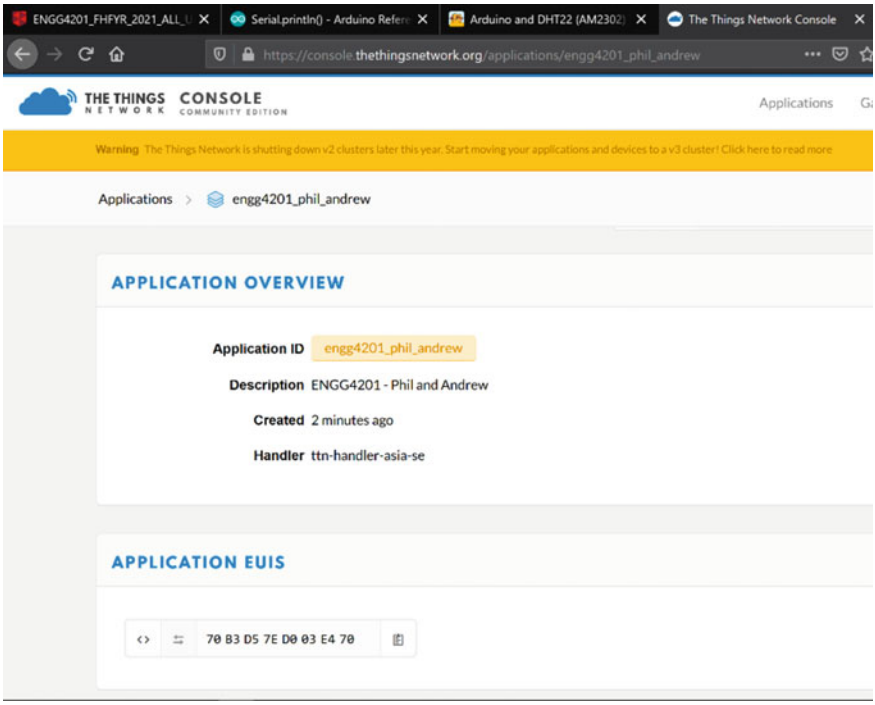


Fig. 8.13 The screenshot of completed application setup in TTN

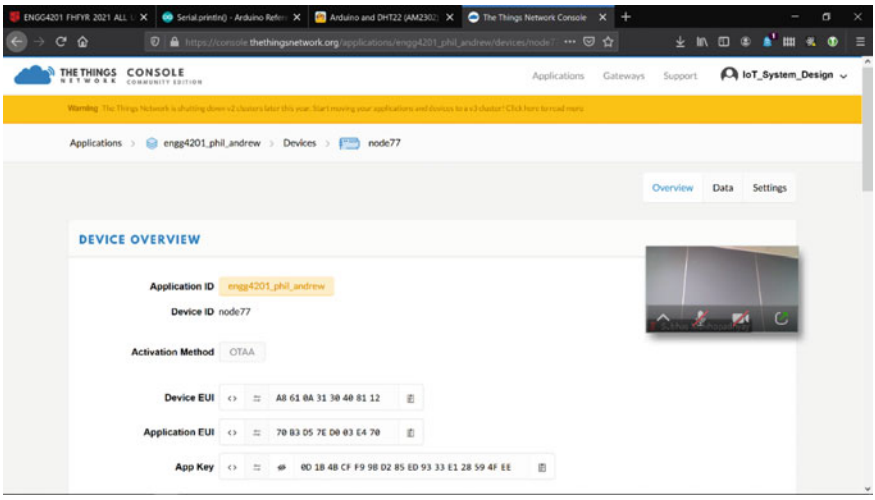


Fig. 8.14 The screenshot of using the TTN console to register the MKR1300 with the application

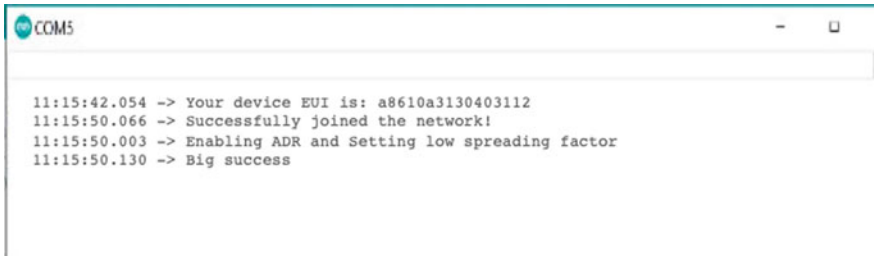


Fig. 8.15 The Sending a payload that says "Hello World"

Payload Assignment

Sending their sensor data to their Application within TTN required creating a payload that consisted of a compact binary encoding of their sensor data. The student group followed the suggestion to implement the CayenneLPP format.

Sending data using the CayenneLPP payload format was simple. The library provided convenience functions for encoding quantities typical for IoT sensors. The four sensor values were assigned to four channels as shown in the code fragment below and sent to their application:

```
#include <CayenneLPP.h> CayenneLPP lpp(51);
…
// Prepare Cayenne LPP
lpp.reset();
lpp.addTemperature(1, t);
lpp.addRelativeHumidity(2, h);
lpp.addDigitalOutput(3,pirState);
lpp.addAnalogOutput(4, distance);
// Send the data
modem.beginPacket();
modem.write(lpp.getBuffer(), lpp.getSize());
int err = modem.endPacket(true);
if (err > 0) {
  Serial.println("Message sent.");
} else {
    Serial.println("Error sending data.");
}
```

Sensor Data on TTN

Figure 8.16 shows the data sent from the MKR WAN 1300 device being received at the application. Since Cayenne LPP is a supported Payload Format on TTN, a custom decoder is not required to extract the sensor data from the payload to display it on the TTN console.

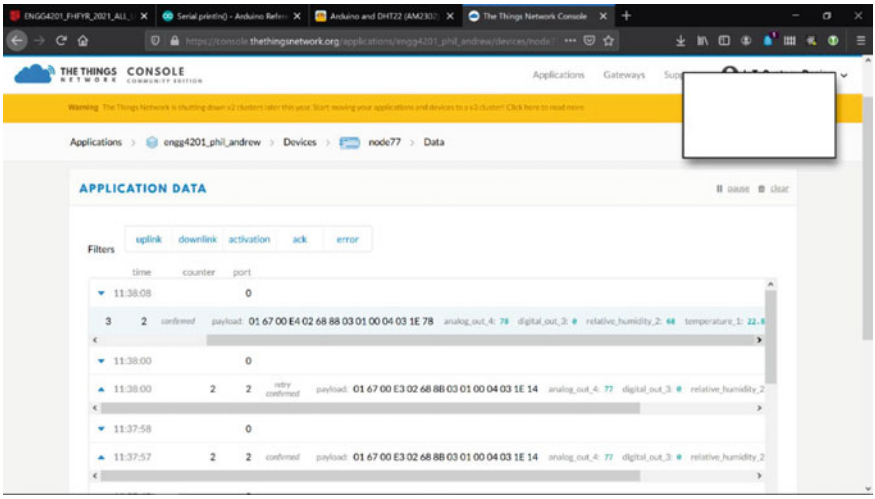


Fig. 8.16 The screenshot of the cayenne LPP payload format for application data on TTN

Understanding the CayenneLPP Payload Format

The documentation for the CayenneLPP payload format helped us to understand the binary payloads sent from the MKR1300, and allowed us to implement the "Custom" JavaScript Decoder() function. This converted the binary payload into a JSON object that was readable by their Application.

Figure 8.17. provides an example of the encoding used for Data Channel 4 (`analog_out_4` which corresponds to the `distance` variable within their sketch:

Custom Decoder

The purpose of a "Custom" Decoder in TTN is to convert payloads from the compact, binary encoding received from the device into a JSON format that can be processed by an application. The Decoder() function shown in Fig. 8.18 was

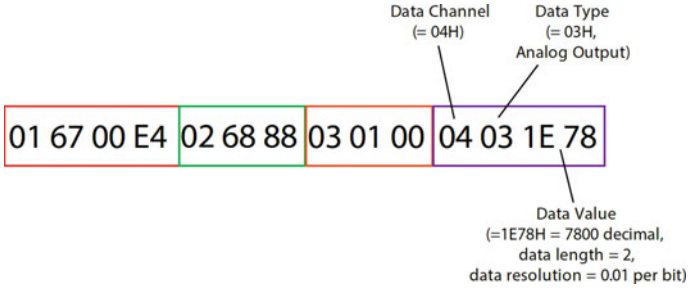


Fig. 8.17 The cayenne LPP payload format description

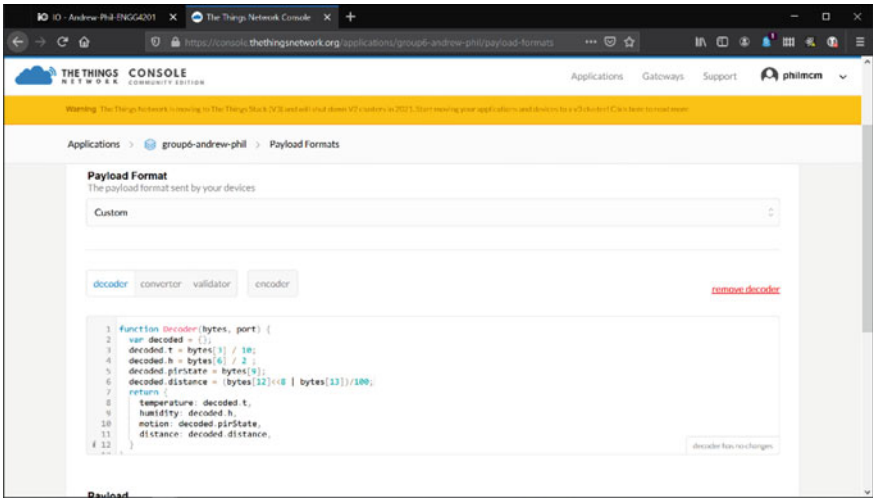


Fig. 8.18 The screenshot of the TTN console of function for decoding payloads

implemented to extract the temperature, humidity, PIR status and ultrasonic distance readings from the payload and to generate a more convenient JSON structure for further application integration with the IFTTT Maker application described in the next section:

Visualisation with Adafruit IO

There is no direct application integration available for Adafruit IO within the TTN console, so the first step required was to select an intermediate application integration that could bridge between the two services. The student group selected the IFTTT Maker integration to perform this function. After setting up an account on IFTTT, they extracted the key for the integration for IFTTT and created the integration on TTN console. They also added three of their four payload fields, to be sent to IFTTT as Maker Events, as shown below in Fig. 8.19.

The student group created an account on Adafruit IO and added the Adafruit service to IFTTT to link them. The student group created three feeds for their sensor variables within Adafruit IO, and then added three applets within IFTTT to forward sensor data received from TTN to them. The configuration is shown in Fig. 8.20.

After the student group configured a dashboard on the Adafruit console, they were able to view their sensor data. Figure 8.21 shows the Adafruit IO dashboard, which can be compared to the ThingSpeak dashboard they created in Week 1 shown in Fig. 8.22.

Challenges

IFTTT is designed to be simple, however the student group spent a lot of time looking for the key required to integrate TTN with IFTTT. They eventually found a

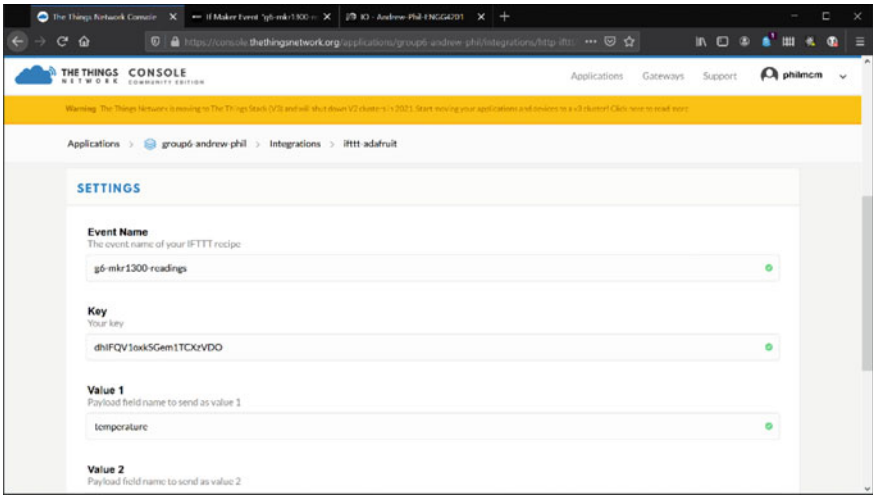


Fig. 8.19 The screenshot of the TTN Console, create an IFTTT integration

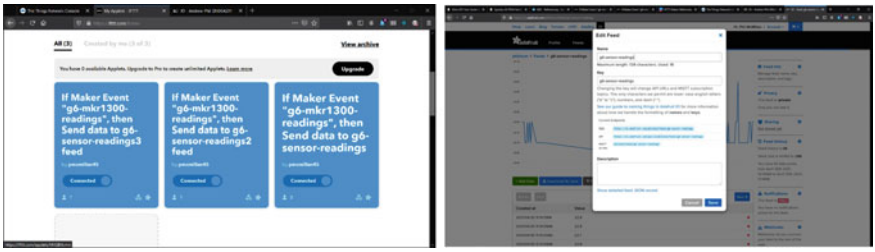


Fig. 8.20 The screenshot of IFTTT applets and Adafruit feeds configuration

tutorial which directed us to a page within the IFTTT Webhook Documentation, as shown in Fig. 8.23. This was counterintuitive, as they would have expected to find the key within the settings for the service:

They were also only able to send three of their four sensor variables via IFTTT. They spent time trying to find workarounds, but they could find no simple way to work around this limitation in the time available.

Possible Future Experiments

The TTN application integrations the student group have investigated to date, namely ThingSpeak and Adafruit IO, offer reasonably equivalent capabilities in terms of dashboards and visualisations of sensor data. One aspect they have not yet investigated are TTN integrations that may support analytics or hosting of machine learning models. Since their major project requires this functionality and they

Fig. 8.21 The screenshot of the Adafruit IO dashboard illustrating the sensor data

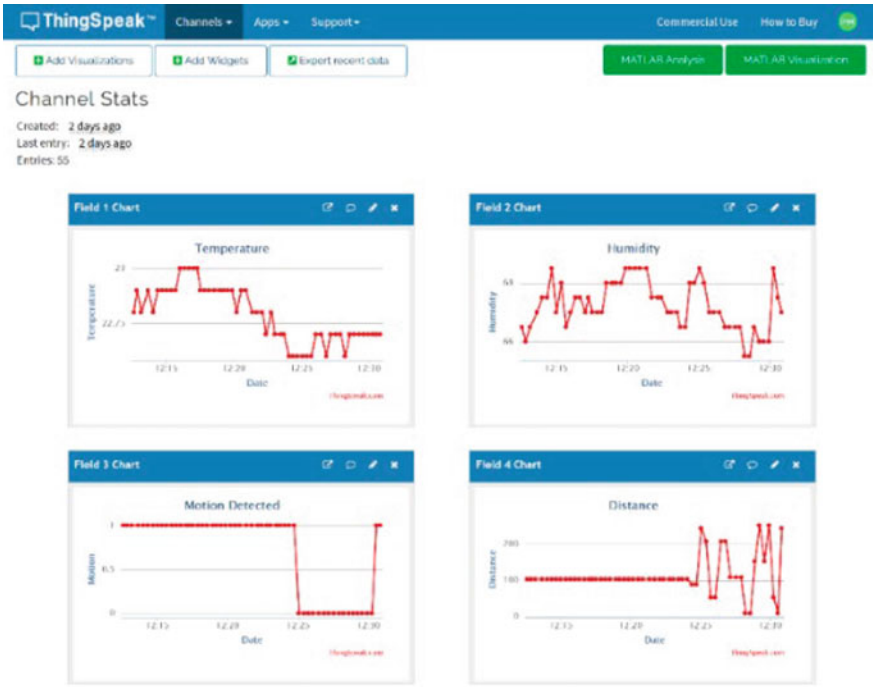


Fig. 8.22 The screenshot of ThingSpeak dashboard as a comparison to Fig. 8.21

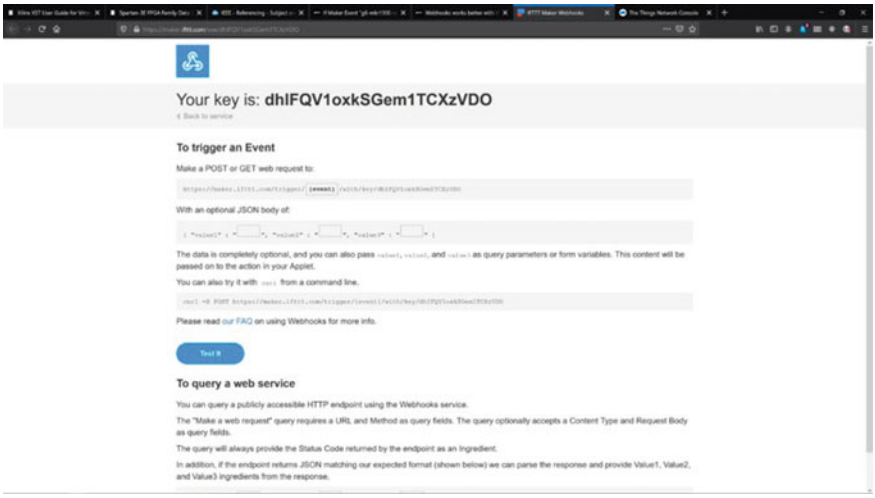


Fig. 8.23 The screenshot of the TTN to IFTTT integration's key

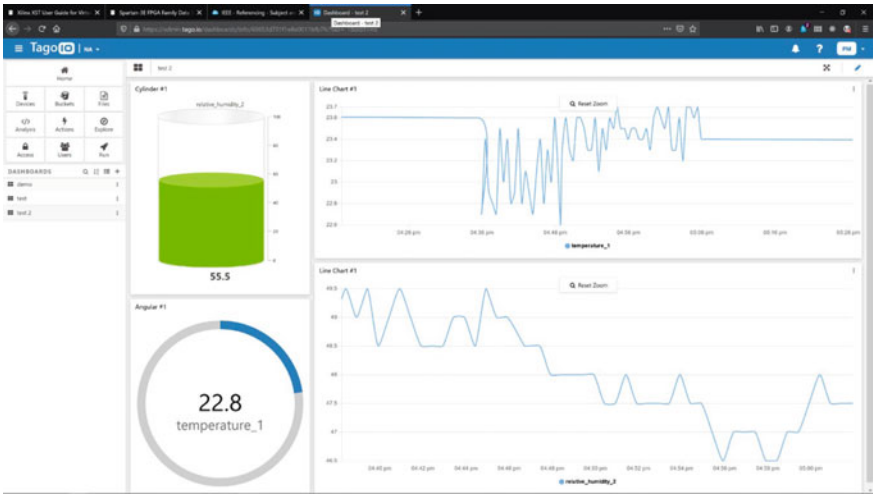


Fig. 8.24 The TTN-integrated Tago.IO application with Arduino MKR WAN 1310 device

intend using LoRaWAN as their communications mechanism, the student group will need to investigate these capabilities further.

One of the other interesting application integrations supported by TTN is Tago.IO as illustrated in Fig. 8.24. This application presents a similar dashboard capability to ThingSpeak and Adafruit IO, however it also supports sending data back to devices via TTN downlink messages, a capability they wish to explore for the project.

Conclusion

When a LoRaWAN network such as TTN is not available, it is still possible to use LoRa radios when two nodes need to be connected, as shown by the first task. While many of the benefits of using a LoRaWAN network (such as security and application integration) will not be directly supported, LoRa radio communication will still provide the benefits of long range and low power consumption and as such may be beneficial in certain remote IoT scenarios.

When LoRaWAN networks such as TTN are used, the IoT developer has multiple options for data visualisation using some pre-built integrations. Adafruit IO allowed us to set up a dashboard with three of their four sensor variables. However, they found the ThingSpeak dashboard easier to set up as it required no intermediate application such as IFTTT and supported all four of their sensor variables. One of the benefits of using LoRaWAN networks is that access to multiple cloud applications is available, so developers can choose the one that best fits the needs of the application.

To allow completion of their major project, next steps involve investigation of the available application integrations that could support a machine learning requirement. When combined with the options explored in this practical, they hope to construct a system that is capable of both visualisation and processing of their sensor data to support further analysis.

Suggested Reading

1. N.K. Suryadevara, S.C. Mukhopadhyay, Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sens. J. **12**(6), 1965–1972 (2012)
2. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, S.C. Mukhopadhyay, An IoT-enabled portable sensing system with MWCNTs/PDMS sensor for nitrate detection in water. Measurement, **178**, 109424 (2021)
3. F. Akhter, H.R. Siddiquei, M.E.E. Alahi, K. Jayasundera, S.C. Mukhopadhyay, An IoT-enabled portable water quality monitoring system with MWCNT/PDMS multifunctional sensor for agricultural applications. IEEE Internet Things J. (2021)
4. D. Thomas, R. Shankaran, M. Orgun, S. Mukhopadhyay, A secure barrier coverage scheduling framework for WSN-based IoT applications. IEEE Trans. Green Commun. Netw. (2021)
5. J. Henry, S. Tang, S. Mukhopadhyay, M.H. Yap, A randomised control trial for measuring student engagement through the internet of things and serious games. Internet of Things **13**, 100332 (2021)
6. S.C. Mukhopadhyay, S.K.S Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial Intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
7. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
8. S. Pal, M. Hitchens, T. Rabehaja, S. Mukhopadhyay, Security requirements for the internet of things: a systematic approach. MDPI Sens. 5897 (2020)

9. F. Akhter, S. Khadivizand, H.R. Siddiquei, M.E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
10. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using graphite/PDMS sensors. Sens. Actuators, A **286**, 43–50 (2019)
11. M.E.E. Alahi, N. Pereira-Ishak, S.C. Mukhopadhyay, L. Burkitt, An internet-of-things enabled smart sensing system for nitrate monitoring. IEEE Internet Things J. **5**(6), 4409–4417 (2018)
12. S.C. Mukhopadhyay, S.K.S. Tyagi, N.K. Suryadevara, V. Piuri, F. Scotti, S. Zeadally, Artificial intelligence-based sensors for next generation IoT applications: a review. IEEE Sens. J. (2021)
13. R. Yan, G. Tian, S.C. Mukhopadhyay, in *Special Issue on Intelligent Sensing and Information Mining*, http://www.mdpi.com/journal/sensors/special_issues/10th_ICST. A Special Issue of Sensors (2017). ISSN: 1424-8220
14. S. Mukhopadhyay, in *Special Issue "Theory, Design and Prototyping of Wearable Electronics and Computing*". A Special Issue of Computers (2017). ISSN: 2073-431X
15. R. Morello, S. Mukhopadhyay, E. Gaura, Z. Li, D. Slomovitz, S.R. Samantaray, Special issue on smart sensors for smart grids and smart cities. IEEE Sens. J. **17**(23) (2017). ISSN: 1530-437X
16. E. Sazonov, S.C. Mukhopadhyay (eds.), in *Wearable and Ambient Sensors for Healthcare and Wellness Applications*. A Special Issue of Sensors (2017). ISSN: 1424-8220
17. R. Shankaran, S.C. Mukhopadhyay (eds.), in *Security in IoT Enabled Sensors*. A Special Issue of Sensors (2018). ISSN: 1424-8220
18. S.C. Mukhopadhyay, E. Sazonov (eds.), in *Wearable and Ambient Sensors for Healthcare and Wellness Applications 2018*. A Special Issue of Sensors (2018). ISSN: 1424-8220
19. S.C. Mukhopadhyay, T. Islam (eds.), in *Innovative Technologies and Services for Smart Cities*. A Special Issue of Electronics (2018). ISSN: 2079-9292
20. S.C. Mukhopadhyay, N.K. Suryadevara, A. Nag (eds.), in *Wearabe Sensors and Systems in IoT Era*. A Special Issue of Sensors (2019). ISSN: 1424-8220
21. S.C. Mukhopadhyay (ed.), in *Home Automation for the Internet of Things*. A Special Issue of Sensors (2019). ISSN: 1424-8220
22. S.C. Mukhopadhyay, K.P. Jayasundera (eds.), in *Smart Sensor Networks*. A Special Issue of Electronics (2020). ISSN: 2079-9292
23. S.C. Mukhopadhyay, A. Sangaih (eds.), in *AI Enabled Communications on IoT Edge Computing*. A Special Issue of Electronics (2020). ISSN: 2079-9292
24. Z. Hussain, I. Ahmed, S.C. Mukhopadhyay (eds.), in *Signal Processing Techniques for Smart Sensor Communications*. A special issue of Sensors (2020). ISSN: 1424-8220
25. S.K.S. Tyagi, S.C. Mukhopadhyay, S. Zeadaly, W. Wei, A. Striegel, W. Wang, V. Piyri, O. Elloumi, N. Kumar, Artificial intelligence based sensors for next generation IoT applications. IEEE Sens. J. (2021). ISSN: 1530-437X
26. N.K. Suryadevara, S.C. Mukhopadhyay, in *Smart Homes: Design, Implementation and Issues*. Smart Sensors, Measurement and Instrumentation, vol. 14 (Springer, 2015). ISBN: 978-3-319-13556-4
27. R. Yan, X. Chen, S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 26. Structural Health Monitoring: An Advanced Signal Processing Perspective (Springer, 2017). ISBN: 978-3-319-56125-7
28. S.C. Mukhopadhyay, A. Mason, in *Smart Sensors, Measurement and Instrumentation*, vol. 4. Smart Sensors for Real-Time Water Quality Monitoring (Springer, 2013). ISBN: 978-3-642-37005-2-1
29. S.C. Mukhopadhyay, in *Lecture Notes in Electrical Engineering*, vol. 96. New Developments in Sensing Technology for Structural Health Monitoring (Springer, 2011). ISBN: 978-3-642-21098-3

30. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors, https://doi.org/10.3390/s150510350
31. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045

Chapter 9
Cloud Computing for IoT Systems

9.1 Introduction

Cloud computing refers to the distribution of accessing computing services, such as apps, storage, and computing capability, over the internet and on a pay-per-use basis. The technology provides a way to access apps as utilities over the Internet, and enables to design, modify, and personalise apps over the internet. The word 'Cloud' refers to Internet or a Network, where data is sent through wirelessly through the air, hence the word 'cloud'. The term 'Cloud Computing' was used for the first time in 2006 by the then Google CEO. Since then, the technology has seen continuous research and application in the IT industry.

Cloud computing is the process of altering, setting, and using software over the internet. It provides storage of data, infrastructure, and applications online. Figure 9.1 shows the basic architecture of cloud services in a network.

It is possible that a piece of software will not need to be installed on a local PC, and this is how cloud computing avoids difficulties related platform dependency. Due to this advantage, Cloud Computing makes corporate applications more accessible and collaborative.

This chapter discusses the cloud computing when applied to IoT systems. It describes the techniques and methods to apply cloud services in an IoT node project.

9.2 Need of Cloud Computing

New technological innovations are being added very often. Cloud computing and the Internet of Things are now closely related to web developments, with one creating a basis for the other's development.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_9

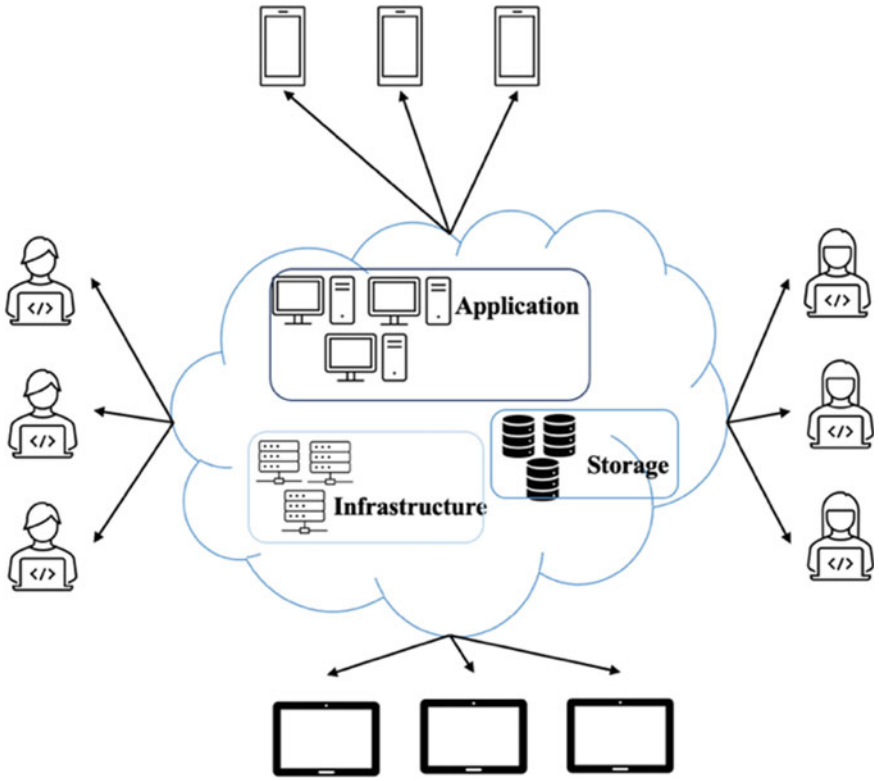


Fig. 9.1 The basic architecture of cloud services in a network

In the Internet of Things, cloud computing is used to store IoT data as a part of a coordinated effort. A cloud is a main server with computing resources that may be used at any time. Cloud computing is a convenient way to transport huge data volumes generated by the Internet of Things. The responsibilities on infrastructure management vary depending on whether an organization opts for on-premises, IaaS, PaaS, or SaaS implementation as seen in Fig. 9.2.

The cloud, which is based on the ideas of scalability and agility, is acclaimed as a game-changing technology all over the world. Cloud solutions have the potential to accelerate the adoption of IoT efforts on a wide scale. Here are a few important reasons why the cloud is critical to IoT success.

- Device to Device Communication
  Device to device connectivity and communication refer to the different 'things' in IoT connected and talking to each other. IoT systems need to link with one other in addition to talking with users. Cloud solutions allow IoT devices to communicate with one another in an efficient way. Many sophisticated APIs,

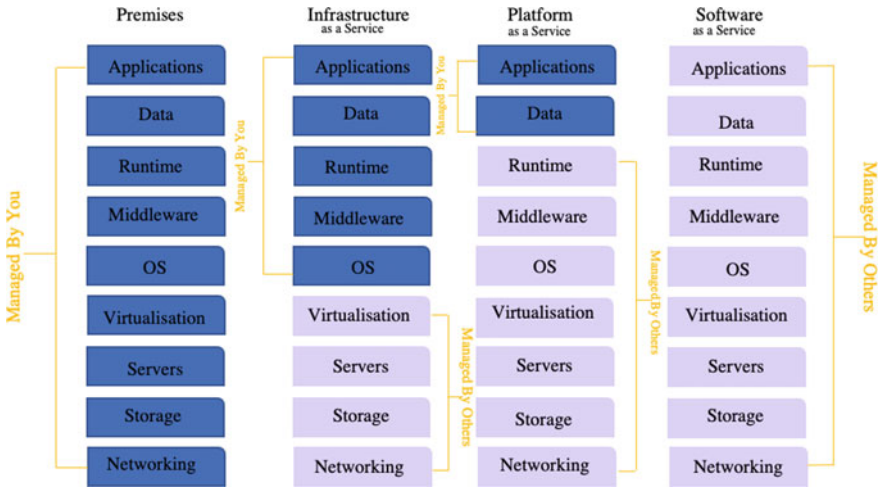


Fig. 9.2 An IoT embedded device's general architecture

such as Amazon CloudFront and Fastly, are enabled by them, allowing interface between connected smart devices, creating the path for the development of interconnected devices.

- Computing Processing Mobility
  Cloud technology is becoming more widespread as 5G and internet speeds im-prove, allowing organisations to access distant computing services remotely with mobility. The cloud has enabled enterprises to move beyond traditional IoT applications (for example, in household appliances) by lowering the requirement for on-premises infrastructure maintenance. This has paved the way for large-scale IoT deployment in previously uncharted territory.
- Security and Access Control
  Although the proliferation of IoT devices has enabled companies to automate processes, it has also raised serious security issues. The cloud, with its extensive set of restrictions, may be a feasible option in this case. Cloud solutions make it easier to deploy failsafe security measures by allowing businesses to use strong cryptographic techniques methods. It is feasible to monitor and safeguard the identification of people using IoT devices using enhanced cloud solutions.
- Big Data Scalability
  Big Data and Cloud Services can be combined to store massive volumes of data and enable extensible automation and real-time data analysis. The absence of physical structure required to bring Big Data, IoT, and the Cloud collectively lowers expenses and places emphasis on improving base analysis rather than development and maintenance.
- Business Reliability
  Cloud computing is well-known for its flexibility and dependability. Cloud services are designed to take advantage of a set of servers spread across several

places. The data is stored in numerous data centres throughout the country by their systems. Due to this reliability, IoT-based activities may manage to function as there are multiple servers. Furthermore, this decreases the possibility of data loss.

9.3 IaaS

The **I**nfrastructure **a**s **a** **s**ervice (IaaS) is a type of cloud computing that delivers virtualized system resources via the internet. A cloud provider hosts the infrastructure components that would generally be found in an on-premises data centre under an IaaS service model. This comprises the virtualization or hypervisor layer, as well as servers, storage, and networking gear. These services seem to be more policy-driven, designed to allow IaaS users to automate and synchronize key infrastructure activities to a higher level. For example, to ensure application availability and performance, a user might set policies to drive load balancing. The advantage of IaaS is illustrated in Fig. 9.3.

These IaaS clients connect to resources and services through a wide area network (WAN), such as the internet, and then use the cloud provider's services to complete the application stack. The user may utilize the IaaS platform to build virtual

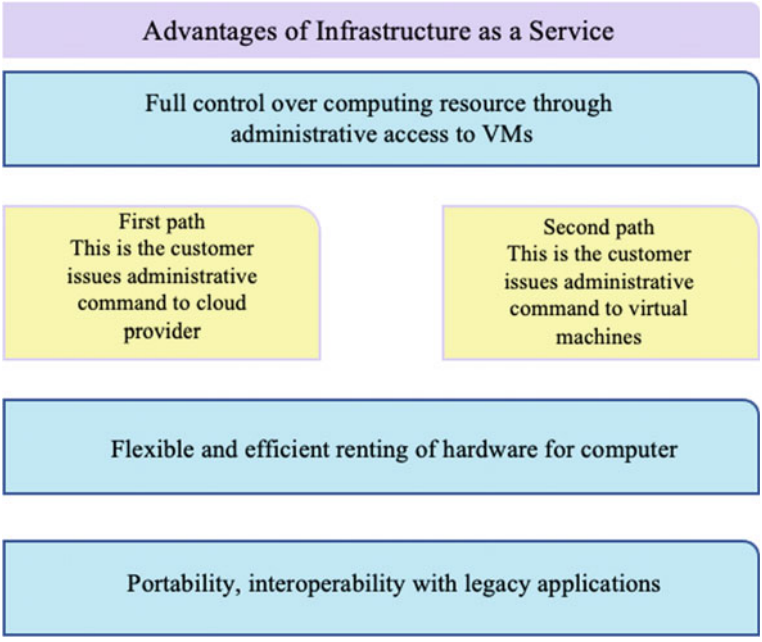


Fig. 9.3 Advantages of the IaaS

machines (VMs), install operating systems in each VM, deploy middleware such as databases, and create storage buckets. Prominent examples of IaaS are AWS EC2, Google Compute Engine (GCE).

IaaS billing may be an issue for certain organizations, despite its flexible, pay-as-you-go approach. Cloud invoicing is highly detailed, and it is split down to represent specific service consumption. Another issue that IaaS consumers face is lack of insight. Because IaaS service owns the system, the configuration and performance of that infrastructure are rarely transparent to IaaS customers.

9.4 PaaS

PaaS is short form of **P**latform **a**s **a** **S**ervice. PaaS cloud services supply certain software with network elements when used principally for programs. PaaS enables programmers with a platform that they can utilise and build on to design specific projects. The company or a third-party supplier may handle all servers, storage, and networking, while developers are able to retain application administration. PaaS enables programmers with a platform that they can utilise and build on to design configurable projects. The company or an enterprise supplier may handle all storage, networking, and servers, while programmers are able to retain software control.

PaaS distribution concept is related to SaaS as described in the next section except that PaaS offers a platform for building software rather than distributing the programme on the Web. It is provided through the Internet, enabling users to focus on programme construction with less worrying concerning operating systems (OS), upgrades to program, storage or infrastructure.

In the PaaS concepts, a computer framework, a programming structure, a storage server and a webserver, are generally provided by cloud services. PaaS enables companies to develop apps with specific development tools that are integrated with PaaS. Sometimes referred to as ‘middleware’, these programmes are expandable and highly available as they take over certain cloud features.

PaaS has various advantages. Some of the important ones are listed below:

- The service is simplified, development is cost efficient and application deployment is simple to implement.
- The platform is scalable due to its configurability options.
- It provides multiple accessibility options to users.
- The Dev Ops customisation options are plenty and need for software maintenance is redundant.
- Programming required for developers is reduced drastically due to multiple framework options.
- Company policy optimization and automation is provided.
- Hybrid models combine public and private cloud services. PaaS allows simple hybrid model migration to users.

While PaaS has many useful applications, some of the limitations and drawbacks of this service are shown in Fig. 9.4.

Lastly, some of the applications that provide PaaS systems are given: SAP Cloud, Microsoft Azure, Heroku, AWS Lambda, Google App Engine, Dokku, IBM Cloud Foundry and Oracle Cloud Platform.

9.5 SaaS

A **S**oftware **a**s **a** **s**ervice (SaaS) is a cloud-based software delivery model in which a cloud provider creates and sustains software for cloud application, that offers automated software upgrades, and makes software available to consumers on a pay-as-you-go basis through the internet. In comparison to on-premises applications, the SaaS model was created to provide a fundamental set of business benefits. The advantages are illustrated in Fig. 9.5.

The prominent SaaS examples: Google Apps, Dropbox, DocuSign, Slack, HubSpot.

The highlights of SaaS in comparison to IaaS and PaaS are:

- SaaS allows providers to host, administer, and provide consumers with the full infrastructure as well as applications. Users using SaaS applications do not need to install anything; instead, they just log in and utilize the provider's application, which is hosted on the provider's infrastructure. Users have some control over how the program functions and which users are allowed to use it, however the provider of SaaS is responsible for everything else.

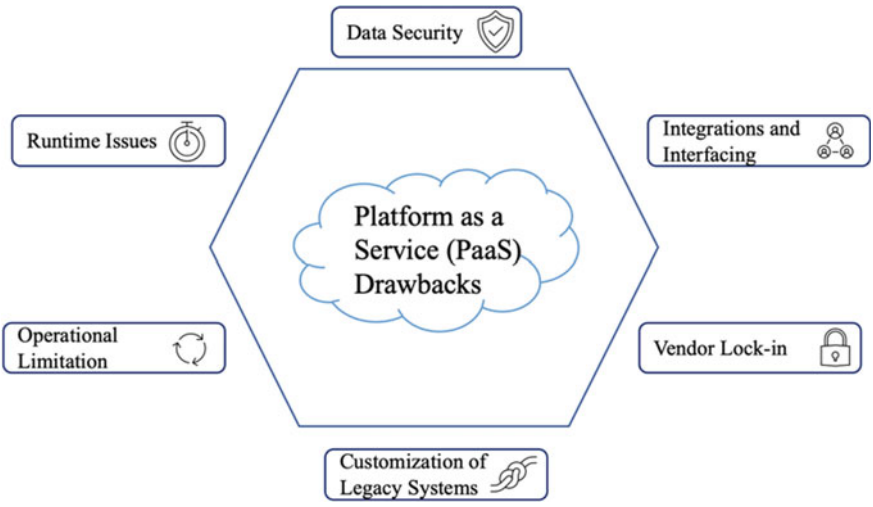


Fig. 9.4 Limitations of PaaS

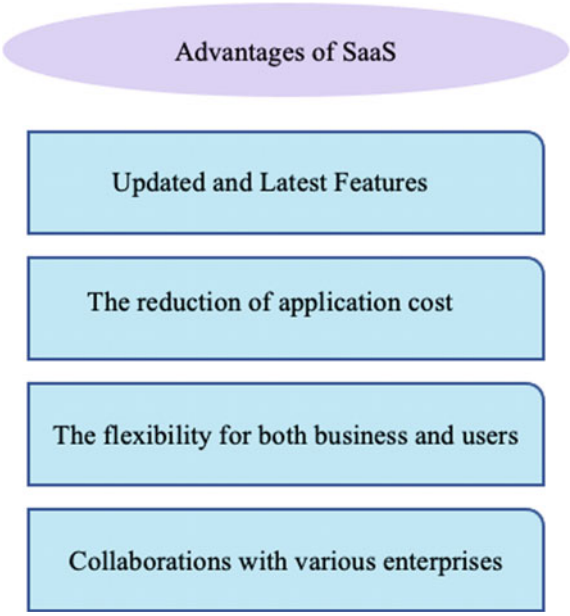


Fig. 9.5 Advantages of the SaaS

- PaaS expands on the IaaS paradigm since providers host, maintain, and supply operating systems, middleware, and other runtimes for cloud customers in addition to the core infrastructure components. While PaaS makes workload deployment easier, it limits a company's ability to design the environment it wants.
- IaaS is only one of numerous cloud computing models that may be combined with PaaS and SaaS to provide a more comprehensive solution.

While SaaS has many useful applications, some of the limitations and drawbacks of this service are listed below:

- Maintenance Issues: Since the service provider owns and administers the SaaS system, business customers then rely on the seller to keep the service reliable and efficient. Despite appropriate Service Level Agreement (SLA) protection in place, scheduled and unexpected maintenance, cyber threats, or connection problems might affect the operation of the SaaS app.
- Feature Constraints: Since SaaS programmes are frequently available as default, the selection of features might risk safety, price, efficiency and other business regulations.
- Vendor Lock-in: In addition, vendor lock-in, pricing or security issues may imply that suppliers or services cannot be switched on in the future to meet new design criteria.

9.6 Example of Cloud Service Used: ThingSpeak

Communication is one of the key ThingSpeak is a popular IoT based cloud and data analytics platform. The service provides data visualisation, aggregation, analysis and storage options as cloud services to connected devices. The service is usually used to build prototypes or proof of concept systems. Figure 9.6 shows how ThingSpeak and MATLAB work in an IoT network cloud.

The service works in real time enabling variety of IoT applications where sensors continuously send data to the cloud. ThingSpeak's interfacing with MATLAB programs allows for real-time data processing and analysis.

The process of this system works as follows:

On the right, the smart devices are present at the edge of a network system. These nodes monitor and send the data and consist of things like smart door lock, smart bicycle, medical sensors, fitness trackers, smart security system among many.

In the centre, we have the cloud, which aggregates and analyses information from a multitude of sources in real time, typically using an IoT analytics platform developed in the ThingSpeak system.

The left side of the figure shows how the IoT system develops the methodology. Here, a data analyst attempts by making historical analyses on the data to obtain a

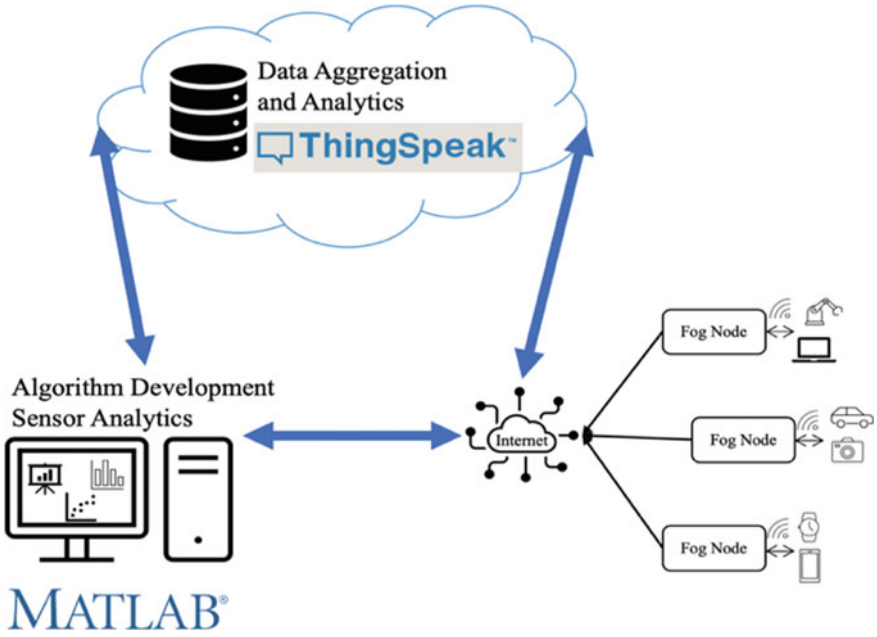


Fig. 9.6 The basic architecture of cloud services in a network

deeper understanding of the gathered data. This helps achieve strategists and scientists to create prototyped algorithms which can be run in the cloud or in the intelligent node itself from the IoT platform.

9.7 Example of Cloud Service Used: AdaFruit IO

The Adafruit IO is a useful cloud provider with more attention to IoT cloud installations. Adafruit IO enables various devices such as Raspberry Pi, ESP8266 and Arduino models. Figure 9.7 shows a screenshot of an Adafruit IO cloud generated for temperature, humidity, and PIR motion sensor based IoT project.

For the following reasons, IoT developers favour Adafruit IO above other IoT cloud suppliers:

- Robust API—The service offers several software programs with libraries, which also support the built-in user interface and application programming interface (API).

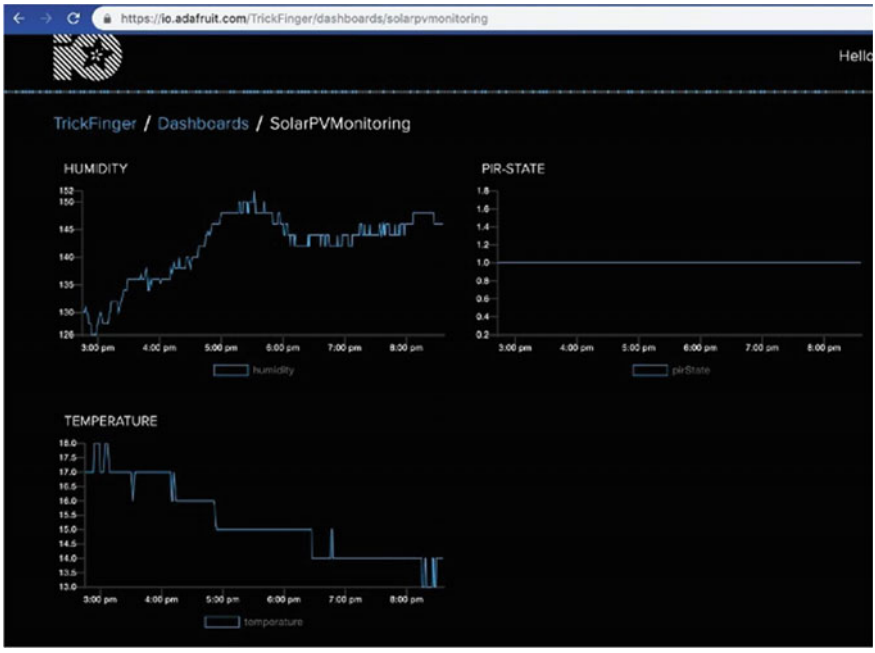


Fig. 9.7 The basic architecture of cloud services in a network

- Manuals and Community—Many articles with incredible support for the community enable continual product development. The open-source documentation available provides confidence while doing projects with debug-ging resources available on forums.
- Dashboard Interface—Data comprehension through charts and infographics al-lows to make better forecasting models.
- Data protection—Improved encryption technique secures data within the cloud platform.

Suggested Reading

1. N.K. Suryadevara, S.C. Mukhopadhyay, in *Smart Homes: Design, Implementation and Issues*. Smart Sensors, Meas and Instrumentation, vol. 14 (Springer, 2015). ISBN: 978-3-319-13556-4
2. R. Yan, X. Chen, S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 26. Structural Health Monitoring: An Advanced Signal Processing Perspective (Springer, 2017). ISBN: 978-3-319-56125-7
3. S.C. Mukhopadhyay, A. Mason, in *Smart Sensors, Measurement and Instrumentation*, vol. 4. Smart Sensors for Real-Time Water Quality Monitoring (Springer, 2013). ISBN: 978-3-642-37005-2-1
4. S.C. Mukhopadhyay, in *Lecture Notes in Electrical Engineering*, vol. 96. New Developments in Sensing Technology for Structural Health Monitoring (Springer, 2011). ISBN: 978-3-642-21098-3
5. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350
6. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045
7. S.C. Mukhopadhyay, O. Postolache, in *Smart Sensors, Measurement and Instrumentation*, vol 2. Pervasive and Mobile Sensing and Computing for Healthcare: Technological and Social Issues (Springer, 2012). ISBN: 978-3-642-32537-3
8. S. Mukhopadhyay, in *Special Issue "Theory, Design and Prototyping of Wearable Electronics and Computing"*. A Special Issue of Computers (2017). ISSN: 2073-431X
9. S.C. Mukhopadhyay, A. Sangaih (eds.), *AI Enabled Communications on IoT Edge Computing, A sepcial issue of Electronics* (2020). ISSN: 2079-9292
10. W. Wu, S. Pirbhulal, A.K. Sangaiah, S.C. Mukhopadhyay, G. Li, Optimization of signal quality over comfortability of textile electrodes for ECG monitoring in fog computing based medical applications. Future Gener. Comput. Syst. **18**, 515–526 (2018)
11. S.C. Mukhopadhyay, N.K. Suryadevara, R.K. Rayudu, in *Are Technologically Assisted Homes Safer for the Elderly, Smart Sensors, Measurement and Instrumentation*, vol. 2. eds. by S.C. Mukhopadhyay, O. Postolache, Pervasive and Mobile Sensing and Computing for Healthcare: Technological and Social Issues (Springer, 2012), pp. 51–68. ISBN: 978-3-642-32537-3

12. O.A. Postolache, E. Sazonov, S.C. Mukhopadhyay (eds.), *Sensors in the Age of the Internet of Things: Technologies and Applications, IET Book*. https://digital-library.theiet.org/content/books/ce/pbce122e, ISBN: 9781785616341, e-ISBN: 9781785616358 (2019)
13. N.K. Suryadevara, S.C. Mukhopadhyay (eds.), *Assistive Technology for the Elderly* (Elsevier, 2020). ISBN: 978-0-12-818546-9
14. A.K. Sangaiah, S.C. Mukhopadhyay (ed.), *Intelligent IOT Systems in Personalized Health Care* (Elsevier, 2021). ISBN: 978-0-12-821187-8

Chapter 10
Machine Learning in IoT System

10.1 Introduction

The Internet of Things (IoT) was established on the production of billions of physical things and mobile devices that are connected to the Internet. Machine learning is motivated by data and uses it to produce ideas. Machine learning identifies patterns in previous behaviour and creates models to predict subsequent behaviour and occurrences. Algorithms are trained to produce classifications or predictions using statistical approaches, revealing significant insights in data mining initiatives. Following that, these insights drive decision-making within applications and companies, with the goal of influencing important growth indicators. Figure 10.1 shows the broader process of the implementation of the machine learning algorithm.

By analysing vast amounts of data with advanced algorithms, machine learning may assist decode the hidden patterns in IoT data. In essential operations, machine learning inference can augment or replace manual processes with automated systems that use statistically determined actions. In this chapter, the architecture of machine learning is briefed on followed by the benefits. This is finally concluded with the implementation of machine learning in an IoT systems is illustrated.

10.2 Machine Learning Architecture

Machine learning is a subfield of artificial intelligence (AI) and computer science that emphasizes on using data and algorithms to adapt to the way people learn, with the goal of steadily improving precision. Machine learning for IoT is being used by businesses to conduct accurate predictions on a wide range of used cases, allowing them to acquire fresh perspectives and enhanced automation capabilities. Users may use machine learning for IoT to: Consume and convert data into a standardized

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_10

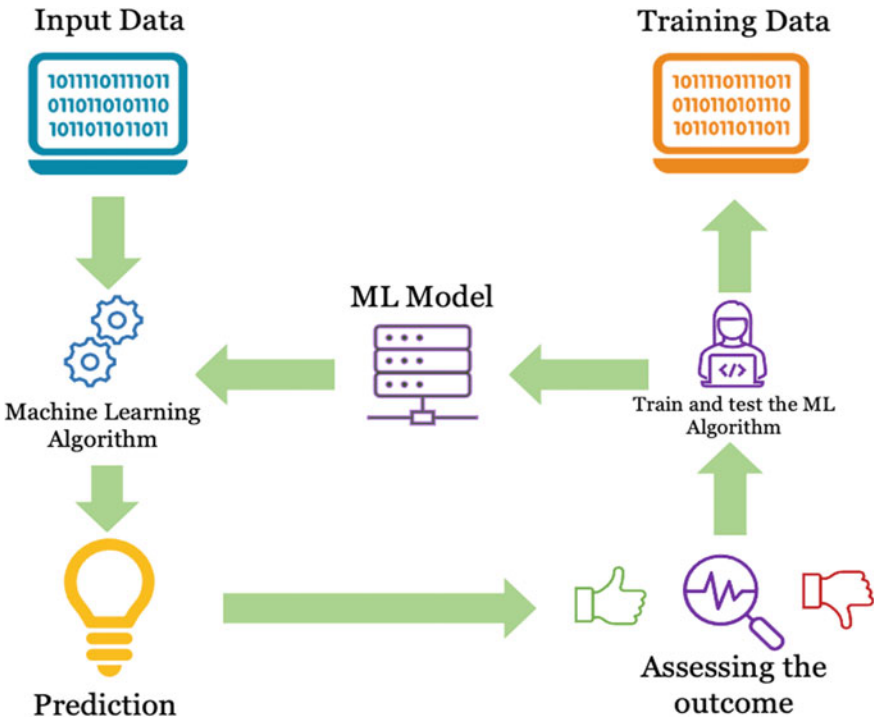


Fig. 10.1 The process of implementation of machine learning algorithm

format, create a machine learning model and deploy it to the cloud, the edge, and the device. Figure 10.2 shows the architecture of Machine Learning in an IoT system that has been illustrated by Cumulocity in 2020. The proposed Cumulocity IoT system is a minimal-code, self-service IoT platform includes machine learning as a vital element. The platform includes all the tools needed such as device communication and administration, application enablement and integration, streaming analytics, machine learning, and model deployment, these are highlighted in the architecture below. Such architectures can be developed in IoT systems built by the users.

Every day, many gadgets connect to the internet and transmit data of sensors. Without analysis, this information is futile. The data that projects create, on the other hand, is efficiently gathered, processed, and stored when an IoT system with machine learning is in place. Tgus, it optimizes operations at all levels, enhance decision-making, and realize a variety of advantages that will be discussed in the next section.

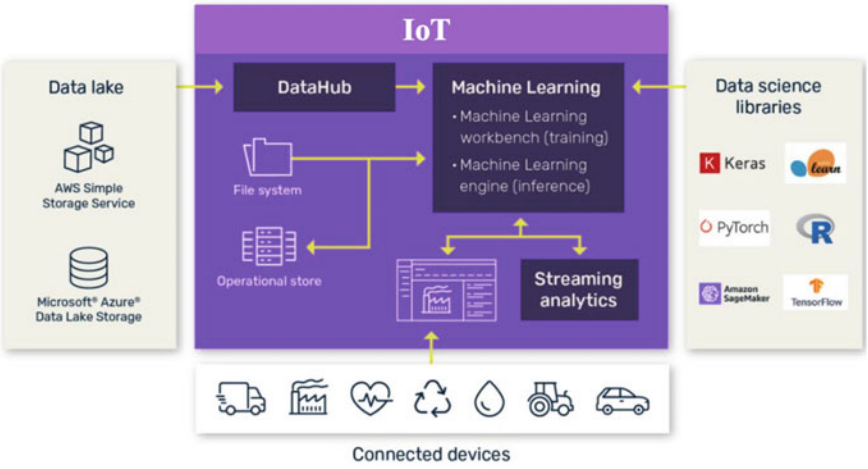


Fig. 10.2 The basic architecture overview of Machine Learning in IoT by Cumulocity in 2020

10.3 Benefits and Challenges of Machine Learning

The machine learning algorithm's learning mechanism is divided into three primary components by UC. Berkley: (a) Prediction or Classification Process: Machine learning algorithms are used to produce predictions or classifications in general. The algorithm will provide an estimate about a pattern in the data based on some input data, which can be labelled or unlabelled. (b) Error Function: An error function is used to assess the model's prediction. If there are known instances, an error function may be used to compare the prediction accuracy. (c) Model Optimisation Process: Weights are changed to decrease the gap between the known example and the model estimate if the model can fit better to the data points in the training set. This assessment and optimization procedure will be repeated by the algorithm, which will update weights on its own until a certain level of accuracy is realized.

The top 3 benefits of machine learning in IoT systems are:

- Efficiency Improvement Methods.
  About 26% of businesses today have IoT enabled in the system. ML enables IoT systems to accurately detect and notify inefficiencies in system by analysing data. This allows companies to develop better efficient models for the processes.
- Robotic Process Automation (RPA)
  RPA has lately been adopted by major industries to allow automation of routine tasks. IoT devices generate crucial intricate data, machine learning models would help improve such RPA systems.
- Smart Supply Chain

With the advent of online ordering, supply chains have become an important asset to companies. IoT devices can monitor and track real time movement, quality, and location of products. These real-time data from IoT sensors are taken from and implemented by machine training algorithms. Such system could pinpoint the error locations and allow quick action as well.

Machine learning has made our lives simpler as it evolves. However, incorporating machine learning into IoT systems and projects has generated a variety of issues about AI technology. Here are a few examples:

- Discrimination and bias: Bias and prejudice in a variety of intelligent systems have generated several concerns about the use of artificial intelligence. How can we prevent the prejudice and discriminative results when the training data itself is prone to bias?
- Privacy protection: Privacy is usually considered in terms of data privacy, data protection, and data security, and these concerns have prompted researchers to make greater progress in this area in recent years.
- Accountability: There is no meaningful enforcement mechanism to guarantee that ethical AI is implemented because there is no substantial policy to govern AI techniques. The present motives for organisations to follow these rules are the financial consequences of an immoral AI system.

10.4 Data Analytics

Data analytics in general is the process of analysing raw data. Fundamentally, IoT and data remain to stay interconnected. Recently defined, the analysis of large amounts of data created by linked devices is IoT data analytics or as 'IoT Analytics'. The emerging field of IoT Analytics is complex. The compilation, storage, and extraction of commercial value from IoT devices require large amounts of heterogeneous data. This needs a range of resources, including data lakes and streaming processing frames. Organizations may profit from this through optimising operations, automating controlling processes, attracting more consumers, and empowerment of workforce. In ecommerce, medical tech, telecoms, industrial production and smart cities, the integration of IoT with data analytics has already been successful. Its actual worth has, although, yet to be completely recognized for organisations.

Data generated by IoT devices is a part of big data and develops in volume, variety, and velocity steadily (the 3Vs model). It comprises of heterogeneous resources to be processed and integrated to generate current, complete, and correct data for corporate analysis and interpretation. Interoperability with other IoT devices and systems was not the purpose of designing many IoT systems. Industrial IoT (IIoT) is thus soon becoming analytics ready to become future ready.

10.4.1 Types of IoT Analytics

The following points discuss the types of IoT Analytics:

1. Illustrative Analytics in IoT
   Such analytics system emphasises on the data gathered from IoT devices using real time tracking of devices, installations, equipment, and assets. It also evaluates whether things proceed as designed and reports abnormalities. Illustrative analytics are usually used as dashboards, which provide past and present (real time) sensor data, Objectives and Key Results (OKRs), trends and warnings. It helps to answer the following questions:

- Do any of the data show irregularities that require scrutiny?
- What is the expected performance and output of this device?
- Can we reduce the power consumption of this machine?

2. Reasoning Analytics in IoT

Diagnostic proficiencies generally include dashboard enhancements that help consumers to dig in, correlate and determine variations when required. Instead of data scientists, several companies hire domain specialists with information concerning a certain process, mechanism, technology, or product. It helps to answer the following questions:

- Why is the device not working at optimal performance?
- Why are the consumers complaining about a specific part of this product?
- Why isn't the machinery producing the required goods?

3. Predictive Analytics in IoT

This system has been developed and designed using machine learning (ML) models that use training datasets from statistical data and deployed on cloud system for use by front end applications. It makes predictions and applies probability theory to the system to pre-emptively rectify, limit the risk, or identify chances before an unwanted event happens. It helps to answer the following questions:

- What is the product lifetime of this device?
- When are the goods expected to arrive to the facility?
- What will be the requirement for this add-on in the future?

4. Prescriptive Analytics in IoT

This type of system recommends steps that are focused on a forecast, a diagnosis or understand the reason behind a forecast or diagnosis visibility. Suggestions are generally over how changes may be optimised or fixed. It helps to answer the following questions:

- This device is giving errors in reading. How can it be improved?
- The data shown is lagging on the server. What action needs to be performed?

- The hardware design of a sensor node is faulty. How can this be rectified?

Several manufacturers offer purpose-built IoT data analysis motors. One can utilise one of these technologies or directly analyse IoT data using normal analytical techniques, such as any large-scale data. The most popular and useful analytics systems are provided by Amazon Web Services (AWS) IoT Analytics and Microsoft Azure IoT analytics.

10.5 Edge Computing/Edge Analytics

Edge computing is the method of analysing and processing data that is gathered by sensing IoT nodes that are away from the main gateway or cloud servers, and closer to the theoretical edge of the network. The 'Edge' in Edge Computing refers to these IoT nodes that use computing at the edge of the network layer. Some of the benefits include improved latency, reduction in bandwidth requirement, and better response times. The goal of this architecture development is enabling IoT devices to process higher computational capability requiring applications. Unlike IoT, which is a technology, edge computing refers to an architecture or infrastructure. The infrastructure of an Edge Computing system is shown in Fig. 10.3.

An edge device, as seen in the image above, can broadly be explained as a network element that manages the connection of the Local Area Network (LAN) to a peripheral Wide Area Network (WAN), where data the collected is processed. Some of the applications of edge computing devices are cloud gaming, smart cities, Industry 4.0 (smart industry), robotics and video processing to name a few.

10.6 Example of Machine Learning: Forecasting

This project will illustrate a forecasting model used in an IoT system. To predict weather and detect floods, a machine learning regression model will be used. For this data need to be collected to google sheets. MATLAB is used in ThingSpeak and IFTTT (If This Then That) with webhook is used to record the data on to the google sheets. This data can be used to train and implement machine learning. Figure 10.4 shows the google sheets recording data for implementation of the ML model.

For predicting the weather on google sheets the users are using the forecast function in excel to collect the data and predict various parameters. The time of prediction will have the time for which the user wants the prediction. The forecast function is shown below in Fig. 10.5 along with the prediction.

```
=ARRAYFORMULA(FORECAST(G2,C2:C574,F2:F574))
```

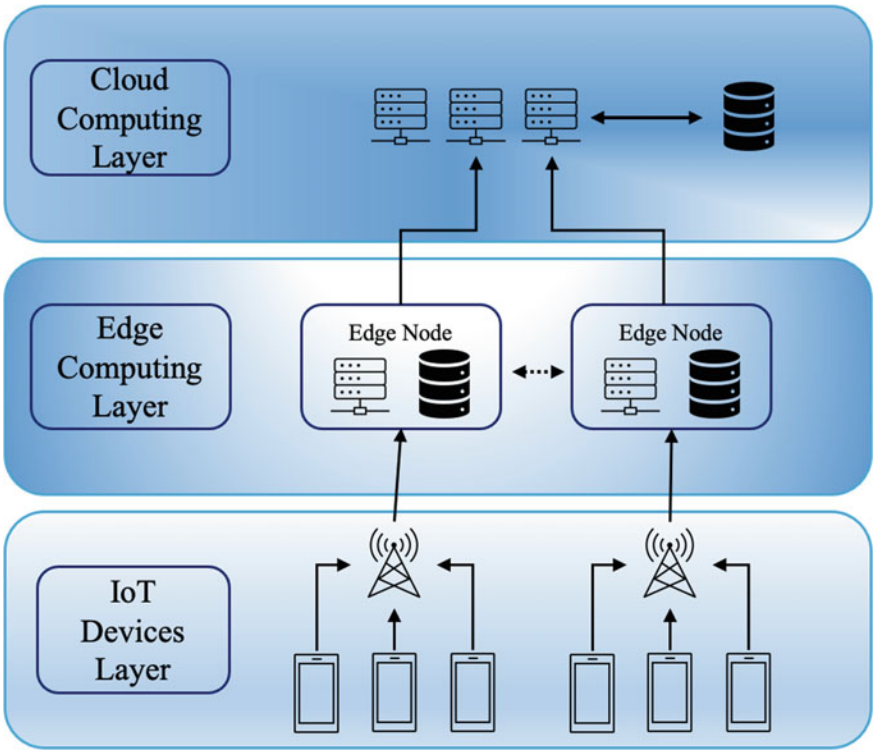


Fig. 10.3 The basic architecture of Edge Computing

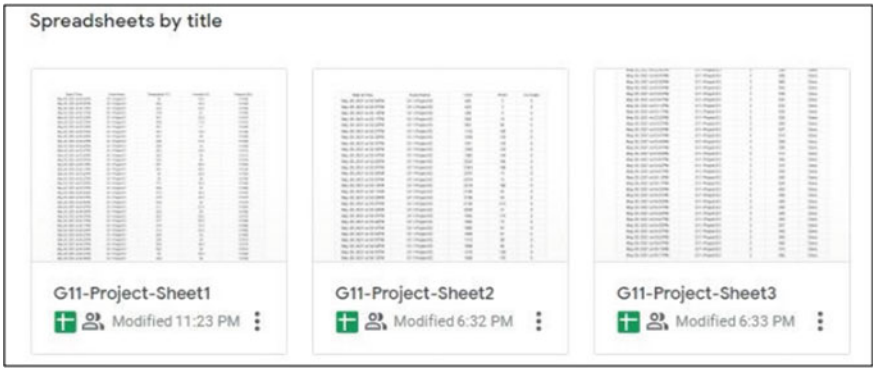


Fig. 10.4 The screenshot of Google Sheets linked with IFTTT API

G	H
Time of Prediction	Temperature Prediction
12:00 AM	25.50912081
2:00 PM	24.36816375
10:00:00 AM	24.64880195

Fig. 10.5 The screenshot of Temperature Prediction on Excel

```
function getAddressFromCoordinates(latitude, longitude) {
    const url = `https://revgeocode.search.hereapi.com/v1/revgeocode?apiKey=${process.env.
    REACT_APP_GEO_KEY}&at=${latitude}%2C${longitude}`

    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", url, false );
    xmlHttp.send( null );
    try{
        return JSON.parse(xmlHttp.responseText).items[0].title;
    }catch(e){
        return null;
    }
}
```

Fig. 10.6 The snippet of JS code to find address from Lat and Lon

10.7 Example of Machine Learning: Location. Prediction

The Machine Processing and Data Analysis for this project was demonstrated by using a training a deep neural network to be able to predict the user's future location from historical data. Using third party resources to provide reverse geocoding on the user's position as seen in Fig. 10.6.

10.7.1 Reverse Geocoding

To get the address from a latitude and longitude coordinate (reverse geocoding the users use a third-party API called here.com. For this they registered a project and created a key in the here.com web interface. From their UI they then perform a xml call to this API using their key, and the last known coordinates. After this the users then extract the address from the returned JSON object.

10.7.2 Location Prediction

To predict a future location, the users use MATLAB's Machine Learning and Deep Neural Network Toolboxes. They initially will create a `PostgreSQL` interface given in the `databaseExplorer` tool. From here they then run a MATLAB script which trains a new network with all location data within the last 3 months from the database. Or it re-trains an existing network with any new coordinates data.

They decided on having two features in their network as seen in Fig. 10.7. The seconds since midnight of the input as well as the day of the week (Where Sunday = 1 and Saturday = 8). Both inputs have been normalised to be between –1 and 1.

The users also have to response of their network latitude and longitude, these are also normalised to be between –1 and 1.

The users normalise the data to increase training speed and a faster convergence as it allows all inputs and outputs to contribute the same weighting to the mean square error.

The users also have to response of their network latitude and longitude as seen in Fig. 10.8, these are also normalised to be between –1 and 1. They normalise the data to increase training speed and a faster convergence as it allows all inputs and outputs to contribute the same weighting to the mean square error.

Unfortunately, this is unable to predict locations currently due to a shortage of data. Because of this reason they have also been unable to optimise the settings of their network.

In the future, once they get more data, a possible improvement could be to make the neural network recursive, that is for each feed through they also add features for the location that they were at 1 h ago (2 features would be. required, 1 for latitude, and 1 for longitude). This would allow the machine to be adaptive to a new situation rather.

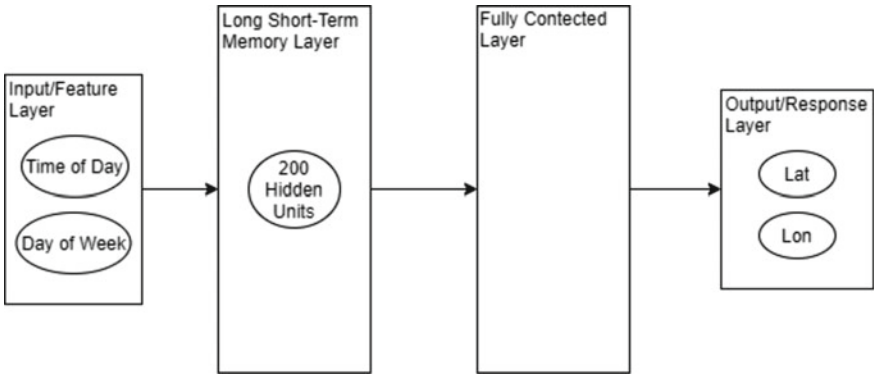


Fig. 10.7 The basic architecture of their DNN

```
function [lat, lon] = predictLocation(h)
lat = 0;
lon = 0;
if exist('mlData.mat') == 2
   load("mlData.mat")
   if exist('net', 'var')
       currentTime = datetime(now, 'ConvertFrom', 'datenum') + hours(h);
        x = [];
        x(1) = floor(second(currentTime))/86400 - 0.5;
        x(2) = (weekday(currentTime) - 1)/7 - 0.5;
        x = x.';
        YPred = predict(net, x);
        lat = YPred(1) * 90;
        lon = YPred(2) * 180;
   end
end
end
```

Fig. 10.8 The snippet of Predicting a location from the neural network code

then just remembering the average location at each time step. For this to be effective the users would need months, if not years, of data. For example, Google to do a similar technique in Android because they have billions of users that they can aggregate data from.

Another possible feature would be to have a layer of hard coded logic, for example if it's a public holiday it might use Sunday's predictions instead. An alternative to this would be to have another feature/s that feed in whether it's a public holiday or other special time of years. This neural network could also trivially be changed to use any of the fields, for example temperature and humidity, to learn predictive behaviours around them.

10.7.2.1 Uses for Location Prediction

If the users can train an accurate model, there are numerous use cases for this.

- The model could feed back into Rabbit MQ/PostgreSQL as another field in the UI, giving a new prediction every 20 min for an hour in the future.
- This could be then implemented into other smart devices. Such as automatically turning on the air conditioner or coffee machine just before the user is predicted to come home.
- It could remind the user to do difficult tasks, such as going to the gym.

Suggested Reading

1. N.K. Suryadevara, S.C. Mukhopadhyay, in *Smart Homes: Design, Implementation and Issues*, vol. 14. Smart Sensors, Measurements and Instrumentation (Springer, 2015). ISBN: 978-3-319-13556-4
2. H. Ghayvat, S.C. Mukhopadhyay, in *Wellness Protocol for Smart Homes: An Integrated Framework for Ambient Assisted Living* (Online), vol. 24. Smart Sensors, Meas. and Instrumentation (Springer, 2016). ISBN: 978-3-319-52047-6 (Print) 978-3-319-52048-3

3. S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 9. Internet of Things: Challenges and Opportunities (Springer, 2014). ISBN 978-3-319-04222-0
4. R. Yan, X. Chen, S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 26. Structural Health Monitoring: An Advanced Signal Processing Perspective (Springer, 2017). ISBN 978-3-319-56125-7
5. O.A. Postolache, E. Sazonov, S. Chandra Mukhopadhyay (eds.), *Sensors in the Age of the Internet of Things: Technologies and applications, IET Book*. https://digital-library.theiet.org/content/books/ce/pbce122e. ISBN: 9781785616341, e-ISBN: 9781785616358 (2019)
6. N.K. Suryadevara, S.C. Mukhopadhyay (eds.), *Assistive Technology for the Elderly* (Elsevier, 2020). ISBN: 978-0-12-818546-9
7. A.K. Sangaiah, S.C. Mukhopadhyay (ed.), *Intelligent IOT Systems in Personalized Health Care* (Elsevier, 2021). ISBN: 978-0-12-821187-8
8. C. Wang, M. Daneshmand, M. Dohler, X. Mao, S.C. Mukhopadhyay, R.Q. Hu, H. Wang (eds.), Special issue on internet of things: architecture, protocols and services. IEEE Sens. J. **13**(10) (2013). ISSN 1530-437X
9. R. Yan, G. Tian, S.C. Mukhopadhyay, in *Special issue on intelligent sensing and information mining*, http://www.mdpi.com/journal/sensors/special_issues/10th_ICST. A special issue of Sensors (ISSN 1424-8220) (2017)
10. S.C. Mukhopadhyay, A. Sangaih (eds.), in *AI Enabled Communications on IoT Edge Computing*. A special issue of Electronics (ISSN 2079-9292) (2020)
11. Z. Hussain, I. Ahmed, S.C. Mukhopadhyay (eds.), *Signal Processing Techniques for Smart Sensor Communications.* A special issue of Sensors (ISSN 1424-8220) (2020)
12. A. James, A. Seth, S.C. Mukhopadhyay, IoT enabled sensor node: a tutorial paper. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–18 (2020)
13. A. Seth, A. James, S.C. Mukhopadhyay, 1/10th scale autonomous vehicle based on convolutional neural network. Int. J. Smart Sens. Intell. Syst. **13**(1), 1–17 (2020)
14. N.T.P. Van, L. Tang, S.F. Hasan, S. Mukhopadhyay, N.D. Minh, Combination of artificial intelligence and continuous wave radar sensor in diagnosing breathing disorder. Intell. Comput. Eng. 853–863 (2020)
15. F. Akhter, S. Khadivizand, H.R. Siddiquei, M. E.E. Alahi, S. Mukhopadhyay, IoT enabled intelligent sensor node for smart city: pedestrian counting and ambient monitoring. Sensors **19**, 3374 (2019). https://doi.org/10.3390/s19153374 (19 pages)
16. H. Ghayvat, M. Awais, S. Pandya, H. Ren, S. Akbarzadeh, S.C. Mukhopadhyay, C. Chen, P. Gope, A. Chouhan, W. Chen, Smart aging system: uncovering the hidden wellness parameter for well-being monitoring and anomaly detection. Sensors, **19**, 766 (2019). https://doi.org/10.3390/s19040766, www.mdpi.com/journal/sensors (32 pages)
17. G. Zheng, R. Shankaran, W. Yang, C. Valli, L. Qiao, M.A. Orgun, S.C. Mukhopadhyay, A critical analysis of ECG-based key distribution for securing wearable and implantable medical devices. IEEE Sens. J. **19**(3), 1186–1198 (2019)
18. A. Nag, M.E.E. Alahi, S. Feng, S.C. Mukhopadhyay, IoT-based sensing system for phosphate detection using Graphite/PDMS sensors. Sens. Actuators A **286**, 43–50 (2019)
19. IoT and ML based Electrical Fault Predictor using Bolt IoT. https://www.hackster.io/tunirdas/iot-and-ml-based-electrical-fault-predictor-using-bolt-iot-4dcd6f
20. Machine Learning Smart Inventory Tracking with Raspberry Pi. https://www.hackster.io/supperted825/machine-learning-smart-inventory-tracking-with-raspberry-pi-d866c7
21. Machine Learning with Azure and XinaBox. https://www.hackster.io/PragmaticPhil/machine-learning-with-azure-and-xinabox-692cfa
22. Auto Lightning Using IoT and ML. https://www.hackster.io/analegaonkar/auto-lightning-using-iot-and-ml-72269a

Chapter 11
Simulation Based Projects on IoT Systems

11.1 Introduction

Systems and networks for the Internet of Things (IoT) are getting extremely widespread, complex, diverse, and pervasive. They incorporate a wide range of physical devices (IoT devices and sensors) that communicate via various networking connections (cellular, WiFi) and are distributed across several design levels (cloud, fog, edge). IoT systems, in other words, span both virtual and physical realms. Designing and testing IoT services is difficult due to the vast scale and variety of IoT devices and networks. During the early design phase, prototyping with a large number of hardware nodes may not be feasible. Benchmarking and building up repeatable experiments are also difficult performing jobs. To this regard, simulation-based techniques are believed to be critical for benchmarking, developing, testing, and experimenting with IoT systems and networks.

We are amid a technical and digital revolution. A significant shift in the world around us has been observed in less than a decade. Currently, there are AI-powered smart assistants, driverless cars, surgical bots, intelligent cancer detection systems, and, of course, the Internet of Things at everyone's disposal, thanks to recent breakthroughs in Data Science (IoT).

Users may use IoT simulators to develop, construct, and test IoT programs and devices without having to utilize real IoT boards.

In circumstances to test the feasibility of an IoT system before purchasing expensive hardware to implement it, simulation is very useful. In many countries resources to develop an IoT system such as hardware boards, sensors, transmission modules may not be easily available. In such circumstances it is helpful to simulate and develop a project that is entirely online. Listed below are the few points explaining how online tools/simulators may be helpful:

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_11

- The user needs the finest IoT simulators to build and test apps without utilizing real IoT boards.
- In the virtual IoT lab, the user may mimic large-scale IoT deployments using the tools.
- Another tool allows the user to create and test smart devices in the cloud, as well as gather and analyse IoT data.
- The user may also use one of the programs on our list to simulate a virtual network and evaluate the performance of actual apps.

11.2 Example of Online Simulator: AWS

The infrastructure as a service (IaaS) Amazon Web Services (AWS) offers a variety of services that allow clients to develop IoT-based database programs that collect, organize, analyse, and operate smart devices with no requirement of developing their own infrastructure, that might help lower expenses and enhance creativity and productivity. However, testing of IoT and backend operations without a wide range of interconnected devices might be a hurdle.

AWS provides the IoT Device Simulator platform to assist businesses to better access device interoperability and IoT SaaS applications. This system delivers a Graphical User Interface console that allows users to build and simulate multiple virtually smart objects without configuring and managing actual devices or using tedious algorithms.

The AWS service for simulation process for devices that enables users to create from a user-defined framework a broad range of virtual network of devices (widgets) and simulate data-published apps/widgets to AWS IoT on a periodic basis. One may also test specific controls from the simulator or examine when data is processed by backend services. Figure 11.1 shows the architecture of the AWS IoT Device as illustrated in 2021.

The architecture contains an API device simulator that uses Amazon API Gateway to call the subsystems called microservices of the solution (AWS Lambda functions). Such microservices are the core of the organization that performs tasks on virtual devices and device kinds, collects and administers simulation measures. Amazon API Gateway calls the relevant Lambda function if the device simulator API has an approved request.

If an information is approved for simulation of the device, the device transmits the query in Amazon Simple Queue Service to a simulated queue (Amazon SQS). The sim queue use FIFO format that processes tasks processed by Amazon ECS Elastic Container Service containers enabled by AWS Fargate.

The simulation engine will launch a virtual device, where simulated data is published to the AWS IoT endpoint, when the starting request reception is processed. Once the set period has elapsed, the simulation system ends the simulation, stops the virtual device and changes Amazon DynamoDB device state and metrics.

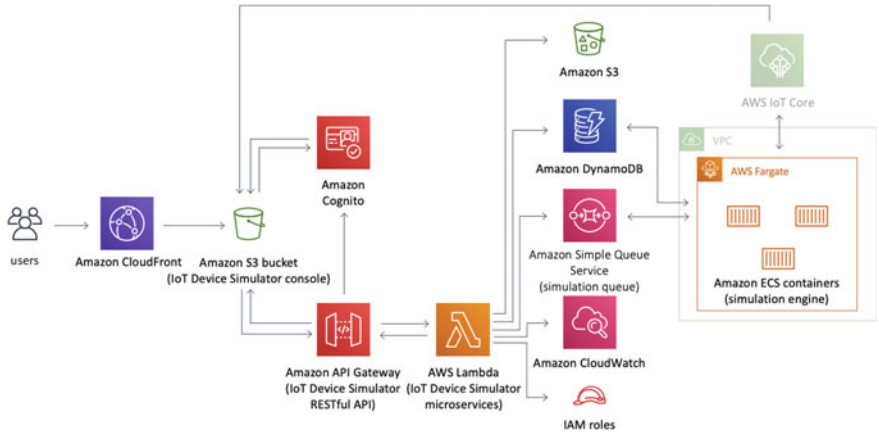


Fig. 11.1 The AWS IoT device simulator architecture [1]

A simulation console gives details on the kind for the simulation states of virtual devices, and user accounts. To build and end virtual devices, to begin and finish simulations, and to examine measurements one utilises the terminal.

11.3 Example of Online Simulator: Microsoft Azure

Microsoft Azure IoT technology solutions give an opportunity to develop a vast array of IoT & Cloud products that allow a company to digitally modernize their systems. The IoT management platform Azure IoT Central enables a reliable IoT solution to be built and deployed onto existing systems. To speed up the application design task, IoT Central offers a variety of industry-specific frameworks, for e-commerce and medical care. The customised IoT strategy may be built from the ground up using Azure platform services such as the Azure IoT Hub and the Azure IoT device SDKs. Figure 11.2 shows the Azure IoT technologies, services, and solutions.

Microsoft Azure IoT Hub is a maintained cloud service that serves as a key messaging hub between an IoT platform and its associated devices enabling connectivity messages in both ways. One can safely and effectively link thousands of devices and associated background operations. Virtually any device may be linked to an IoT hub.

Various communication protocols are available, including device-to-cloud telemetry, device upload files and cloud request response techniques. IoT Hub also enables tracking to let you monitor device generation, device connection and equipment issues.

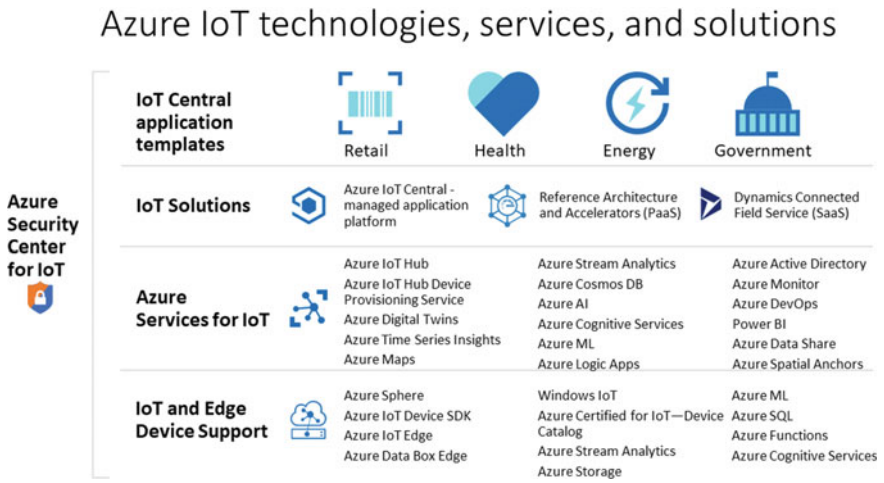


Fig. 11.2 The Azure IoT technologies, services, and solutions [2]

With IoT Hub capacity, it is possible to create scalable IoT applications, complete with features, including management of manufacturing industrial machinery, monitoring of important medical properties and control of the use of commercial buildings.

11.4 Example of Online Simulator: CupCarbon

CupCarbon is a simulating software that uses a mapping smart city environment to create IoT based Wireless Sensor Networks (SCI-WSN) applications. Its goal is to develop, display, test and evaluate networked analysis techniques, ambient data collecting, etc.; and construct ecological scenarios such as wildfire, air quality, cell phones, etc. In addition, scientists can also be encouraged to examine their Wireless Topologies, protocols etc. in graphically explaining the essential principles and the way they function in sensor nodes. Two simulation settings are offered by CupCarbon.

Figure 11.3 shows an example of a CupCarbon project created with multiple sensors in the Macquarie University area.

The initial mobility screens are designed and natural occurrences such as combustion and gas may be generated as well as a simulation environment for mobiles such as cars and airborne devices (e.g., Drones, Quadrotors, etc.). The next simulation tool provides a discreet occurrence simulation of WSNs that takes on board the first setting model. The project created has 9 sensor nodes placed in a distributed environment around Macquarie University. The nodes can be configured to perform various functions with different protocols for wireless communication like LoRa, WiFi, and XBee.

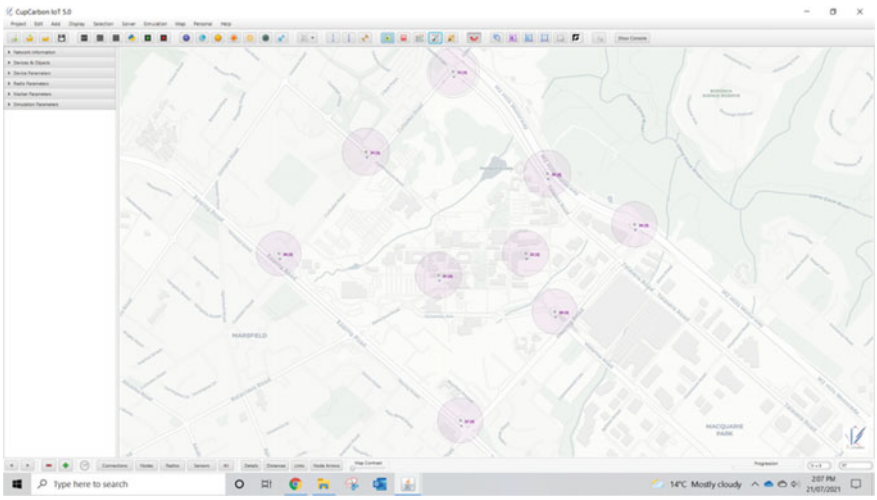


Fig. 11.3 An example of a CupCarbon Project Simulation at Macquarie University

An ergonomic interface with an OpenStreetMap (OSM) framework may be used to build and optimised networks for the immediate deployment of sensors. It has a script called SenScript that enables every sensor node to be independently programmed and configured. From this script codes for operating systems like Arduino/XBee may also be generated. This component is not fully developed in CupCarbon, it enables rudimentary connection and algorithm coding. Guidelines and sample codes can be found on the link given in [12].

Thus, CupCarbon is a great tool to get started into simulating IoT based projects.

11.5 Brief Guidelines to User

For an online user, it is not possible to acquire or have access to all hardware components to build an IoT system. However, it is possible to build a simulation system to propose a functioning IoT system.

One such online simulator is: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-raspberry-pi-web-simulator-get-started.

In the above link, the simulator uses Raspberry pi and interfaces it with a sensor and displays the data on Microsoft Azure as briefed in the earlier section. The implementation is illustrated in the next section. There is not many free software available for Raspberry pi and Arduino specifically.

One sample platform is: http://www.cupcarbon.com/.

Another platform is: https://www.opensourceforu.com/2018/12/ifogsim-an-open-source-simulator-for-edge-computing-fog-computing-and-iot/.

This is only a sample platform; the user is free to use and simulate in any similar platform.

The scope of the project includes but is not limited to the following:

1. Define the IoT System:
2. Choose the Online Simulator:
   a. Define the Microcontroller/Processor to be used
   b. Define the sensors to be used
   c. Define the transmission medium to be used (WiFi/Bluetooth)
   d. Define the Cloud Server
   e. Define the User Dashboard to view the data received.
3. Display the results in project Demonstration.

11.6 Implementation of Simulation Based IoT Project

Title: Cloud Computing with Microsoft Azure IoT Hub.

11.6.1 Introduction and Classification

The purpose of this task was to create an Azure IoT Hub and perform the steps necessary to visualise data within the hub.

The notes given in the link given earlier provided information on how to understand a Cloud Computing offering by its classification. It makes sense as a first step to classify the IoT Hub. The IoT Hub provides the user with control of platform parameters such as tier selection.

Therefore, it seems reasonable to classify the solution as Platform-as-a-Service (PaaS). Supporting this classification are the following observations:

- Direct user access is not provided to the underlying hardware, such as servers and OS. Therefore, IoT Hub is not classifiable as an Infrastructure-as-a-Service (IaaS) offering; similarly
- Applications and Data are not managed by Azure. We were responsible for setting up the applications to visualise our data. Therefore, IoT Hub is not classifiable as a Software-as-a-Service (SaaS) offering.

Completing tasks in this section of the lab involved following three tutorial procedures provided by Microsoft Azure as given in these three links:

- link 1: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-raspberry-pi-web-simulator-get-started

- link 2: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-vscode-iot-toolkit-cloud-device-messaging
- link 3: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-live-data-visualization-in-web-apps

11.6.2 Setup of the Hub

Setup of the IoT Hub involved performing the following tasks:

- Creating an IoT Hub in the Azure Portal;
- Registering the Raspberry Pi Simulator with IoT Hub; and
- Using the Raspberry Pi Simulator to send sample data into the IoT Hub.

Figure 11.4 shows the operation of the Raspberry Pi simulator once it had been configured with the correct connection string for our IoT Hub.

11.6.3 VS Code Extension for IoT Hub

After configuring the simulator to send data to our IoT Hub, we installed the IoT Hub Extension for Visual Studio Code. This provided us with a console to monitor the messages being sent from the simulator to the IoT hub. The Extension also

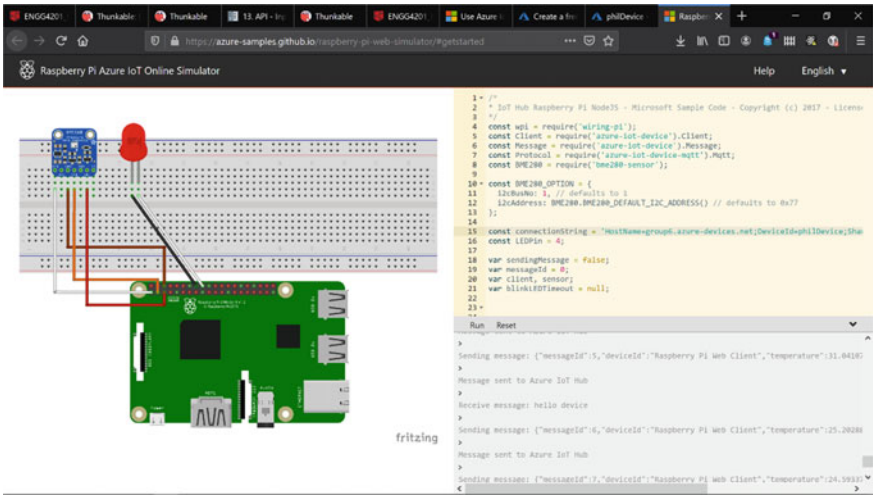


Fig. 11.4 The screenshot of Raspberry Pi Simulator

provided us with an interesting mechanism to send data in the reverse direction (from IoT Hub to the simulator). Both messages are shown in the "Output" tab of Fig. 11.5.

After configuring the simulator Note that the messages, the users sent from IoT can also be seen in the console output of the Simulator, in Fig. 11.4 (circled).

11.6.4 Web App Visualisation

The VS Code Extension provided confirmation of data sent to IoT Hub from the simulator but provided no way to visualise that data. Fortunately, a web app was available that allows data visualisation.

Producing this visualisation involved the following steps:

- Installation of NodeJS software on the laptop.
- Configuration of a Consumer group within IoT Hub.
- Download of a web app written in JavaScript to visualise the simulated data; and
- Building and running the web app on the laptop and using a browser to view the data.

Figure 11.6 shows the visualisation of our data via the web browser. The visualisation updated as new data was sent to our IoT Hub via from the simulator.

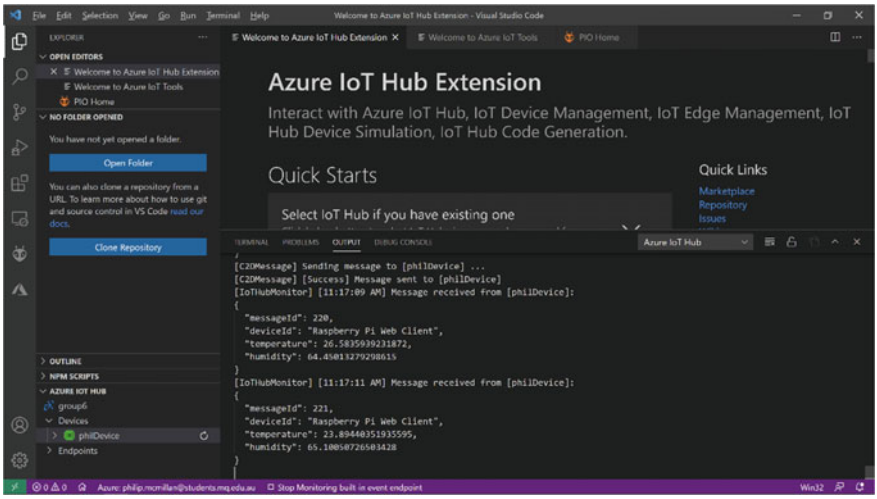


Fig. 11.5 The Monitoring Sent and Received Messages at IoT Hub using VS Code Extension

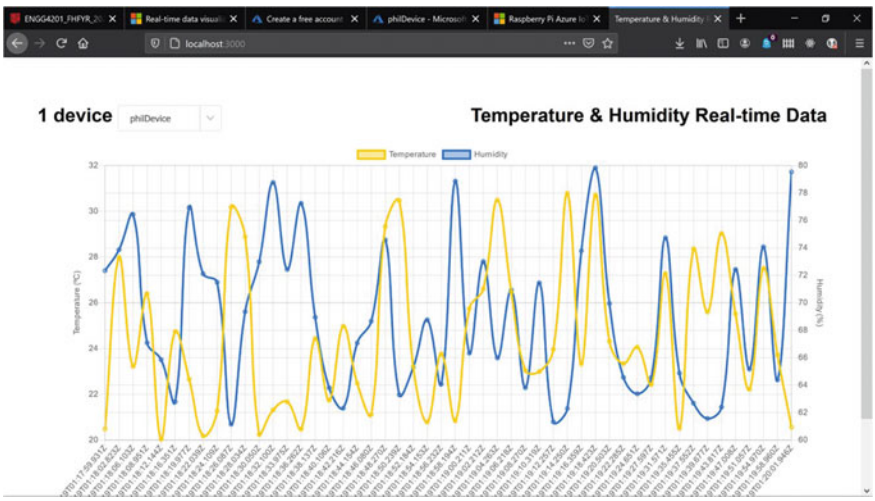


Fig. 11.6 The screenshot of Web App for Visualisation of IoT Hub Data

11.6.5 Challenges

No major challenges were experienced in completing the tutorial. However, substantial additional effort would be required to understand all the features supported, such as integrations with other Azure services, and to become proficient in developing apps that process and visualise the platform data.

11.6.6 Possible Future Experiments

- It would be useful to re-run the tutorial without the simulator and use an actual Raspberry Pi and sensor. This would provide additional experience in integrating a sensor using the I2C protocol, and the use of a new sensor type (BME280) for Temperature/Pressure/Humidity.
- It would be interesting to compare the IoT capabilities of JavaScript applications on Raspberry Pi against our experience with Python; and
- The user could further explore the usefulness of being able to send data in "reverse" (from the IoT Hub to a device). They also noted this "reverse" capability is supported by The Things Network.

Suggested Reading

1. AWS IoT Device Simulator architecture. Website Link: https://aws.amazon.com/solutions/implementations/iot-device-simulator/
2. Microsoft Azure IoT. Website Link: https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-services-and-technologies
3. N.K. Suryadevara, S.C. Mukhopadhyay, Internet of things: a review and future perspective. Eur. Bus. Rev. 18–20 (May–June 2014). http://www.europeanbusinessreview.com/?p=4431
4. N.K. Suryadevara, S.C. Mukhopadhyay, R. Wang, R.K. Rayudu, Forecasting the behavior of an elderly using wireless sensors data in a smart home. Eng. Appl. Artif. Intell. **26**(10), 2641–2652 (2013). ISSN 0952-1976. https://doi.org/10.1016/j.engappai.2013.08.004
5. S.D. Tebje Kelly, N.K. Suryadevara, S.C. Mukhopadhyay, Towards the implementation of IoT for environmental condition monitoring in homes. IEEE Sens. J. **13**(10), 3846–3853 (2013)
6. N.K. Suryadevara, S.C. Mukhopadhyay, Wireless sensor network based home monitoring system for wellness determination of elderly. IEEE Sens. J. **12**(6), 1965–1972 (2012)
7. N.K. Suryadevara, A. Gaddam, R.K. Rayudu, S.C. Mukhopadhyay, Wireless sensors network based safe home to care elderly people: behaviour detection. Sens. Actuators A: Phys. **186**, 277–283 (2012). https://doi.org/10.1016/j.sna.2012.03.020
8. R. Singh, A. Gehlot, L.R. Gupta, B. Singh, M. Swain, Internet of Things with Raspberry PI and Arduino (CRC Press, Taylor & Francis Group, 2020). ISBN 978-0-367-24812-5
9. S.S. Iyengar, N. Parameshwaran, V.V. Phoha, N. Balakrishnan, C.D. Okoye, Fundamentals of Sensor Network Programming (Wiley, IEEE Press, 2011). ISBN 978-0-470-87614-5
10. N. Mukherjee, S. Neogy, S. Roy, Building Wireless Sensor Networks, Theoretical and Practical Perspective (CRC Press, Taylor & Francis Group, 2016). ISBN 978-1-4822-3006-2
11. V. Sharma, A. Pughat (eds.), Energy Efficient Wireless Sensors Networks (CRC Press, Taylor & Francis Group, 2018). ISBN 978-1-4987-8334-7
12. CupCarbon IoT Simulator. Website Link: http://www.cupcarbon.com/

Chapter 12
Projects on IoT Systems

12.1 Introduction

Three projects on IoT systems design are presented in this chapter.

12.2 Project 1: Wireless Sensor Node for Precision Agriculture

12.2.1 Abstract

A Wireless Sensor Node for Precision Agriculture is described, comprising sensors for soil moisture, soil nutrient levels (NPK), ambient light, temperature, humidity, and pressure. It also features a solenoid valve for drip irrigation, which is controlled wirelessly from the cloud. An Arduino microcontroller is used, with bi-directional LoRaWAN communication to The Things Network. Dashboards for Monitoring and Analytics are also enabled via integrations to the Tago.IO and ThingSpeak web applications.

12.2.2 Introduction

Precision Agriculture can be defined as "a management strategy that employs information technology to improve quality and production" [1]. It is particularly applicable to situations where resources such as water can be scarce [2]. The node is implemented inside a plastic enclosure with external sensor and power connectors. Figure 12.1 shows the configuration of the developed node.

A. James et al., *IoT System Design*, Smart Sensors, Measurement and Instrumentation 41, https://doi.org/10.1007/978-3-030-85863-6_12

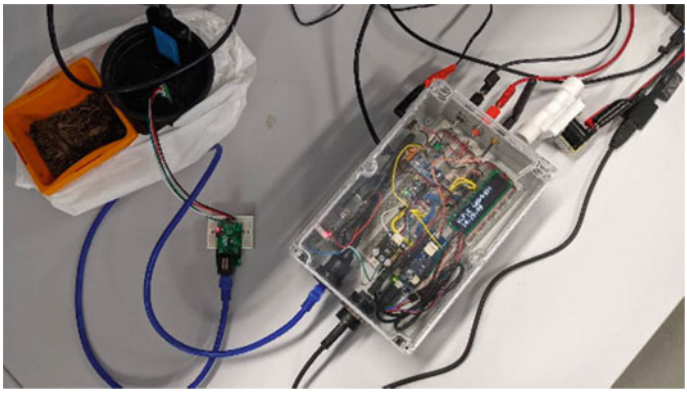


Fig. 12.1 The photo of the developed Precision Agriculture node

The developed nodes help farmers to optimise consumption of inputs, while maximising outputs, is a key component of a strategy that can "fulfil the food requirements of the present generation" [3]. The sensors interface to an Arduino-based microcontroller via a variety of analogue and digital protocols (I2C, SPI and RS-485). Additionally, a solenoid valve actuator is interfaced to support automation: a Machine Learning algorithm can utilise sensor data and then predict a schedule for irrigation. The ultimate vision is for "intelligent irrigation", or one that "reduces human labour to a great extent" [3]. The communications protocol selected is LoRaWAN, considered appropriate for Precision Agriculture due to its long range, low power and security mechanisms [1]. The choice of TTN as LoRaWAN network provides out-of-the-box integrations with Tago.IO and ThingSpeak, allowing rapid creation of dashboards for visualisation and data analytics.

12.2.3 Block Diagram

The block diagram for the node is shown in Fig. 12.2.

12.2.4 Sensors

The Precision Agriculture Node utilises the following sensors:

1. BME280 Temperature, Pressure, Humidity Sensor
2. Adafruit STEMMA Capacitive Soil Moisture Sensor

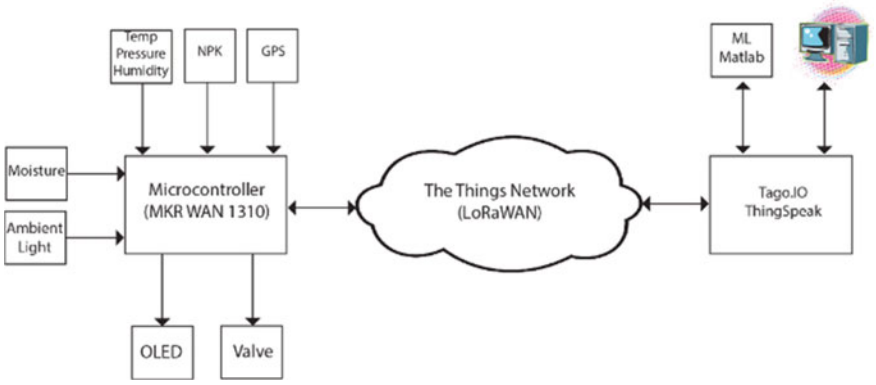


Fig. 12.2 The final block diagram Precision Agriculture node block diagram

3. IP68 NPK Soil Sensor
4. LDR Light Sensor
5. 1/2″ Water Solenoid Valve

Interfacing the BME280 requires installation of the Adafruit BME280 library [4]. This library handles the SPI configuration and communication as well as the sampling rate for the different values measured. Using the default sampling rate, a measurement is taken approximately every 13.2 ms and stored in an internal data register.

Once the BME280 is initialised with the begin() function, a reading for each measured value is taken from the internal register by calling the respective function; readTemperature(), readPressure(), readHumidity(), and readAltitude(). Note that readAltitude() requires the current sea level pressure (in hPa) to be passed as a parameter. This value may be manually calculated and entered for each Altitude reading. The serial output is shown in Fig. 12.3.

The Soil Moisture Sensor utilizes an ATSAMD10 chip to provide capacitive touch measurement and allows I2C communication through the Seesaw framework. This sensor requires installation of the Seesaw Library [5].

Fig. 12.3 The screenshot of the serial monitor for BME280 serial output

To get a reading of the current value of the probe, the Seesaw touchRead() function is used. This function returns the current capacitive reading of the probe as a 16-bit unsigned integer. When placed in soil, we obtained values between 300 and 1016, depending on the moisture content as shown in Fig. 12.4.

The LDR light sensor measures the intensity of ambient light with a photo-transistor, increasing voltage output with an increase in light intensity. Implementation of this sensor is very simple. The data pin is connected to A0 on the MKR WAN, and an analogue read of that pin returns the data value. During testing we received values between 0 (sensor covered) to around 1300 (LED torch pointed directly at sensor), shown in Fig. 12.5.

The NPK sensor requires an RS485 interface, so we are connecting it to the Arduino through a TTL UART to RS485 converter module. UART is configured using the Arduino serial1 interface. The NPK sensor has a different serial address (sent as an inquiry frame) configured for each nutrient probe. These addresses are shown in Table 12.1.

Fig. 12.4 The screenshot of the serial monitor for soil sensor serial output

Fig. 12.5 The screenshot of the serial monitor for LDR serial output

Table 12.1 Inquiry frames for NPK sensor

	Address code	Function code	Register start	Register length	CRC_L	CRC_H
Nitrogen	0x01	0x03	0x00 01E	0x00 0x01	0xE4	0x0C
Phosphorus	0x01	0x03	0x00 0x1F	0x00 0x01	0xB5	0xCC
Potassium	0x01	0x03	0x00 0x20	0x00 0x01	0x85	0xc0

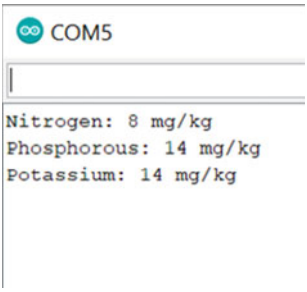


Fig. 12.6 The screenshot of the serial monitor for NPK sensor serial output

To receive a reading from the NPK sensor, we first write the inquiry frame to the serial1 interface. The response from the NPK sensor will be returned in a response frame with the current reading of the probe in the 5th byte position. Note that the values are returned in hexadecimal format so the HEX parameter may be passed to print()so that the results are interpreted correctly. An example of the serial output is shown in Fig. 12.6.

For sensor feedback at the node, we have included a 16 × 2 Newhaven OLED display interfaced through the I2C bus. The display is managed with the NewOLED library, a custom-created library based on example code for Newhaven OLEDs.

The screen displays the values obtained from two sensors at any one time and cycles between screens every 10 s. Figure 12.7 shows an example of the sensor reading display.

For ease of readability, we compressed the serial output of each sensor into a single line as shown in Fig. 12.8. The OLED display acts as the primary feedback method for the Node with the serial output used primarily for troubleshooting.

12.2.5 Libraries

Two libraries have been created to support the node's major functions, as shown in Table 12.2. Both libraries have been added to GitHub:

The libraries have been created according to Arduino guidelines, and include recommended features such as keyword highlighting.

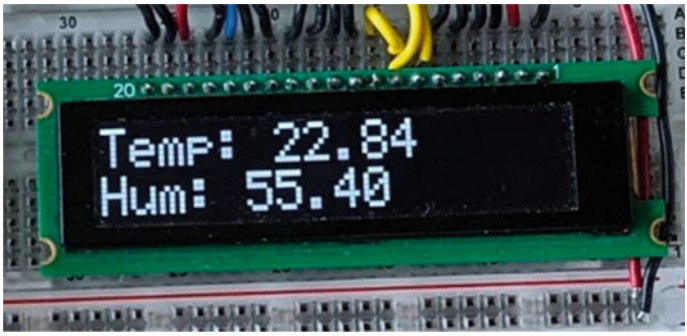


Fig. 12.7 The image highlighting the OLED display

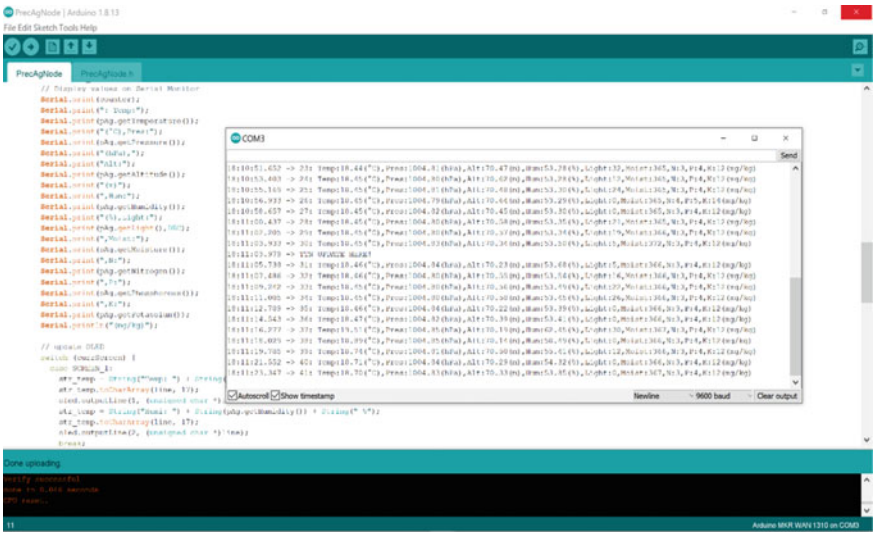


Fig. 12.8 The final screenshot of the combined sensor serial output

12.2.6 Timing

A lot of effort was spent refining the timing within the sketch's loop(). We avoided the use of the delay() function. Our first choice was to attempt the use of SAMD timer interrupts as described in [7], however conflicts were encountered with the serial protocol used by the NPK sensor. We opted instead for the use of millis() within the loop() function of the main sketch. Due to the use of a serial-connected sensor (the NPK sensor provides an RS-485 connection), we required a state machine to provide the precise control over timing required by that sensor.

Table 12.2 Libraries to support node functions

Library	Function	Github repository
PrecAg	Provides a class that takes readings from each of the sensor types attached to the node. Provides functions that can be called from the main sketch to return sensor values for display on Serial Monitor and OLED, or for sending in TTN updates	https://github.com/philmcm/PrecAgSensor
NewOLED	Library to support the Newhaven OLED used in the project. Sample code was taken from Newhaven's example sketch (see [6]) and re-implemented as a library	https://github.com/philmcm/NewOLED

Table 12.3 State summary

State	Description
READ_SENSORS	Read values of all sensors
DISPLAY_VALUES	Display values on Serial Monitor
CHANGE_OLED_SCREEN	Change currently displayed value on OLED screen
UPDATE_TTN	Encapsulate data and send to TTN; read downlink message from TTN
NITRO_REQ, NITRO_RESP	Send nitrogen inquiry frame/Receive nitrogen response frame
PHOS_REQ, PHOS_RESP	Send phosphorus inquiry frame/Receive phosphorus response frame
POTA_REQ, POTA_RESP	Send potassium inquiry frame/Receive potassium response frame

Each state transition is triggered at a set interval that is calculated based on elapsed time. At the start of the main loop the value of millis() is copied into a currentMillis variable. Time elapsed is then determined by subtracting the previousMillis value from the currentMillis value and compared to the interval. If elapsed time is greater than our defined interval, a state change is initiated. A summary of the different states used is outlined in Table 12.3.

12.2.7 Circuitry

All connected peripherals except for the solenoid valve operate at a logic level of either 3.3 or 5 V that can be supplied by the Arduino. The solenoid valve used by our node requires 12VDC supply and consumes 4.8 W of power. This exceeds the specifications of the Arduino board, additional components are required for interfacing. Shown in Fig. 12.9, the IRLB8721 MOSFET is used to perform the switching. A flyback diode (1N4004) is implemented in parallel with the solenoid

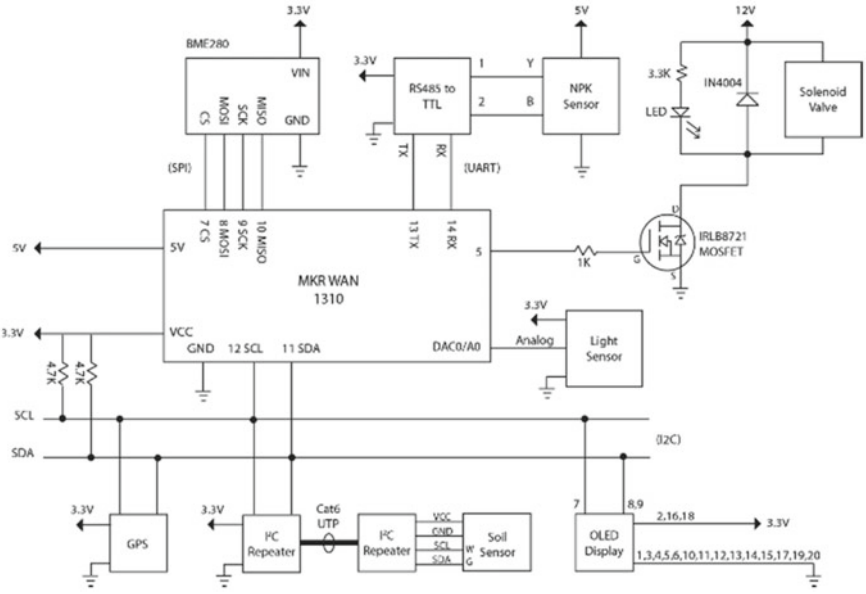


Fig. 12.9 The final precision agriculture node circuit diagram

terminals which prevents damage to the MOSFET when the solenoid is switched off. Finally, an LED and appropriate current-limiting resistor is included to provide a visual indication of the solenoid valve's state.

The OLED display, GPS shield and Soil sensor communicate on the I2C bus. A 4.7 kΩ pull up resistor is implemented on both the SCL and SDA lines to so that the voltage may be driven high after transmission of a data frame. The soil sensor's range is extended to approximately 30 m with the use of two QwiccBus EndPoint repeaters. These repeaters split the SCL and SDA lines into two separate differential signals each for transmission over CAT6 UTP.

12.2.8 Transmission Protocol

Our node uses LoRaWAN as its transmission protocol. We deployed an application in TTN (see Fig. 12.10) to manage communication with our node and perform the required integrations for visualisation and analytics.

Uplinks: Sending Sensor Data Into the Cloud.

CayenneLPP was the payload format chosen to deliver data from our sensor to our application. The variables sent are as shown in Table 12.4, with the code for the sketch and TNN decoder in the Appendix. (Decoder names are chosen to be compatible with the ThingSpeak application).

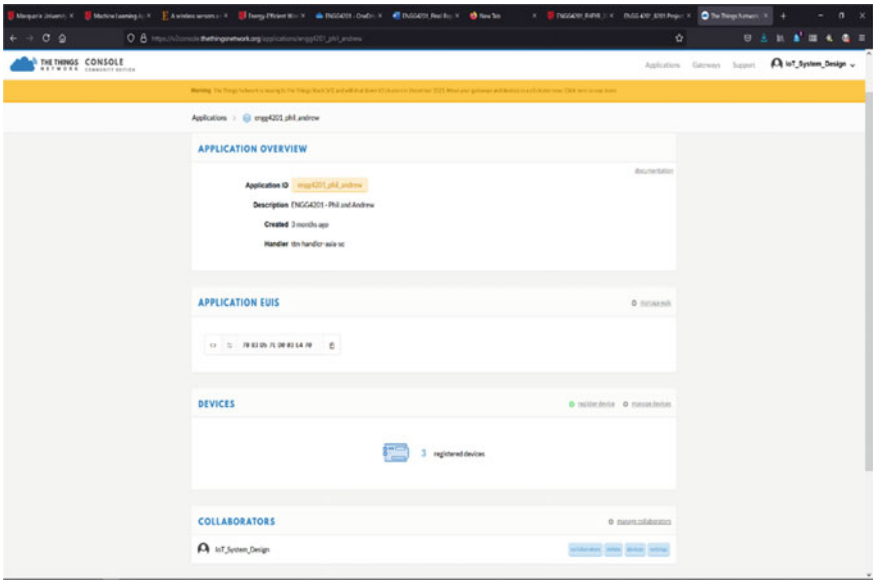


Fig. 12.10 The final screenshot of the application within The Things Network

Table 12.4 Variables sent from our node to the TTN application

Decoder name	Sensor	CayenneLPP type (see [8])
field1	Temperature (BME280)	Temperature Sensor
field2	Barometric Pressure (BME280)	Barometer
(not used)	Altitude (BME280)	Analogue Output
field3	Humidity (BME280)	Humidity Sensor
field4	Ambient Light (Gravity Light)	Illuminance Sensor
field5	Soil Moisture (Soil Sensor)	Analogue Output
field6	Nitrogen (NPK)	Analogue Output
field7	Phosphorous (NPK)	Analogue Output
field8	Potassium (NPK)	Analogue Output

Downlinks: Receiving Solenoid Valve Control Messages from the Cloud. The downlink uses a very basic payload format:

```
<openHours><openMins><openSecs><closeHours><closeMins><closeSecs>
```

Figure 12.11 shows the processing of downlinks sent from the TTNv2 console by the node.

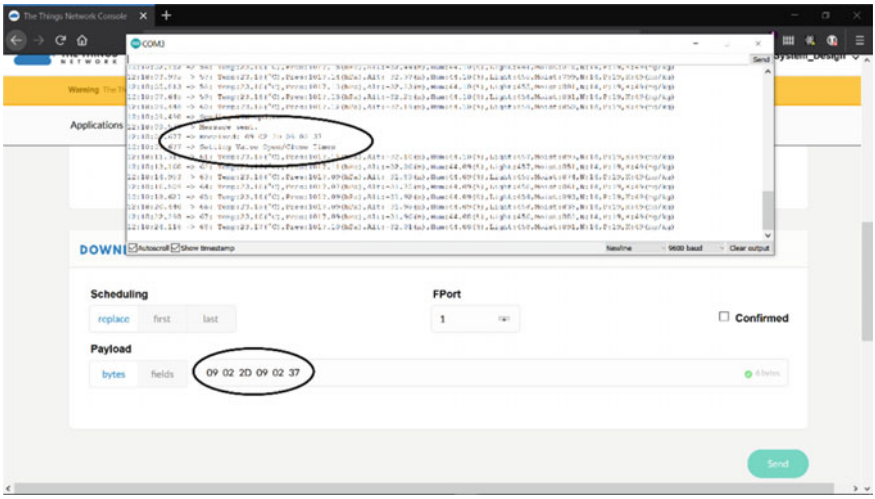


Fig. 12.11 The final screenshot of the Downlink Processing

12.2.9 Dashboard

The figures such as Figs. 12.12, 12.13 and 12.14 below show actual real-time and historical sensor data from our node:

We have implemented LoRaWAN "downlink" messages to allow control of the solenoid valve from the cloud. Rather than allow direct "on/off" control of the valve remotely, we considered it better from a security standpoint to send a schedule of times for which the valve will turn on and off. This would address a scenario where a "denial of service" might prevent an open valve from ever closing, risking damage to crops by flood.

Due to out-of-the-box application integrations within TTN, configuration of dash-boards in both in Tago.IO and ThingSpeak to show both real-time and historical sensor data is trivial.

12.2.10 Algorithm

The main sketch (PrecAgNode.ino and PrecAgNode.h) performs the following functions:

1. Includes the NewOLED library for managing the OLED display;
2. Includes the PrecAg library for managing the sensors connected to the node (which in turn includes the libraries provided by the manufacturers of the sensors);

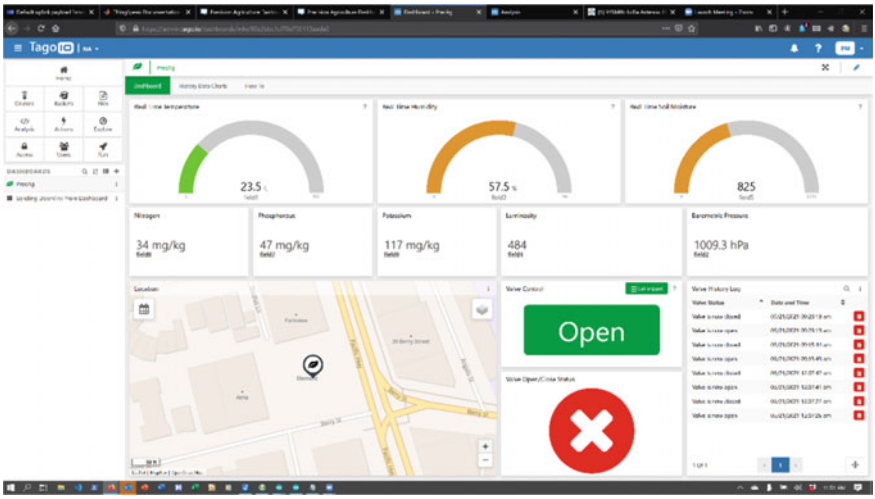


Fig. 12.12 The final screenshot of the Tago.IO dashboard, showing real-time sensor data

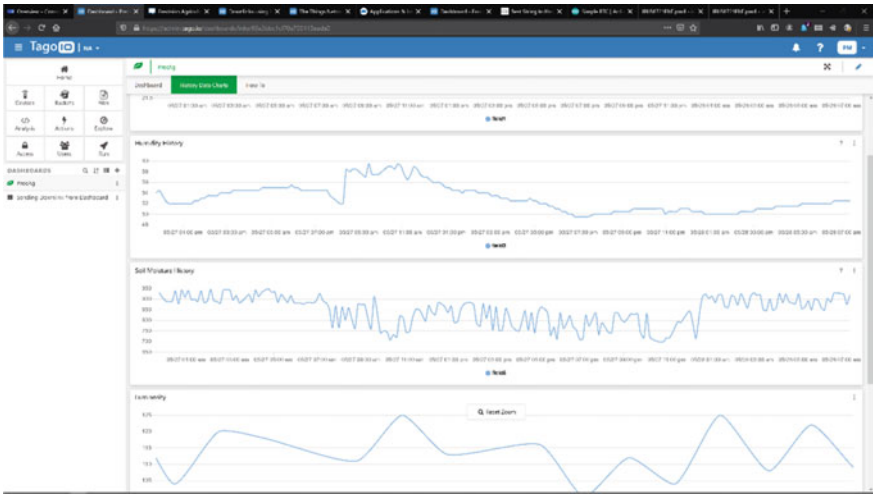


Fig. 12.13 The final screenshot of the Tago.IO historical sensor data dashboard

3. Implements a state machine for managing serial communications with the NPK sensor and overall co-ordination of system timing;
4. Implements "screen change" functionality for the OLED display;
5. Calls the function for sending/receiving TTN uplinks and downlinks; and
6. Controls the frequency of sensor readings and TTN updates using macros in the header file.

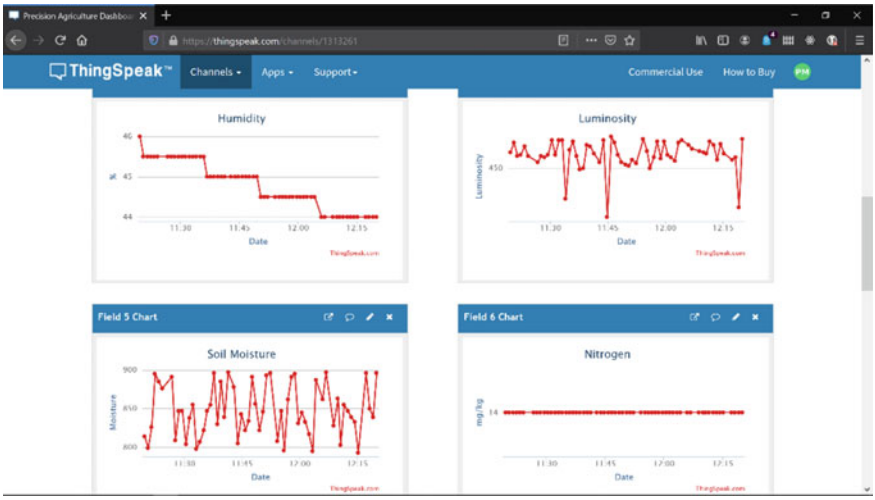


Fig. 12.14 The final screenshot of the ThingSpeak dashboard

The state machine developed for controlling node functions is illustrated in Fig. 12.15 below. The frequency of the OLED screen changes and TTN updates is configurable within the header file.

12.2.11 Edge Computing

The Arduino microcontroller has basic capabilities for Edge Computing, which we leverage in the following two ways.

Real Time Clock for Solenoid Valve Activation:

We have implemented the real-time clock feature of the Arduino's SAMD microcontroller using the Arduino RTCZero library as seen in Fig. 12.16. Our application continually monitors the current time against the schedule received from TTN in "down-link" messages (refer "Transmission Protocol" section), and we have verified opening and closing of the valve at the precise times required.

Multiple Value Averaging of Soil Moisture Data.

The "Soil Moisture History" graph in Fig. 12.14 shows substantial variability in the values returned by the Soil Sensor. Another application for edge computing is the "smoothing" of this data through an averaging technique. Our node implements aver-aging of the previous five values of soil moisture.

This helped reduce the variability between successive soil moisture readings. Note that the number of samples that are averaged is configurable via a macro in the header file for our library.

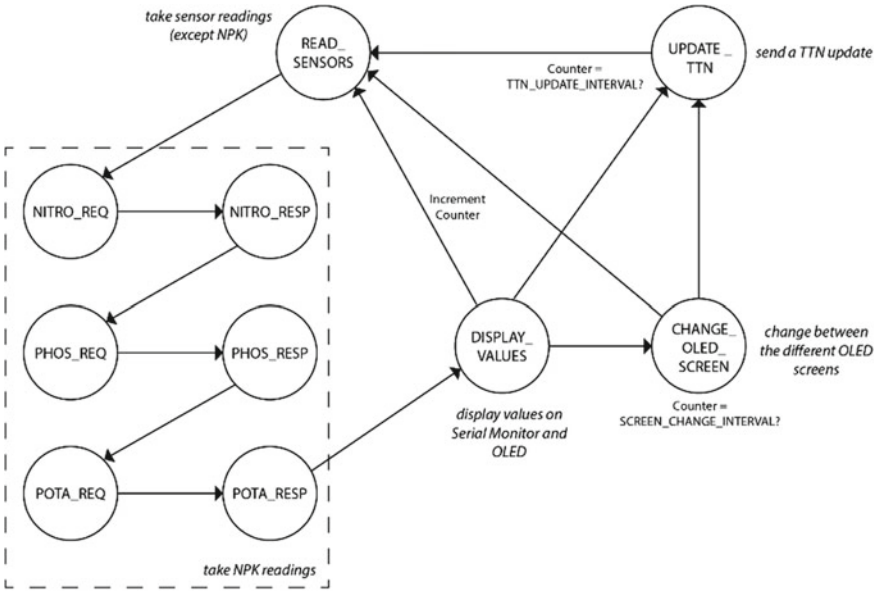


Fig. 12.15 The final screenshot of the illustrated state machine

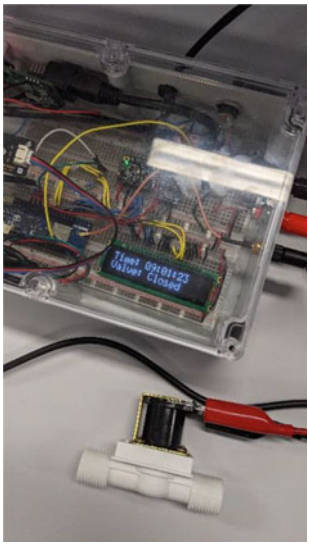


Fig. 12.16 The controlling the valve using RTC and TTN downlinks

```
uint16_t PrecAg::getMoisture() { int sum=0;
for (int i=0; i < FILTER_TERMS; i++) sum += _moisture[i];
return (sum/FILTER_TERMS);
}
```

12.2.12 Data Analysis

It was highly desirable to go beyond the simple visualisation capabilities of Thing-Speak and Tago.IO, and to extend them with further data analytics and machine learn-ing. Some of the possibilities considered were to:

1. Develop a machine learning algorithm that predicts crop yield or crop health based on our sensor measurements.
2. Perform alerting based on high or low watermarks (such as sending an email based on low moisture, or chemical levels that are too high); or
3. Predict an irrigation schedule based on soil moisture and potentially other data, with automation of valve activation to reduce the human effort in-volved with crop irrigation.

The challenges that needed to be overcome in implementing the above are listed in Table 12.5.

A starting point considering, we selected a linear regression analysis based on the rate of decrease of soil moisture: we hypothesised that this might be used to predict an irrigation schedule. This also aligns well with our decision to include a solenoid valve that could be turned on and off by our node.

The use of the five-value averaging at our node for soil moisture (see the previous section "Edge Computing") did not sufficiently remove the large variations in readings taken at that sensor. We implemented an Analysis routine within ThingSpeak to compute hourly averages every fifteen minutes (the code for this route is included in the Appendix). Around a day's worth of soil moisture averages from ThingSpeak was imported into MATLAB. We were then able to calculate linear regression coefficients for this data that we used to predict the soil moisture content at a particular point in the future. This analysis is shown in the Fig. 12.17.

Once the coefficients were calculated, it was again trivial to create a dashboard widget within Tago.IO that uses them to predicts the time at which a particular moisture level will be reached. This value could be used to calculate an irrigation schedule to turn on the valve. Figure 12.18 below shows the dashboard widget and the formula used to calculate the valve turn-on time.

Table 12.5 Issues to be overcome with machine learning and analytics

Issue	Description
Data quality	Inexpensive sensors without the appropriate signal conditioning (or sensors not installed properly in the soil) cause inaccurate readings or unexpected variations in sensor data. These tend to hide underlying patterns that we would like machine learning algorithms to detect
Quantity of data available for ML	Ideally, we would have a large data set with seasonal data across many different soil samples to populate our machine learning models
Difficulties implementing supervised learning	The data set above incorporates training data that labels data with a "success" outcome for the crop (positive or negative)
Lack of advanced sensors	Literature reviewed in [3] indicates that crop yield and health is often predicted with more advanced sensors than we had available (for example, using satellite imagery, UAVs, robotics, computer vision)
Simple alerting is trivial	Simple alerts are provided out-of-the-box with ThingSpeak and do not demonstrate more interesting and powerful capabilities of our node

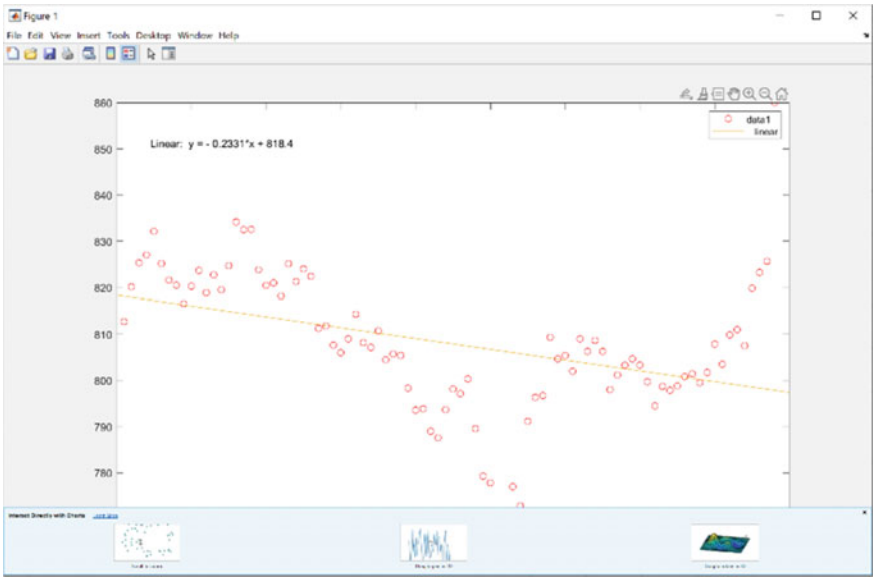


Fig. 12.17 The final screenshot of the linear regression ML: calculating constants for change in soil moisture using MATLAB

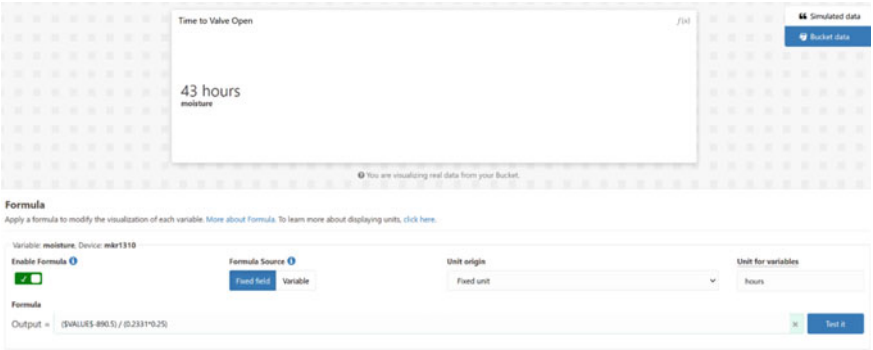


Fig. 12.18 The final screenshot of the creating a dashboard widget for valve opening time in Tago.IO

12.2.13 Challenges and Advantages

The Challenges we faced in the project are listed as follows:

1. GPS Conflicts
   The GPS MKR Shield supports connectivity using either UART or I2C protocols. UART is not a candidate for our node as the MKR WAN 1310 only has a single UART, and that UART is required for the NPK Sensor. However, when we integrated the GPS sensor using the I2C protocol, we noted that readings from the soil moisture sensor (also connected via I2C) became unreliable.
   Our troubleshooting indicated that there was no conflict caused by duplicate addresses on the I2C bus. We believe the issue may be an incompatibility between the libraries used by the GPS shield and the ones used by the soil sensor. Due to the fact the GPS is not considered mandatory for our node, we decided to de-prioritise in favour of data analysis activities.
2. SAMD Timer Interrupts Incompatible (Resolved)
   Noted in [7], "inside the attached (timer interrupt) function, delay() won't work, and the value returned by millis() will not increment. Serial data received while in the function may be lost." We experienced each of these issues when communicating with our NPK sensor (it returned occasional incorrect readings), which meant that the SAMD Timer Interrupt library was not feasible for our application. We instead implemented a state machine within the sketch providing precise co-ordination of communications with the NPK (and other) sensors, and successfully worked around this issue.
3. Generation of Downlink Traffic from Tago.IO Service (Workaround)
   A key reason for choosing the Tago.IO integration was its ability to send "downlink" messages from its dashboards. However, despite substantial effort, we have not been able to get this functionality to work correctly. We are instead sending these messages from the TTNv2 console rather than Tago.IO.

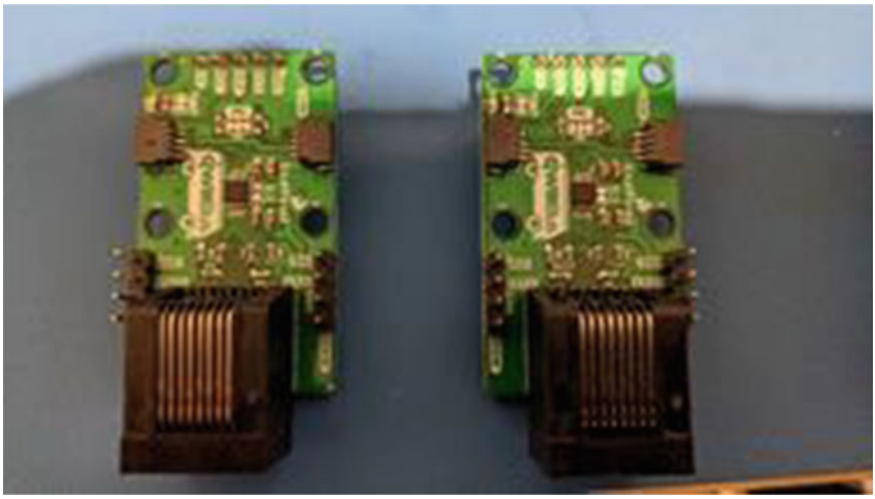


Fig. 12.19 The image of SparkFun QwiccBus EndPoint I2C repeaters

The Advantages of our Solution are as follows:

1. Machine Learning to Reduce Human Effort
   Precision Agriculture aims to introduce efficiencies into agricultural operations. We have demonstrated the concept whereby a solenoid valve for irrigation is controlled by a machine learning algorithm. If the input data and models are of high quality, the automation can help minimise the human labour associated with crop management.
2. Enclosure with External Connections
   It is not practical to implement a Wireless Sensor Node which requires the microcontroller to be located alongside the sensors in the soil. Our node design includes an enclosure and external components (solenoid valve, NPK sensor and soil sensor). The external sensors, power supply and USB port are interfaced via appropriate connectors on the enclosure. In a practical scenario, this would allow a faulty sensor to be easily replaced without needing to open the enclosure. To allow our I2C-connected Soil Sensor to be located up to 30 m from our node, we utilised a pair of Spark Fun QwiccBus EndPoint repeaters. This product shown in Fig. 12.19 delivers both power and the differential I2C signals to the Soil Sensor over standard Cat-6 UTP network cable.
3. Bi-Directional LoRaWAN Communication
   In some agricultural locations, our node may not have the option for mobile backhaul, or potentially even power, so LoRaWAN long range and low power characteristics make it a good choice of communications protocol [1]. The other advantage we gain from LoRaWAN is inherent security.
   The protocol features built in to LoRaWAN include AES-128 encryption to protect the data being transmitted, as well as activation processes that prevent

unauthenticated nodes from accessing the application. These features are considered critical for Precision Agriculture applications, as any accidental or malicious data modification might lead to "imprecise" outcomes, which may be avoided if we wish to maintain the security of a community's production of food.

4. Extensibility and Customisation
   Our node interfaces to sensors via inherently extensible protocols such as RS-485, I2C and SPI. This provides a lot of flexibility for adding new sensors or changing sensor types in the future. Potentially a range of nodes could be developed, some with more or fewer sensors, to reflect the needs of specific agricultural operations.

12.2.14 Future Scope

1. Energy harvesting—Solar Power
   In the current design, the Arduino and sensors are powered via USB connection to a PC and the valve is powered by a 240Vac to 12Vdc power adaptor. This is fine for a prototype, but it is impractical to run cables to deliver power if the node be deployed in the field. The use of solar panels to power the Node will eliminate the need for cables and allow the device to operate in a truly wireless manner.
   To implement solar power, the node would also require the addition of a battery and a buck or boost converter (depending on the output voltage of the panel). We would also have to take extra steps to ensure that power usage is as low as possible; using a non-Arduino based microcontroller would allow us to eliminate some of the issues caused with the Arduino libraries, allowing us to properly utilise interrupts so that the microcontroller can sleep until a reading or transmission is required. The OLED screen would also have to be off until required, so the addition of a button to turn the backlight on when required would be a simple addition to reduce power consumption.
2. Building an Agricultural Data Asset
   Considering a Precision Agriculture application will make decisions based on sensor measurements, each sensor is recommended to be of high quality and ensure that it is "only be sensitive to the parameter it measures and not be influenced by any other parameters in the surroundings" [9]. We did not have much confidence in the quality of the sensors we used simply because they were so inexpensive, uncalibrated, and showed significant variability in readings when the quantity measured had not obviously changed. Higher quality sensors with better accuracy could be purchased and tested to ensure that they are meeting the criteria above (for example, the electrical measurements of the NPK sensor could be compared to a chemical analysis of the same sample to determine accuracy).

Once the appropriate sensors have been identified, we could more confidently use to them to build a large "asset" of seasonal data samples across different locations. If subsequent machine learning is performed on the data, we would be much more confident that any results would be correct due to our higher confidence in the quality of the underlying data.

3. LTE or 5G Communication
   In the current design iteration, the Node is reliant on LoRaWAN for communication with the cloud servers. For use in remote locations where mobile backhaul is available, communication over a broadband cellular network may be preferable as the existing infrastructure could be used without requiring extra gateways to be installed.
4. Deploying a Microcontroller on a PCB
   The current prototype contains a breadboard and as such is not suitable for implementation outside of the lab. Any future prototype design may be considered for deployment on a PCB. This would also allow an alternate microcontroller to be chosen that is optimised for the peripherals that are relevant to our scenario. A non-Arduino-based microcontroller would also allow us to address some of the issues encountered with Arduino libraries, and provide more control over low power modes that might allow a more energy-efficient implementation.

12.2.15 Conclusion

The results achieved in implementing a wireless node for Precision Agriculture have been very pleasing. Within a few weeks, a node has been constructed comprising several sensors relevant to Precision Agriculture. The node can be extended by adding more sensors, and the microcontroller and protocols selected support many analogue and digital interfacing options.

The choice of LoRaWAN as communications protocol supports long range, low pow-er and security. It has also facilitated the rapid integration of the node with cloud services and has allowed us to demonstrate both visualisation and machine learning within the cloud.

Challenges needed to be overcome in the interfacing of the sensors and the development of the code. Many issues have been resolved, and pleasingly our node has been shown to operate over extended periods and to reliably send data into the cloud services.

This report has identified several possibilities for further development of the node. If high quality sensors are used, it would be possible for future prototypes to collect a large amount of accurate data across different locations that we could use to drive machine learning algorithms. Energy harvesting would be an important area of development, as many farms might be located away from sources of mains power.

Finally, an end-to-end solution incorporating wireless nodes as well as services in the cloud might form the basis of a SaaS offering (Software-as-a-Service) for Precision Agriculture. The data collected across many farms would grow over time, and if it were properly labelled to support machine learning, would be an asset for any entrepreneurs interested in offering this type of service to their agricultural customers.

12.3 Project 2: VISION—A Guide for the Visually Impaired

12.3.1 Abstract

The gift of sight is a vital component of a human being and is considered to be the most important out of the five senses. It helps us to navigate through our daily lives, keeps us away from danger, and observe our surroundings. It also allows us to read to gain information and learn new things which are important in our development as a person.

Unfortunately, some people are deprived of vision due to accidents or abnormalities. Slowly, as we grow older, some people tend to develop visual impairments over time which hinders their capabilities. People with problems with vision tend to limit their activities and eventually not realize their full potential.

12.3.2 Introduction

The Raspberry Pi has been used numerous times as a platform for creating prototypes for assisting the blind. The authors [12] developed the project "SIGHT: For the Blind" which uses the camera, RPi, and Google Android Things to give a qualitative description of the scenery in front of the camera. The authors in [13] developed an audio guidance system using ultrasonic sensors which provide audio cues to a person to help with navigation. The authors in [14] have shown the capabilities of RPi to perform Optical Character Recognition in real-time.

The Project Vision aims to help people who have problems with their eyesight. This proof of concept design as seen in Fig. 12.20, it features optical character recognition to read texts, and scenery description, to provide a qualitative description of the image taken by the camera and processes by using computer vision and neural networks. The microphone lets the user interact with an AI assistant to ask any questions, quick internet search for information, or entertainment. It also has Global Positioning System (GPS) that collects the position coordinates for the device which is sent to ThingSpeak and mapped the coordinates in the cloud. The Inertial Measurement Unit (IMU) sensor collects accelerometer

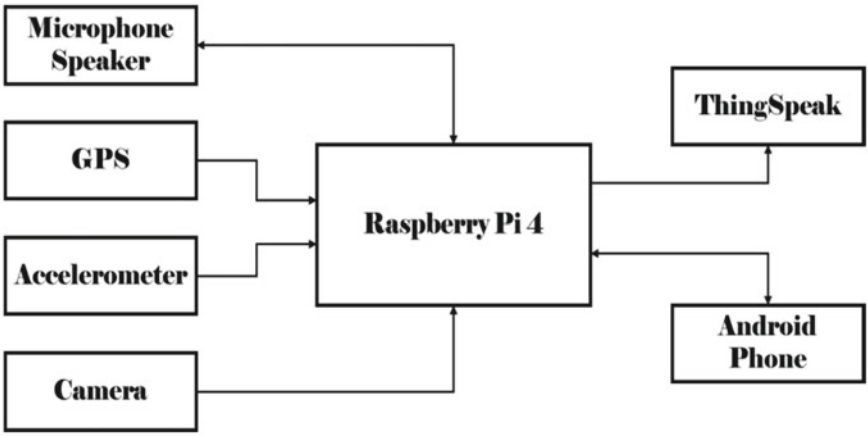


Fig. 12.20 The system block diagram for VISION

and gyroscope readings, which are also sent to the cloud which can be used to analyse the orientation and stabilization of the device.

12.3.3 Block Diagram

The system comprises four sensors, namely the microphone, GPS, Accelerometer, and Camera. All components of the system are connected to the Raspberry Pi 4, which serves as the brain of the entire system.

The Microphone/Speaker sensor uses Inter-IC Sound (I2S) for connecting digital audio devices. It primarily collects sound signals from the user, and also produces information in the form of sound waves. It uses the SPI protocol for communication to RPi.

The GPS sensor has Global Navigation Satellite System (GNSS) and General Packet Radio Service (GPRS) capabilities which can provide location information by connecting to GPS satellites and 2G/3G cellular communication. It is connected to RPi via Serial Interface.

The Accelerometer provides the force/acceleration data across three axes. It also uses the SPI protocol.

The Camera provides visual information to RPi. It uses the I2C protocol for communication. All sensor data is processed inside the RPi 4, and selected, processed information will be uploaded to the ThingSpeak Network for data storage and visualization. It will be uploaded via WiFi. Figure 12.20 shows the System Block Diagram for VISION.

An Android Phone is also be used for additional control over the entire system. It will receive data from the Raspberry Pi 4 and also sends control signals to trigger functions. It will also be connected to the RPi via WiFi.

12.3.4 Initial Sensor Interface and Sending Data to Cloud

The task accomplished for the week is installing the required libraries and testing the sensors acquired. Adafruit Voice Bonnet is shown in Fig. 12.21.

The Adafruit Voice Bonnet seen in Fig. 12.21 features two microphones, two 1 W speakers, and a 3.5 mm stereo jack as seen connected in Fig. 12.22.

12.3.5 Software Requirements

To use the voice bonnet, two software were installed; the Blinka, Voice card. The Blinka is CircuitPython Library that runs can run on CPython and Python. The voice card ensures audio compatibility with the RPi.

The terminal commands for installing the software were given in the appendix. A more detailed installation configuration is referenced in the references.

Upon installation, the alsamixer terminal command checks whether the RPi detects the voice bonnet. It also provides volume control as seen in Fig. 12.23.

The card name for the voice bonnet is seeed-2mic-voicecard, and the volume for the speakers and headphones can be adjusted.

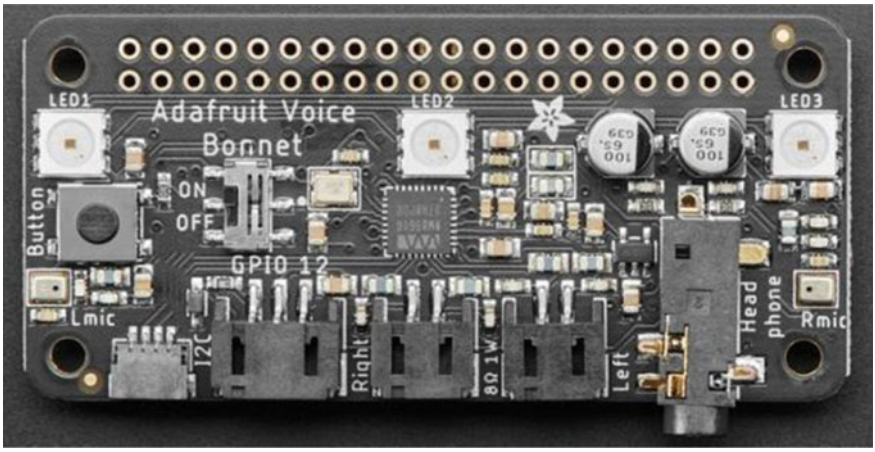


Fig. 12.21 The image of adafruit voice bonnet

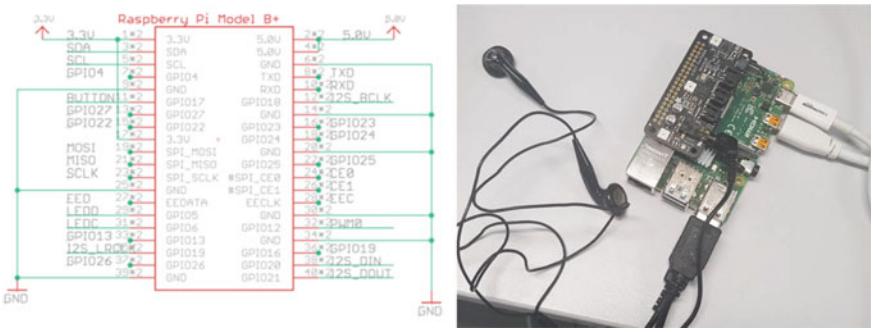


Fig. 12.22 The final screenshot of the (left to right) schematic and actual set-up of voice bonnet and RPi

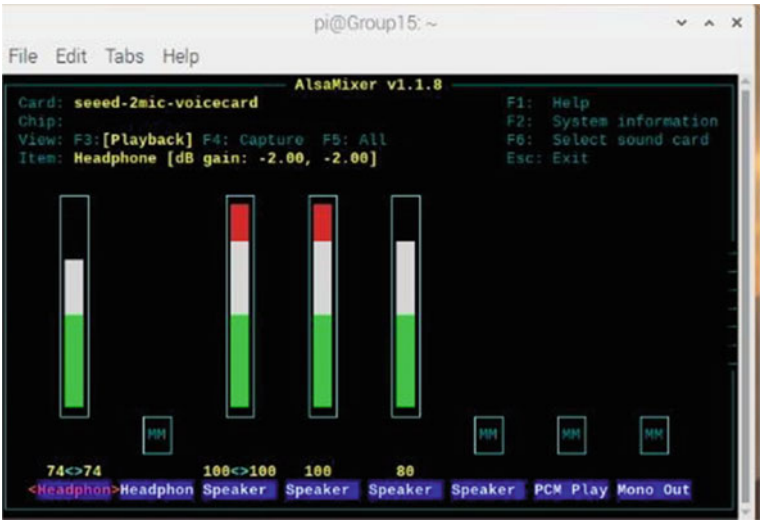


Fig. 12.23 The final screenshot of the Alsamixer output

12.3.6 Headphone and Microphone Test

After successful installation of software requirements, we test its functionality. To test the headphone, we invoke speaker-test -c2 in the terminal as seen in. Figure 12.24. The Adafruit Voice Bonnet seen in Fig. 12.21 features two microphones, two 1 W speakers, and a 3.5 mm stereo jack as seen connected in Fig. 12.24.

We can hear alternating sounds on the left and right sides of the headphones when running. To test the microphone, we invoke sudo a record -f cd -Dhw:2 | aplay -Dhw:2. In this command, Dhw:2 is used because the sound number for the voice bonnet is 2 as seen in Fig. 12.25.

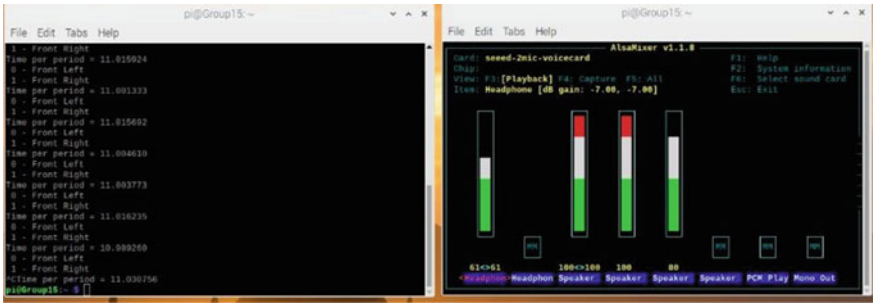


Fig. 12.24 The final screenshot of the testing the headphones

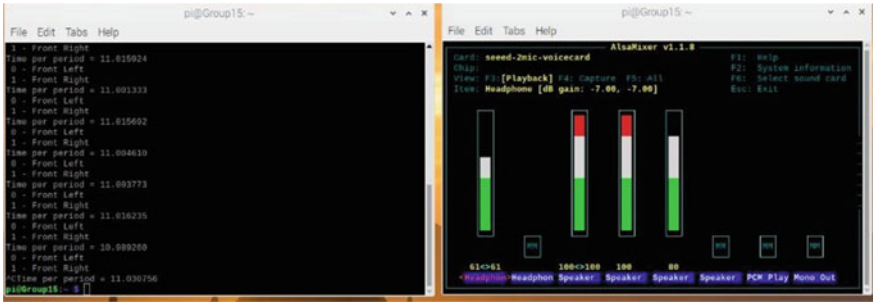


Fig. 12.25 The final screenshot of the testing the microphone

The command records the sound from the surroundings and outputs it to the headphones.

12.3.7 SIM7000E NB-IOT HAT

The GPS Sensor that will be used is the SIM7000E NB-IOT HAT. It provides low power and low cost at wide coverage.

It has a sim card slot, which is compatible with normal SIM cards and NB-IoT specific cards. It can be connected via the GPIO pins or the serial interface.

First, we tested the sensor by connecting it to the USB port. We have to enable Serial communication by invoking sudo raspi-config and choosing Interfacing Options → no → yes.

We installed two packages for testing the AT commands, the minicom, and the scratch. Results are shown below in Figs. 12.26 and 12.27. Figure 12.26 shows The NB-IOT HAT interface with RPi connected.

The SIM7000E NB-IOT hat requires an NB-IOT sim card or a normal sim card to operate. We have tested the device using a normal sim card, however, we have

Fig. 12.26 The NB-IOT HAT interface with RPi

Fig. 12.27 The final screenshot of the Testing AT commands on scratch

failed to connect it to a cellular site. However, the purpose of this device is to get the current map coordinates. Two options can be used for this, which are the General Packet Radio Service (GPRS) and Global Navigation Satellite System (GNSS). GPRS requires mobile 2G or 3G cellular communication while GNSS does not require. To obtain the coordinates we opt to use the GNSS and attached the GPS antenna included in the module for better reception of the signal.

To properly access data from the sensor, it is connected to the USB. An issue usually arises when the RPI has communication problems with the selected serial slot because of the operating system blocking certain responses. To disable such limitation, we invoke socat—/dev/ttyUSB3 to bypass issues with the operating system. The GPS sensor is connected to USB3 in our case.

Information in the sensor is access through AT commands. So the acquisition of the sensor data is done using python with a combination of AT commands. The serial library is used for this.

The latitude and longitude coordinates are accessed using the AT+CGNSINF command, which acquires GNSS navigation information parsed from NMEA sentences. The response from the command is displayed in the following format:

```
+CGNSINF: <GNSS run status>,<Fix status>,
<UTC date & Time>,<Latitude>,<Longitude>, <MSL Altitude>,<Speed Over
Ground>,
<Course Over Ground>, <Fix Mode>,<Reserved1>,<HDOP>,<PDOP>,
<VDOP>,<Reserved2>,<GNSS Satellites in View>,<GNSS Satellites Used>,
<GLONASS Satellites
Used>,<Reserved3>,<C/N0 max>,<HPA>,<VPA>.
```

Since we are only interested in latitude and longitude, we manipulated the string to acquire specific values and sent it to ThingSpeak. The results of the Latitude and Longitude Coordinates Sent to ThingSpeak are shown in Fig. 12.28.

12.4 IMU

The 9-DOF IMU sensor combines the acceleration, gyroscope, and magnetometer in one combined sensor. In using the sensor, we used the mpu6050 library, a library designed for the MPU6050 sensor. We only used specific functions such as get_accel_data and get_gyro_data to obtain acceleration and gyroscope measurements. This data is sent to a ThingSpeak Channel. The results of the measurements and the ThingSpeak Visualization are shown in Fig. 12.29.

Figure 12.30 shows the Raspberry Pi camera connection.

To test the camera as seen in Fig. 12.30, we invoke the command raspistill-o Desktop/image.jpg in the terminal which allows it to capture an image and save it to a jpg file as seen in Fig. 12.31. The use of the camera in a python script requires the installation of the picamera package.

12.4.1 Circuit Diagram

Figure 12.31 shows the connection of the selected four sensors to the Raspberry 4. The IMU9 and the Voice Bonnet are connected to the GPIO pin of the Raspberry Pi, the RPi Camera is inserted into its built-in slot, and the GPS is inserted via the serial interface.

Fig. 12.28 The final screenshot of the Latitude and Longitude Coordinates Sent to ThingSpeak

The design for the Voice Bonnet sensor is to directly interface it on top of the GPIO pins of the RPi. Since we need access to other pins, we connected the only required pins for it to work with all its functionality. Since both the IMU9 sensor and the voice bonnet uses similar pins, a Raspberry Pi GPIO Breakout Lead and

Fig. 12.29 The final screenshot of the acceleration and gyroscope measurements sent to ThingSpeak

Fig. 12.30 The Raspberry Pi camera connection

Header for breadboard are attached for easier connection and prototyping. Both the IMU9 and Voice Bonnet use an I2C connection which uses the SDA and SCL pins. Connecting in parallel with the said pins is not an issue as the Raspberry Pi detects both sensors.

It is also noticed that when using the Voice Bonnet with GPIO Breakout Board that all power pins and ground pins will be connected entirely. Connecting only one 5 V, 3 V and ground make the functionality of the voice bonnet inconsistent, which occasionally fails to work. The schematic of the connection is shown below in Fig. 12.32.

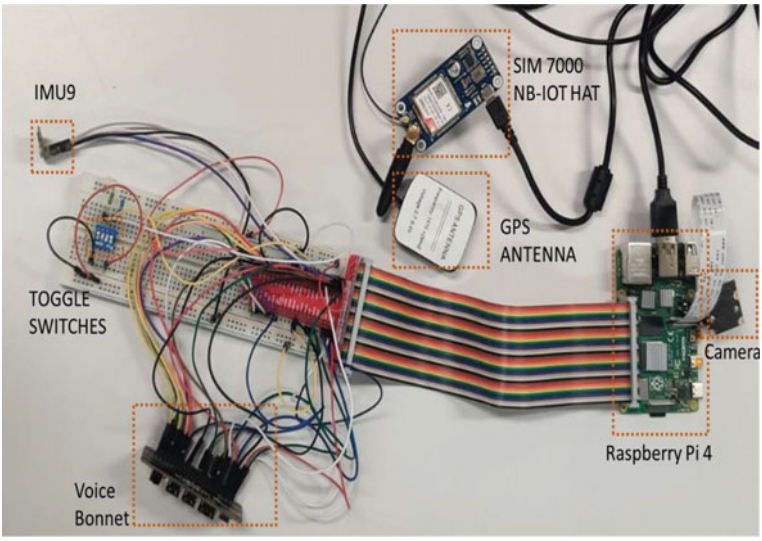


Fig. 12.31 The final screenshot of the assembled Circuit Diagram

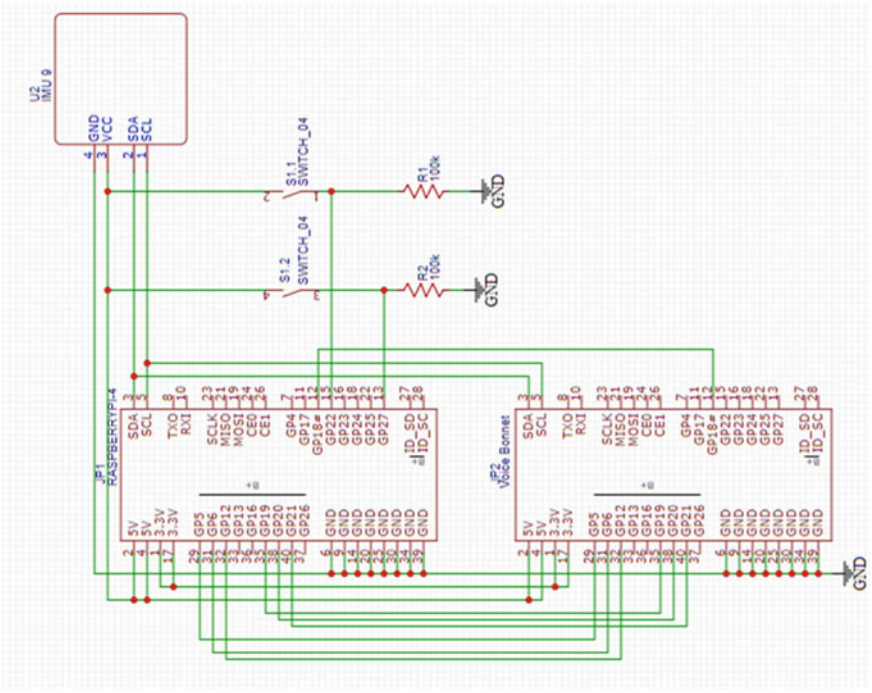


Fig. 12.32 The schematic diagram of the system

Pull-down resistors are connected to GPIO22 and 27 which are connected to a switch. The toggle switch is used to control whether the system is sending data to ThingSpeak.

12.4.2 System Algorithm

The implementation of the final system consists of running applications. Separates pieces of code are created for different features.

Scenery Description:

The Scenery Description generally takes a picture using a camera and the photo is described using texts and read the texts via a speaker. It is controlled via an android application shown in the next section. The simplified algorithm is shown below.

Algorithm: Input: Button, Accelerometer (from the android app)

Output: Text While True
If Button is pressed or Accelerometer is shaking Take a picture
Describe the picture into a string
Read the string output using the speaker
Optical Character Recognition:

Optical Character Recognition allows to extracts text from an image and reads the texts via the speaker. It was controlled by C and S on the keyboard.

Algorithm: Input: C, S (keyboard shortcut)

Output: Text
While True
Wait for key to be pressed If C is pressed
Take a picture
Extract texts from the picture
Read the texts and output using the speaker Else if S is pressed
Exit infinite loop
AI Assistant

The AI assistant allows the user to talk to an installed Google Assistant which can reply with an appropriate response using sound. This feature is generally controlled by a push-button built-in with the Voice bonnet and requires an internet connection.

Algorithm: AI Assistant.
Input: Push Button Output: Speaker response While True.
Wait button to be pressed If button is pressed.
Record message.
Print the recorded message.
Play assistant response into the speaker.
GPS and IMU measurements.

The system can send GPS coordinates, accelerometer, and gyroscope measurements to the cloud. The information is controlled via toggle switches connected to GPIO 22 and 27.

Algorithm: AI Assistant.
Input: GPIO 22, GPIO 27.
Output: latitude, longitude, accelerometer data, gyroscope data While True.
Read latitude and longitude from GPS sensor.
Read accelerometer and gyroscope measurement from IMU If GPIO 22 is high.
Send latitude and longitude to ThingSpeak If GPIO 27 is high.
Send accelerometer and gyroscope data to ThingSpeak Wait for 20 s.

12.4.3 Mobile App

Using MIT App Inventor, an android application was created to communicate with Raspberry Pi via WiFi. The Raspberry Pi hosts a web server that contains the visual description of an image taken from a camera. The Android App extracts the strings or texts in the webserver and using the Text-to-Speech function, it reads the text out loud. The default language for the description in English, but the application allows the text to translate into Polish, Dutch, Spanish or German.

This application can be controlled by tapping the designed buttons or by shaking the device, as it is also programmed to trigger responses using the accelerometer.

The GUI has three buttons that have the following functions:

- Get Data – When this button is pressed, the Android application asks the raspberry to take a photo and create a description of that photo. This description is sent back to the phone which prints in the space below and the phone reads aloud the text received.
- Select Language—Allows the user to translate the text receive to a specified language.
- Translate—This button translates the English text received to the chosen language and prints the translation to space below. This also reads the translated text aloud.

The GUI and the code block design are shown below in Figs. 12.33 and 12.34 respectively.

12.4.4 Applied Machine Learning

The Scenery description, Optical Character Recognition, and AI Assistant use Neural Networks for implementation. In this section, we break down how the neural networks were used to perform the specific tasks.

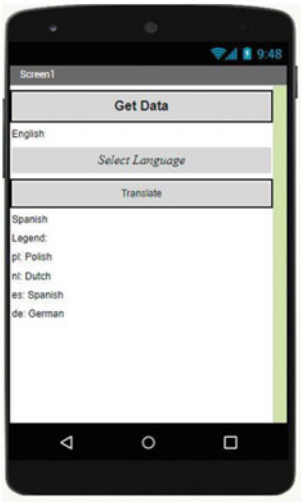


Fig. 12.33 The final screenshot of the Android App graphic user interface

Scenery Description:

To perform the complicated operation of scenery description, two types of neural networks were used, namely Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). Multiple past cases have shown that CNNs are mostly used for processing image inputs while RNNs are used for Natural Language processing.

When an image is captured by a camera, the trained CNN model extracts visual features. These visual cues serve as the base word for a sentence, which will then be an input to the RNN model to generate the captions filling up additional words to create a sentence. The technique and model are derived from the paper of [15] and the base Raspberry Pi implementation is based on the implementation of [16].

Optical Character Recognition:

Optical Character Recognition was accomplished by using computer vision and Recurrent Neural Networks. The OpenCV platform for Raspberry Pi allows performing computer vision tasks as shown in Fig. 12.35. Tesseract is an open-source text recognition engine that can be used directly to print text from images. Tesseract also uses a combination of CNN and RNN to perform these tasks [17]. The base codes for implementation are from [18].

AI Assistant:

The AI assistant is powered by using a Google Assistant. This feature starts by recording a message. Then using Long Short-Term Memory (LSTM), it performs speech recognition to try to predict and get the right words to make a clear message from the recording. After getting the appropriate message, the system sends the data to the server obtaining the keywords from the message and find the appropriate response from the inquiry. The final response is sent back to the Raspberry Pi. Implementation of an AI Assistant is based on [19] and the results are shown below in Fig. 12.36.

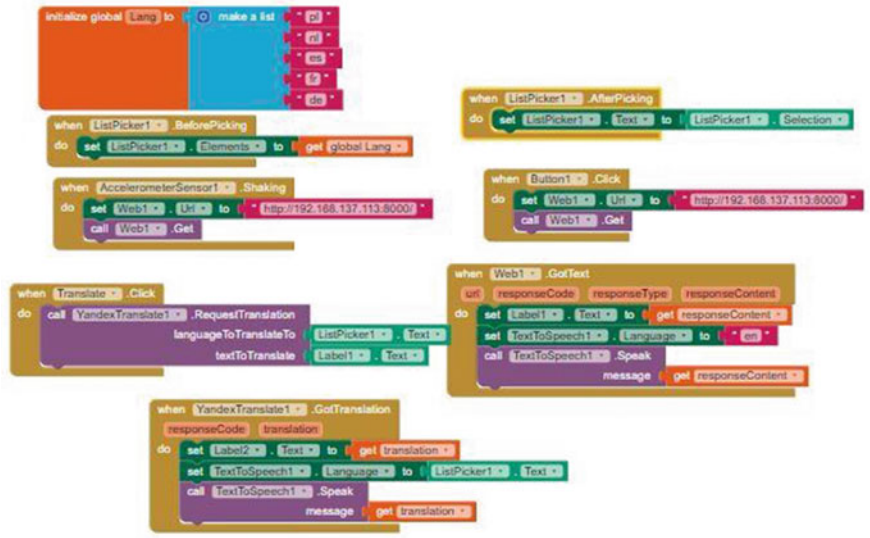


Fig. 12.34 The final screenshot of the Android App code blocks

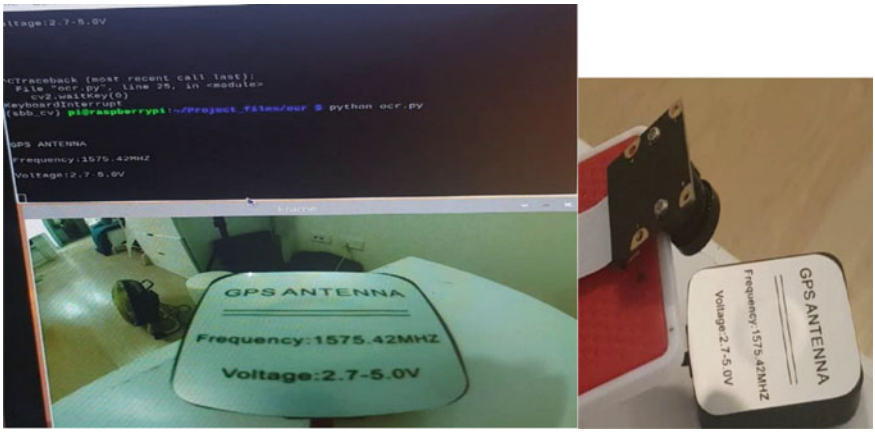


Fig. 12.35 The screenshot of the optical character recognition results

12.4.5 Data Visualisation

The Latitude and Longitude coordinates gathered from the GPS are best viewed with a map. A MATLAB code was created to plot the GPS coordinates in the ThingSpeak Channel. For testing purposes, the code created was used to extract the last recorded map coordinates at a specific date and plotted in a map as shown in Fig. 12.37.

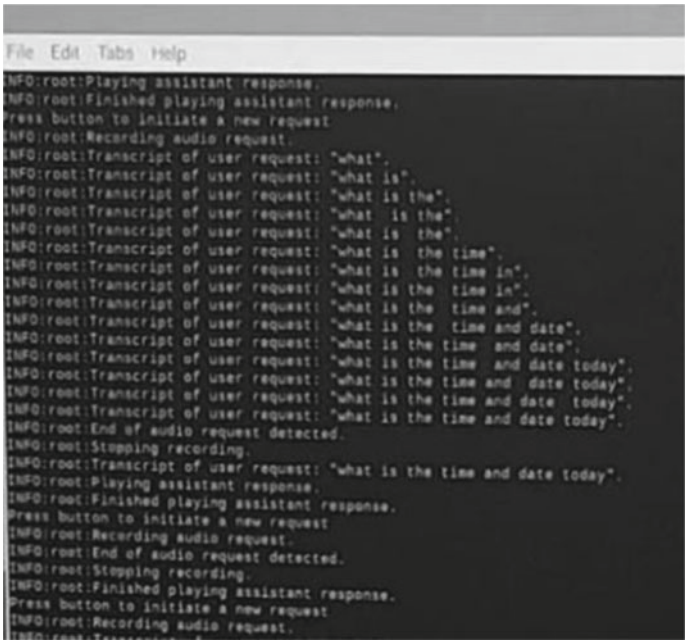


Fig. 12.36 The screenshot of the results from AI Assistant requests

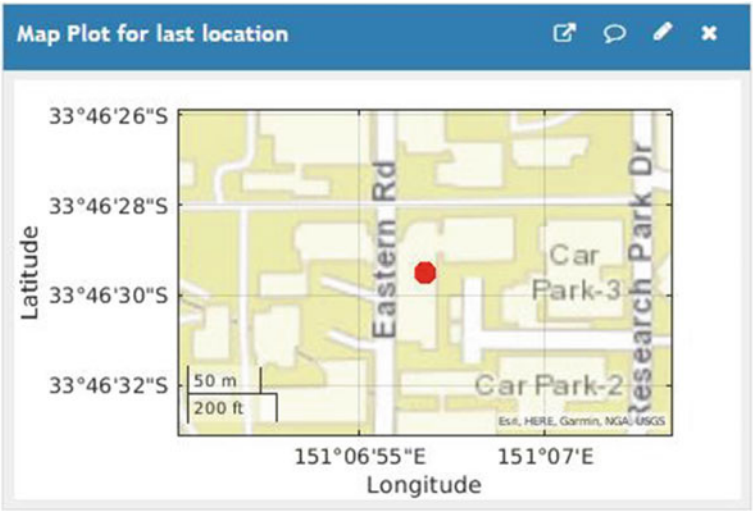


Fig. 12.37 The screenshot of the ThingSpeak Plot for GPS coordinates

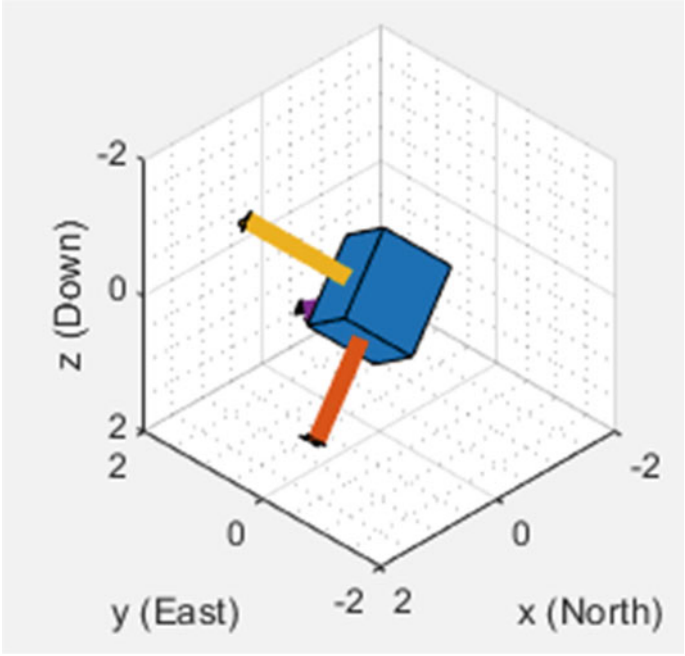


Fig. 12.38 The screenshot of the orientation estimate using Kalman Filters

The acceleration and gyroscope data can be used to estimate the orientation of a device using sensor fusion techniques. A Kalman Filter was applied to combine these two readings to get a better approximation of the orientation. A 3D plot was generated to visualize the orientation of the device as seen in Fig. 12.38.

12.4.6 Challenges

The GPS module interfacing did not happen as planned. After downloading and using the minicom monitor, the device did not respond correctly. The issue was solved by looking up SIM700E troubleshooting documentation. A test code was used to test the module's connectivity.

The GPS module requires a SIM card that has CAT M1/NB-IoT for GPRS functionalities. The SIM tray in the GPS module is not a nano sim card and makes it difficult to get a hold of a proper SIM. The team finalized a source to get a SIM card but later decided to use the GNSS module of the device for GPS functions.

The voice bonnet has a hat that blocks all GPIO ports which prohibits attaching additional components to the GPIO pins. Getting rid of the hat is not possible and the team discussed getting GPIO pin extenders. Alternatively, we sourced only the necessary GPIO pins to a breadboard and were successful in doing speaker and mic tests.

The AT commands to get GPS information went unresponsive after preliminary tests. The code was written was able to extract AT-OK information but was unable to get the AT+ GNSINF latitude and longitude information.

Integration of Google Assistant was not smooth. Installation of few libraries gave errors. The implemented assistant would not completely recognize speech if spoken softly. The use of the voice bonnet for recording using the microphone and playing output sound using a speaker made the program stop. Changing the output sound interface to either the HDMI or the built-in 3.5 mm jack of RPi 4 solved this issue.

The implementation of character recognition using the Pi camera did not perform properly. The camera needs ideal conditions, which means that the lighting may be optimal, the distance from the text, the size and font, and the alignment of the text all matter. The camera was able to recognize and recite under ideal conditions but requires appropriate image filters at different changing environmental conditions.

12.4.7 Conclusion

The proof of concept design for Vision was implemented in this project. Features such as scenery description, optical character recognition, and AI assistant have huge potential in assisting people with problems with their eyesight in different areas in their lives. The scenery description can give visual cues on the events in the surroundings. The optical character recognition can help the person read signs or texts in front. The AI assistant can give fast information such as directions, locations, or fast research on basic topics. The GPS coordinates can be given to a guardian in cases we need to track the location of a person.

IMU measurements can be used later on for calibration and proper placement of the device to a person. Practical application of these features and the entire system is still in its development stage as the whole system is not optimized in hardware and software and there are a lot of bugs and issues that need to be solved. Power consumption, computational time and complexity, and other relevant parameters have not been studied in this project. The best implementation of neural networks and other algorithms have also not been explored. The design of the actual package has also not been created.

12.4.8 Future Scope

To create a system that can be used for practical application, here are some of the recommendations:

- The sensors used are too bulky for use. The SIM7000E NB-IOT Hat does have a long and relatively large antenna for GPS. It is recommended to re-place it with

smaller modules. The voice bonnet has been replaced with a microphone and a push button which are only essential for the foreseen applications.
- The use of the android application is not ideal for the system. Pushbuttons or toggle switches are good selection for controlling the device as they can provide tactile feedback.
- The creation of the casing for the entire system may be considered. The camera may be placed at normal eye level to provide a good angle for taking images. A headgear with earphones or a headset would provide the best comfort and practicality for the user.
- Sensor fusion of GPS and IMU data is recommended to provide better tracking, localization, and orientation of the device which can eventually help with navigation problems
- Connecting to a 4G or 5G network would avoid the necessity to connect to a WiFi network that is always not available. Cellular communication provides longer coverage than WiFi.

12.5 Satellite Tracker

12.5.1 Abstract

The project Group has chosen is a Satellite Tracker, which will follow satellites visible in the sky, such as the moon and the International Space Station (ISS) and take pictures of them. This project will take data in from several sensors to establish the geographical location of the tracker and its 3D orientation. This is to ensure that the tracker is pointing in the correct position in real-time. The Satellite Tracker periodically publishes data to ThingSpeak, a cloud-based data analysis platform.

12.5.2 Introduction

The Satellite Tracker is pictured below in Fig. 12.39. The students of Internet of Things Systems Designs, or ENGG4201, have been assigned the task of developing an “Internet of Things” project in the first term of 2021 Session. This project needs to consist of an IoT node connected to the cloud.

12.5.3 Block Diagram

The Satellite Tracker consists of many components, including a Raspberry Pi 4, GPS sensor, IMU sensor, Pi Camera, 2 Servo Motors, a 2-axis Joystick, and an

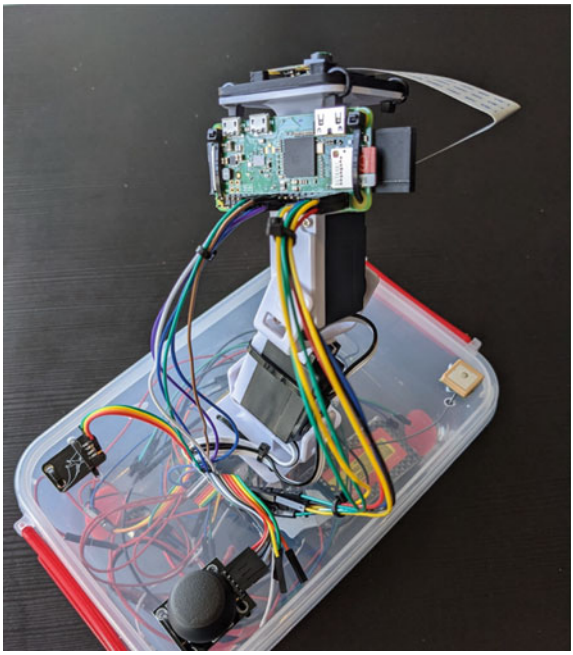


Fig. 12.39 The final Mechanical Design and assembled model

Analog-Digital Converter (ADC). The IMU consists of a 3-Degree of freedom (DoF) gyroscope, 3-DOF Magnetometer, and a 3-DoF accelerometer, along with a barometer. A diagram of the actions can be seen in Fig. 12.40.

12.5.4 Sensor Interface

- GPS
  The GPS receiver allows the Satellite Tracker to know its exact geographical location which is required to be able to point in the correct direction. In order to implement the GPS, serial communication was utilised on the Raspberry Pi. Reading the serial port obtained a GPS string. Once a string is read from the serial port, it may be interpreted. They are in an NMEA (National Marine Electronics Association) protocol with a GPRMC (Time, date, position, course, and speed data) format [20]. We reference a library [21] to scale the serial data accordingly and convert it to the proper latitudinal and longitudinal values. Initially, the program was run, and it displayed latitude and longitude of –33.774947, 151.115576 respectively, which shows up building E6A on Macquarie University's campus, as seen in Fig. 12.41 [22]. The program was also run at home by one of the group members, and the numbers also verified the correct location (Table 12.6).

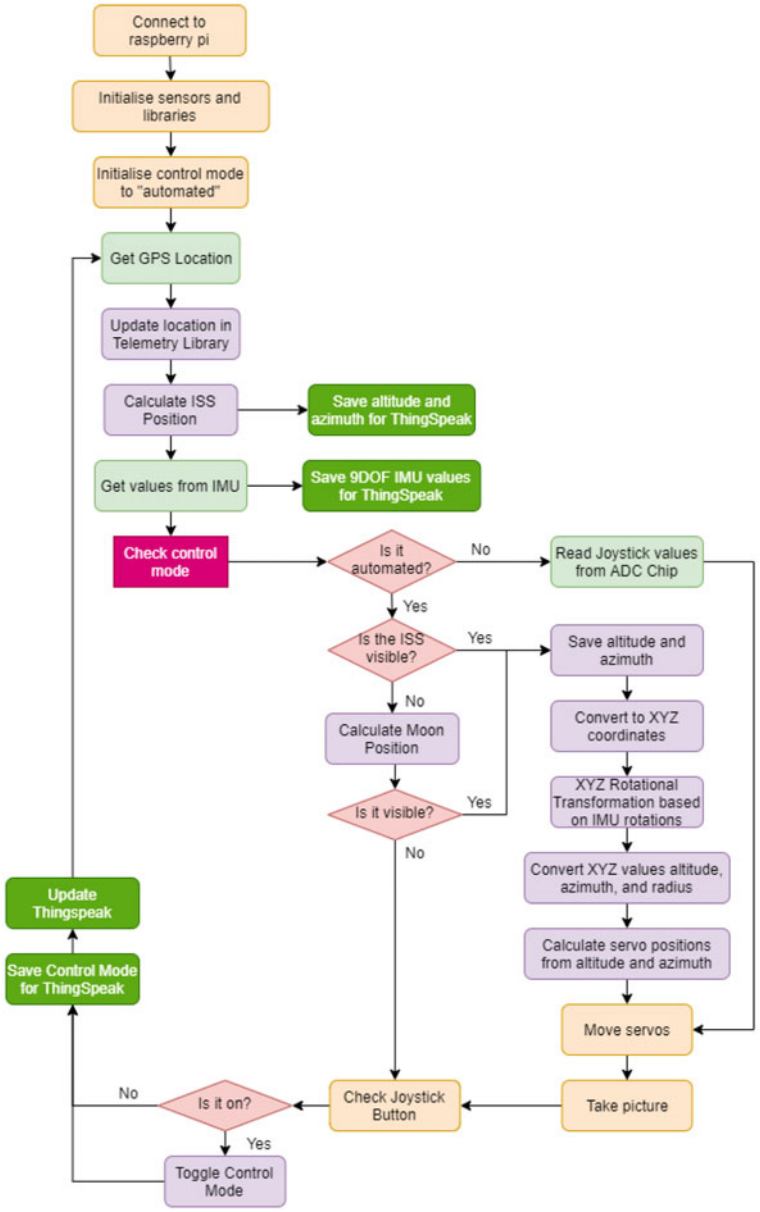


Fig. 12.40 The final screenshot of the Program flow chart

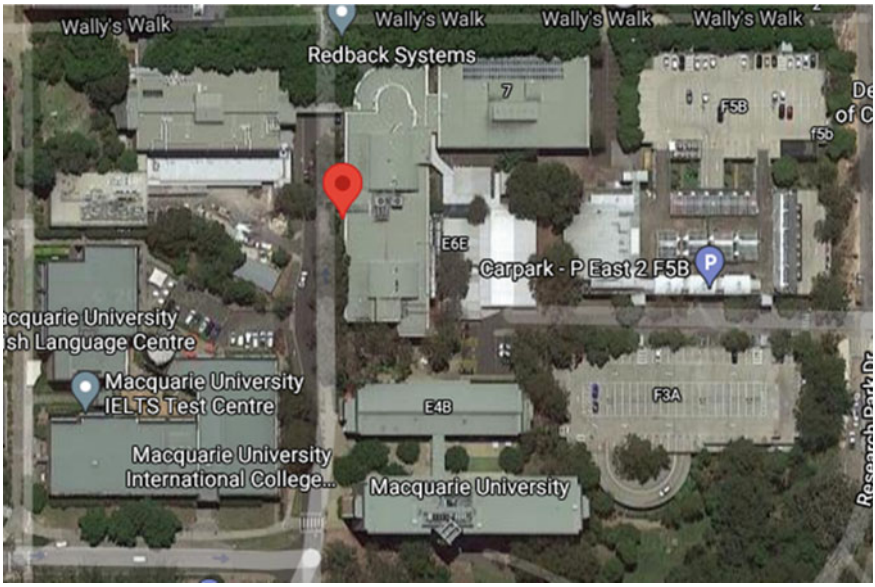


Fig. 12.41 The final screenshot of the GPS coordinates [22]

Table 12.6 Sensors used by the Satellite tracker

Sensor	Specification	Purpose (Output Data)
GPS	MOA171101018	Exact geographical location (Latitude, Longitude) Time (DDMMYY HH:MM:SS)
ADC	MCP3008	Digital value from joystick (Integer)
IMU—Magnetometer	QMC5883L	Bearing (Degrees)
IMU—Accelerometer	ADXL345	Roll and Pitch (Degrees)

The console output of the GPS can be seen in Fig. 12.42.

- Joystick and ADC
  A joystick allows the user to manually operate the servos. The user presses the joystick button to change from automatic mode to manual mode. Then when the user moves the joystick, the servos move correspondingly. In order to implement code for the joystick, code for the analog-to-digital converter may be made. This chip, the MCP3008, uses SPI for communication, with a baud rate of 3 Mbps. After be-ginning this SPI communication, the bits may then be read. After reading the bits, some logical operations may be applied in order to extract the data. These logical operations were modelled 'o' the raspberry Pi Forums [23]. Once the library for the ADC was built, the two joystick axes were wired. The code was tested and appeared to be working. Figure 12.43 shows the joystick being moved around in a somewhat sporadic circle.

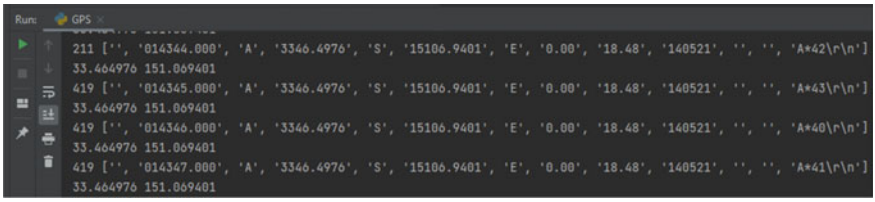


Fig. 12.42 The final screenshot of the GPS console output

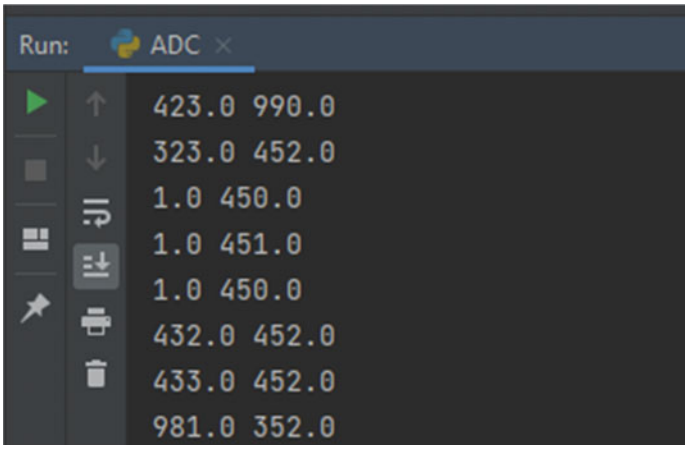


Fig. 12.43 The final screenshot of the ADC joystick values

- IMU

The inertial measurement unit (IMU) contains multiple sensors, of which we use the accelerometer and magnetometer. The IMU allows the Satellite Tracker to know its 3D orientation (roll, pitch, and bearing). These measurements are critical to be able to point in the correct direction. The IMU is accessed via I2C (Inter-Integrated Circuit) and libraries made by the sensor manufacturers.

Placement of the IMU was important for two reasons. First, it is very sensitive to magnetic fields, so it needs to be placed as far away from the servos as possible; and, secondly, the orientation needed to be confirmed and compensated for in the main program. We used an external compass in this calibration process so that once it was mounted, we knew what orientation the IMU was in relation to the Satellite Tracker.

12.5.5 Custom Library

The code was implemented in python with class structures, so making them into a library was as simple as dropping the corresponding python files into the working

directory and importing them. Libraries were written for the ADC chip, the GPS sensor, the IMU, the TLE manager, servos, and for the vision processing component. These libraries all consisted of some initialisations, which would require information such as pin numbers. After initialisations, the objects could then be modified or read, such as setting the servo positions or getting the IMU heading. This allowed for easy individualised testing of sensors, as they could be accessed without the need of most of the other code. This also made integrating these sensors and components very easy.

12.5.6 Circuit Diagram

The components in the project may all be wired together in order to work properly. A full circuit diagram of these can be seen in Fig. 12.44.

12.5.7 Transmission Protocol

We implemented WiFi using the onboard hardware of the Raspberry Pi. We used WiFi as we did not need to purchase any additional hardware and it satisfied our needs for bandwidth and range. We had also tested transmitting to ThingSpeak early into the project which was a success. Other communication protocols that were considered consisted of: LoRaWAN (Long Range Wide Area Network), Bluetooth and Bluetooth LE (Low Energy). Another advantage of using WiFi is that a VNC (Virtual Networking Computer) session can be established with the Raspberry Pi which requires a fast connection. A VNC session allows for remote access to the Raspberry Pi desktop enabling easy development.

12.5.8 Cloud Data Visualisation

The below image in Fig. 12.45 contains: Top Left: The 3D orientation of the Satellite Tracker. Top Right: The location of the Satellite Tracker (Zoomed out). Middle Left: The direction that the Satellite Track is pointing. Middle Right: The location of the Satellite Tracker (Zoomed in). Bottom Left: Automatic control enabled (Green light), manual control (no light). Bottom Right: The location of the International Space Station (Zoomed out).

The Satellite Tracker uploads the following data to ThingSpeak every 10 s: Roll, Pitch, Bearing, Tracker Latitude, Tracker Longitude, Satellite Latitude, Satellite Longitude, Azimuth, Elevation and Automatic Control state. ThingSpeak has a limit of 8 fields, so we implemented a method to enable us to use additional fields.

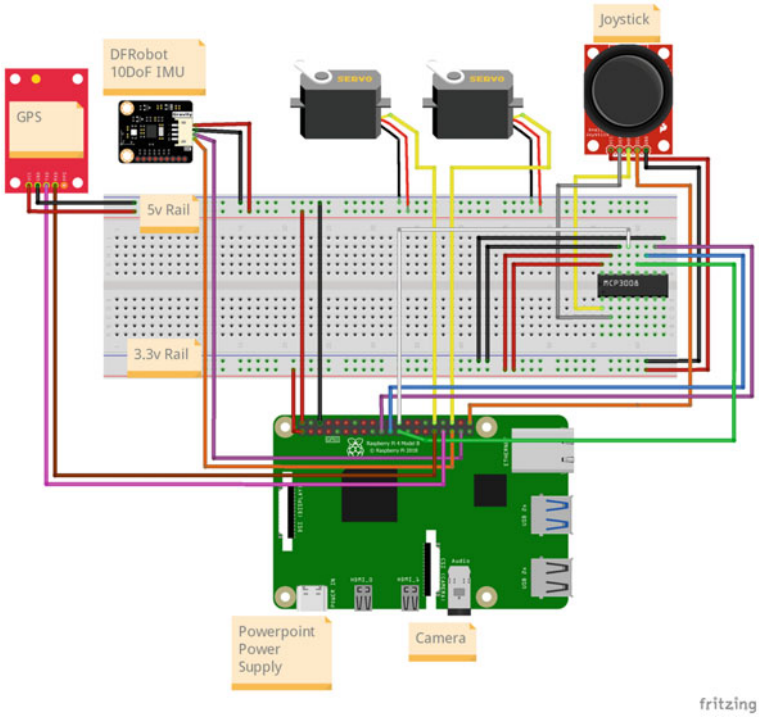


Fig. 12.44 The final circuit diagram constructed for the project

We send multiple data values in a comma delimited string, which MATLAB can split back into individual fields.

There are additional fields that contain the raw data as seen in Fig. 12.47, some of which are 'combined' values, shown below. These combined values need to be interpreted by MATLAB to obtain the full information encoded in them (Fig. 12.46).

12.5.9 Transmit Data to the Cloud

The Satellite Tracker publishes data to the ThingSpeak cloud platform via a commonly used Python library called 'urllib2'. This library contains functions and classes to PUT or GET information from a webpage through HTTP format. Each unique ThingSpeak Channel has a write API key and anyone with this key can publish data to the Channel.

To publish data our main program simply opens an HTTP connection to a URL such as.

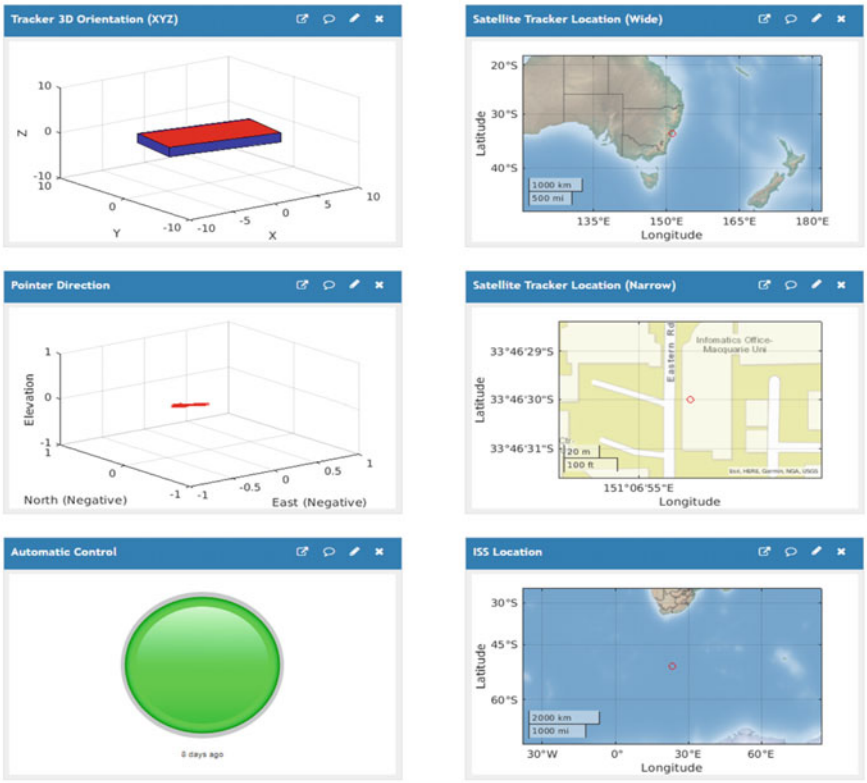


Fig. 12.45 The final screenshot of the Main ThingSpeak Visualisations

'https://api.thingspeak.com/update?apikey=123456&field1=field1-data&field2=field2data'

where 123,456 is our unique write API key, field1 data is the data for field 1 and field 2 data is the data for field 2, and so on. Below, Fig. 12.48 is a picture of the first attempt made to publish the roll, pitch, and bearing data to ThingSpeak which shows the data we are trying to send, and that ThingSpeak correctly displayed this data.

12.5.10 Code Flowchart

The main program was developed to integrate the different libraries for a complete system, to reflect the program flow chart from Fig. 12.49.

The program initializes the Raspberry Pi and sensors, then calculates the position of the ISS and moon to determine if either is above the horizon. If one is, it saves

Fig. 12.46 The final screenshot of the Main ThingSpeak Visualisations for the control of Satellite tracker

that azimuth and altitude then calculates an 'xyz' position relative to the device. After reading the IMU rotation from the accelerometer and magnetometer, the 'xyz' position of the ISS or The Satellite Tracker publishes data to the ThingSpeak cloud

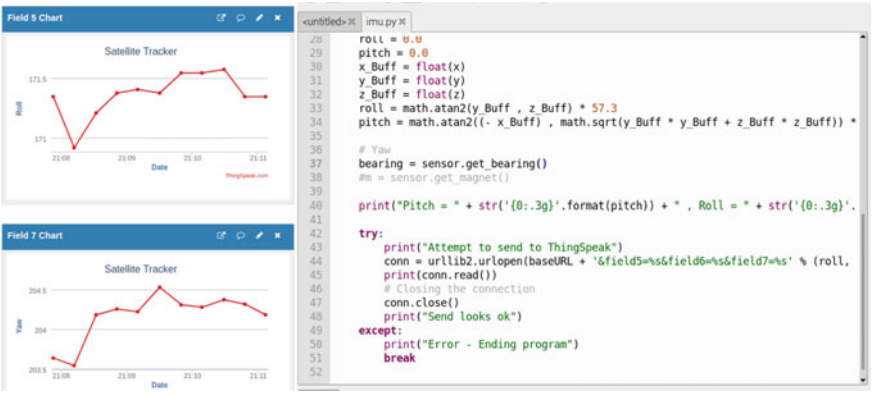


Fig. 12.47 The final screenshot of the Main ThingSpeak Visualisations of Satellite tracker and syntax code of transmission

platform via a moon are rotated around the three axes to determine their position relative to the rotation of the device. The 'xyz' positions are then converted back to elevation and azimuth relative to the device, then undergo inverse kinematics via DH transformations to convert the elevation and azimuth to servo positions.

While testing the program in the lab and attempting to point to the sun in the sky, the servos began following an unexpected path. After looking at the program output, it was noticed that the device was following the ISS. The group looked up the current position of the ISS, and it was in fact overhead, and the servos were pointing in the correct direction.

12.5.11 Edge Computing

The Satellite Tracker utilises edge computing by converting TLE data to Satellite position and Azimuth/Elevation values to Servo PWM signals. In this process, a python library called "skyfield" is used to handle telemetry data for satellites and celestial bodies. This allows for easy integration of finding elevation and azimuth values for various satellites and celestial bodies, including but not limited to the ISS and the moon. A library was implemented such that a user can input their GPS coordinates (later to be connected to the GPS sensor) and they can request the azimuth and elevation of the ISS, the moon, the sun, mars, and other satellites and celestial bodies. An output from this program can be seen below in Fig. 12.49.

12.5.12 Data Analysis & Computer Vision Feedback

In order to further improve the accuracy of locating orbiting objects, a computer-vision feedback system may be implemented. This would be done by using a vision

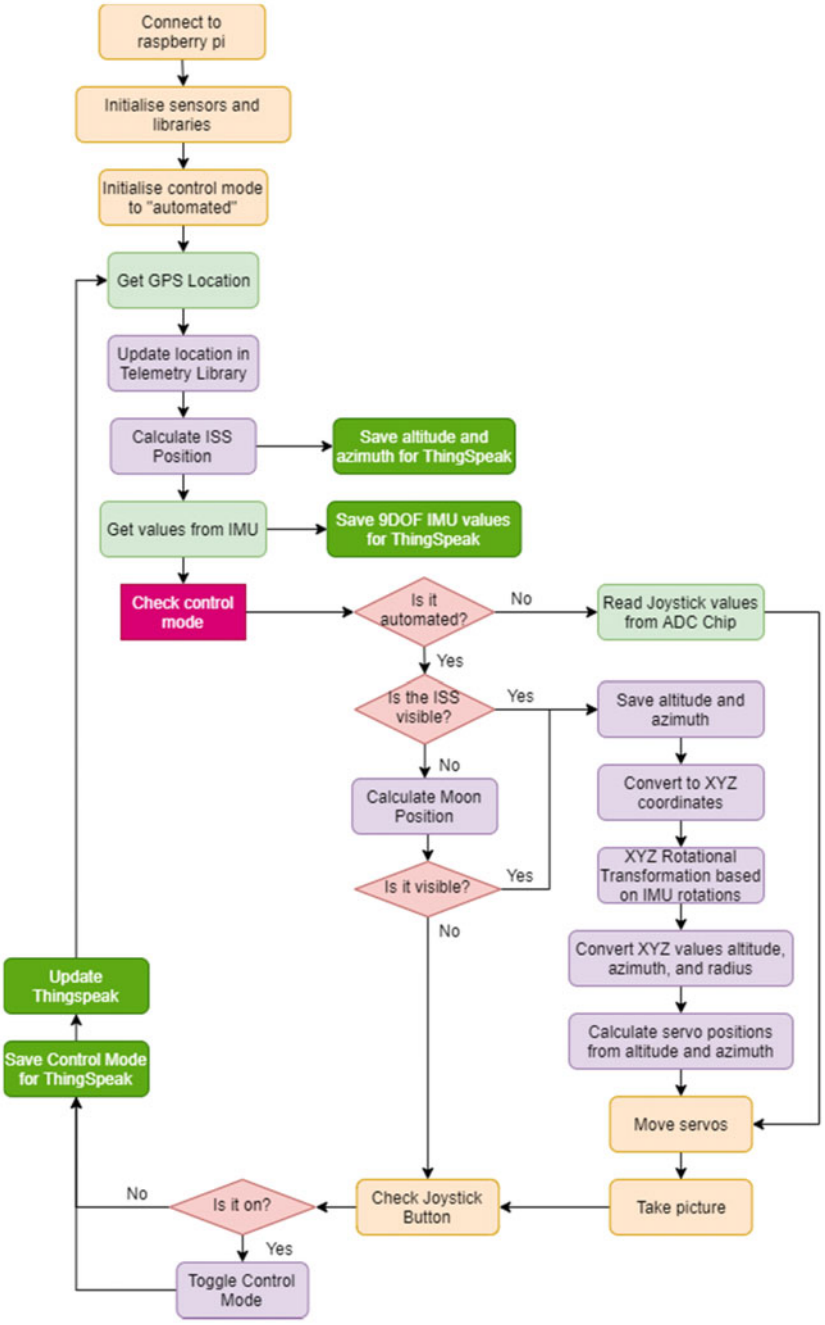


Fig. 12.48 The final screenshot of the Program flowchart

```
BODY    ELEVATION (DEG)      AZIMUTH (DEG)       DISTANCE (KM)
MOON: (-31.085975190774874, 171.58929129514505, 369451.4514404412)
ISS:  (-61.91160130862604, 143.45709499405464, 11740.376440927377)
SUN:  (16.97370587000944, 311.9328724041702, 151622118.58168736)
MARS: (33.14971309799016, 357.2821151047832, 334000300.7335619)
```

Fig. 12.49 The final screenshot of the body elevation, Azimuth, and distance

processing library, known as OpenCV, to analyse the image from the camera and determine how far off the system is from pointing directly at the moon or artificial satellite.

To do this, the system converts the camera into a grayscale image applying a threshold to the image to grab only the pixels which are greater than some brightness.

An algorithm is then applied to find the contours of the threshold image, which may outline the contours on the image. The largest contour which is found is determined to be the object to follow, and a bounding rectangle is drawn to determine the object's centre. An offset angle is then calculated based on the frame height and width and the camera's field of view. This angle offset is then returned and added to the servo positions.

It may be noted that this only works in a dark environment with minimal light pollution, so this is not applied when the sun is above the horizon. An image of the moon run through this vision processing function can be seen below in Fig. 12.50, with the moon's contour outlined in red, a bounding rectangle in green, the centre point located with a red dot, with the centre of frame designated with a white dot.

12.5.13 Challenges

- GPS Challenges
  This biggest issue was data reliability within the confines of the building. If the GPS sensor was in the middle of the lab, far from a window, it was difficult to get proper GPS readings; the sensor would mostly read 0, 0. Once the GPS receiver was moved near a window, however, the readings became accurate. To mitigate this, the Satellite Tracker will remember the last valid GPS location from the sensor and output that when a null reading appears.

The GPS receiver would not connect to any well-known and established Raspberry.

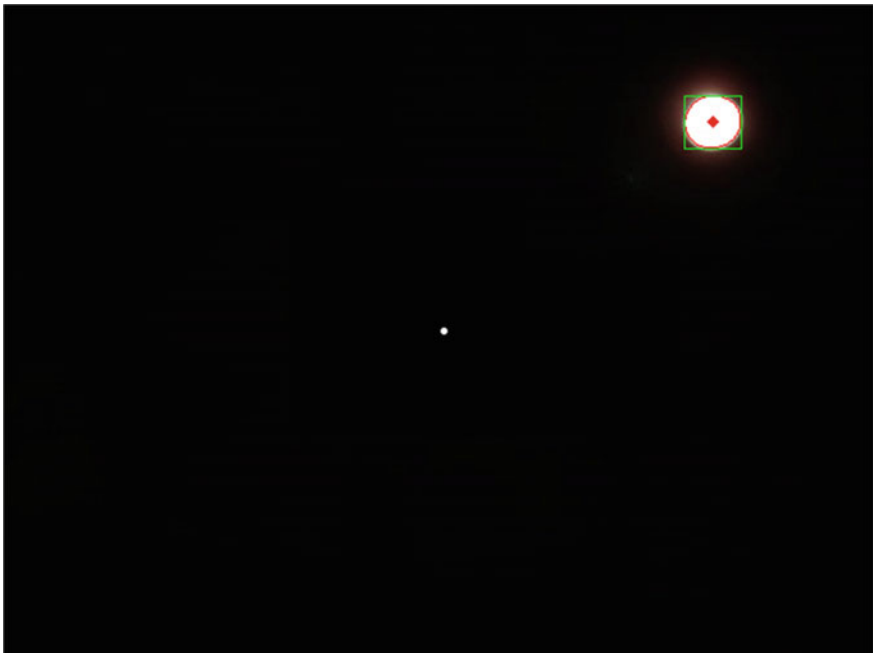


Fig. 12.50 The final screenshot of the moon computer vision

Pi programs such as '`cgps`' or '`gpsmon`'. This led us to spend a lot of time testing and troubleshooting as we thought there was a problem. We realised that the GPS unit was continuously printing data to the serial port, and we can simply read those values periodically. We also had to properly close the serial port connection when finished with it, as not implementing this caused reading errors during testing.

The GPS sensor outputs values in the format of LLDD.DDDD, where LL is the latitude/longitude values, and DD.DDDD are the degrees of the position. These need to be converted to decimal latitude/longitude values by taking the LL and adding DD.DDDD/60 to get the proper readings. Otherwise the GPS location is incorrect.

- Joystick and ADC Challenges
  There were not many challenges with the ADC or joysticks, as most of the code just worked upon implementation. The most challenging portion would have been sourcing the documentation for the MCP3008 chip, enabling proper communication, but that was quite simple as well.
- General Hardware Challenges
  The main considerations in the assembly were mounting the IMU as far away from the servos as possible. The magnets in the servos interfere with the sensitive IMU sensors, particularly the magnetometer which relies on reading the Earth's magnetic field. We also had to program servos limitations into the

Python code. The servos can reach far enough down so that they can hit the Raspberry Pi onto other objects.

Initially the Satellite Tracker was powered by a mains connection, however during testing it was identified that portability was important. Since there were many GPS challenges, access to a window was needed during testing. A small Lithium-Ion battery with a 5 V regulator and charger was implemented. This allows portability of the Satellite Tracker, and a means to recharge the battery. Other methods of energy har-vesting such as Photovoltaic panels were considered, but not chosen due to their added complexity, size and cost.

- Data Visualisations Challenges
  The challenge with these data visualisations is that a plot of an object's 3D orientation is commonly done in real-time. There will be some time delay between moving the Satellite Tracker, and that current orientation being updated in ThingSpeak. Also, as MATLAB is being used, it requires some time and practise to learn the code strength- s/weaknesses and syntax. We have learnt that a weakness of the MATLAB visualisation in ThingSpeak is that they do not auto-update like the data fields. A manual refresh of the webpage is needed to update the MATLAB visualisations.
- IMU Challenges
  It seemed the accelerometer readings were very consistent and reliable, but the magnetometer and gyroscopic sensors were imperfect. The magnetometer seemed to give poor readings when close to the servos, as the magnetic field of the servos overpowers the Earth's magnetic field. This can be fixed by moving the IMU further from the servos. The gyro seemed stable for the most part, but if any readings were missed, or the sensor was not updated quickly enough, it seemed to quickly drift away from the correct angles. With that, a combination of the accelerometer and magnetometer may be sufficient for determining the rotation of the device.
- TLE Challenges
  A Two Line Element (TLE) contains an object's orbit, motion and time data [5] and look like this in Fig. 12.51.

We began monitoring the website publishing the TLE data but noticed it did not update very often, sometimes up to 6 h later. Considering the International Space Station orbits the Earth approximately every 90 min, this created a very large problem for us to create a 'real time' tracker which points in the direction of a satellite. It took a lot of searching before we realised that we needed a different solution. With more research, we discovered that we did not need the TLE data to be refreshed within a very short period. A Python package called 'sgp4',

```
ISS (ZARYA)
1 25544U 98067A   21164.04547793  .00000758  00000-0  21937-4 0  9999
2 25544  51.6446   3.2116 0003497  90.7552  45.6437 15.48977591287958
```

Fig. 12.51 The ISS TLE data

a Simplified General Perturbations mathematical model, can predict the position of the satellite from a TLE set. The newer the TLE the better, as the error in the prediction model increases over time. However, this allowed us to download a new TLE every few hours when published by NORAD or NASA and still be able to point in real-time at the satellite we are tracking.

12.5.14 Advantages

The Satellite Tracker runs primarily from TLE sets obtained from the internet, so we can program it to track anything there is data published for. This includes all the planets in the Solar System, satellites and debris. For example, the 23-ton booster from a Chinese Long March 5B rocket fell back to Earth on May 9th 2021 over the Middle East [6]. The Satellite Tracker would have been able to track it as the data was being published on websites such as 'celestrak'.

This IoT system could become the basis for other space related projects. A ground station with an antenna uses many of the same positioning and tracking capabilities shown in the Satellite Tracker. With a few hardware/software additions, the Satellite Tracker could track a satellite by its signal, allowing for direct communication.

12.5.15 Conclusion

Future improvements include implementing the 'TalkBack' feature in ThingSpeak so that the user can select an object to track, and the Satellite Tracker will auto-update. Presently, the object being tracked is hard coded into the main program. Another feature that would improve the project would be to implement an automatic refreshing of the MATLAB visualizations.

References

1. G. Sen Gupta, S.C. Mukhopadhyay, in *Embedded Microcontroller Interfacing: Designing Integrated Projects*, vol. 65. Lecture Notes in Electrical Engineering (Springer, 2010). ISSN 1876-1100, ISBN 978-3-642-13635-1
2. S.C. Mukhopadhyay, *Intelligent Sensing, Instrumentation and Measurements, Smart Sensors, Measurement and Instrumentation*, vol. 5 (2013). ISBN: 978-3-642-37026-7 (Print) 978-3-642-37027-4 (Online)
3. H.A. Jawad, et al., Energy-efficient wireless sensor networks for precision agriculture: a review. Sensors (Basel, Switzerland), **17**(8), 1781 (MDPI AG, 2017). https://doi.org/10.3390/s17081781

4. H. Navarro-Hellín, et al., in *A Wireless Sensors Architecture for Efficient Irrigation Water Management*, vol. 151, pp. 64–74. Agricultural Water Management (Elsevier B.V, 2015). https://doi.org/10.1016/j.agwat.2014.10.022
5. A. Sharma, et al., Machine learning applications for precision agriculture: a comprehensive review. IEEE Access. **9**, 4843–73 (2021). https://doi.org/10.1109/ACCESS.2020.3048415
6. Github. "Adafruit BME280 Library". Github. https://github.com/adafruit/Adafruit_BME280_Library (accessed 12 June 2021)
7. Github, Adafruit Seesaw Library. Github. https://github.com/adafruit/Adafruit_Seesaw (accessed 12 June 2021)
8. Newhaven Display, Slim OLED Code (Arduino). Newhaven Display. https://www.newhavendisplay.com/resources_dataFiles/excode/txt/Arduino/slim_oled_code.txt (accessed 22 May 2021)
9. K. Hoang, SAMD_TimerInterrupt Library. Github. https://github.com/khoih-prog/SAMD_TimerInterrupt (accessed 22 May 2021)
10. C. Docs, Cayenne Low Power Payload. MyDevices. https://developers.mydevices.com/cayenne/docs/lora/#lora-cayenne-low-power-payload (accessed 29 May 2021)
11. S.C. Mukhopadhyay, S. Chandra, *Intelligent Sensing, Instrumentation and Measurements* (Springer, 2013)
12. SIM7000E NB-IOT HAT, Waveshare, 2021. https://www.waveshare.com/w/upload/archive/7/76/20180703091809%21SIM7000E-NB-IoT-HAT-Manual-EN.pdf. Accessed 14 May 2021
13. K. Rembor, Adafruit Voice Bonnets? Adafruit, 2021. https://cdn-learn.adafruit.com/downloads/pdf/adafruit-voice-bonnet.pdf?timestamp=1604294889. Accessed 14 May 2021
14. S. Faris, Jr. Suhail, SIGHT: For the Blind hackster.io (2017). https://www.hackster.io/makergram/sight-for-the-blind-c1e1b9. Accessed 13 June 2021
15. N. Ma'muriyah, A. Yulianto, Lili. Design prototype of audio guidance system for blind by using raspberry Pi and fuzzy logic controller. J. Phys. Conf. Ser. **1230**(1), 12024 (2019)
16. G. Sagar, T. Shreekanth, Real time implementation of optical character recognition based TTS system using Raspberry Pi. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **7**(7), 149 (2017)
17. S. Tsutsui, D. Crandall, *Using Artificial Tokens to Control Languages for Multilingual Image Caption Generation*: n. pag. Print (2017)
18. Y.Ochi, Raspberry Pi: Real Time Image Captioning and Speech hackster.io (2018). https://www.hackster.io/yoshihiro-ochi/raspberry-pi-real-time-image-captioning-and-speech-e71a43. Accessed 13 June 2021
19. F. Zelic, A. Sable, A comprehensive guide to OCR with Tesseract, OpenCV and Python, Nanonets, 2021. https://nanonets.com/blog/ocr-with-tesseract/. Accessed 13 June 2021
20. M. Aqib, Optical Character Recognition Using Raspberry Pi With OpenCV and Tesseract, MAKERPRO, 2019. https://maker.pro/raspberry-pi/tutorial/optical-character-recognizer-using-raspberry-pi-with-opencv-and-tesseract/. Accessed 13 June 2021
21. M. LeBlanc-Williams, Using Google Assistant on the BrainCraft HAT or Voice Bonnet Adafruit, 2020. https://learn.adafruit.com/using-google-assistant-on-the-braincrafthat. Accessed 13 June 2021
22. GPS Sentences—NMEA Sentences—GPGGA GPGLL GPVTG GPRMC. https://www.rfwireless-world.com/Terminology/GPS-sentences-or-NMEA-sentences.html, accessed: 2021–06–12
23. "GrovePi. https://github.com/DexterInd/GrovePi/blob/master/Software/Python/grovegps/GroveGPS.py, accessed: 2021–05–14
24. GoogleMaps. https://www.google.com/maps/place/33%C2%B046'29.8%22S+151%C2%B006'56.1%22E/@-33.7753413,151.1147601,431m/data=!3m1!1e3!4m5!3m4!1s0x0:0x0!8m2!3d-33.774947!4d151.115576, accessed: 2021–05–14
25. MCP3008bitstream. https://www.raspberrypi.org/forums/viewtopic.php?t=199793, accessed: 2021–05–14
26. .NORAD Two-Line Element Set Format. https://celestrak.com/NORAD/documentation/tle-fmt.php, accessed: 2021–06–12

27. Huge Chinese rocket booster falls to Earth over Arabian Peninsula. https://www.space.com/chinese-rocket-booster-long-march-5b-space-junk-crash, accessed: 2021–06–12
28. G.M. Mendez, M.A.M. Yunus, S.C. Mukhopadhyay, in *A WiFi based Smart Wireless Sensor Network for Monitoring an Agricultural Environment*. Proceedings of IEEE I2MTC 2012 conference, IEEE Catalog number CFP12MT-CDR, May 13–16, 2012 (Graz, Austria), pp. 2640–2645. ISBN 978-1-4577-1771-0
29. N.K. Suryadevara, S.C. Mukhopadhyay, in *Smart Homes: Design, Implementation and Issues*, vol. 14. Smart Sensors, Meas. and Instrumentation (Springer, 2015). ISBN 978-3-319-13556-4
30. R. Yan, X. Chen, S.C. Mukhopadhyay, in *Smart Sensors, Measurement and Instrumentation*, vol. 26. Structural Health Monitoring: An Advanced Signal Processing Perspective (Springer, 2017). ISBN 978-3-319-56125-7
31. S.C. Mukhopadhyay, A. Mason, in *Smart Sensors, Measurement and Instrumentation*, vol. 4. Smart Sensors for Real-Time Water Quality Monitoring (Springer, 2013). ISBN 978-3-642-37005-2-1
32. S.C. Mukhopadhyay, in *Lecture Notes in Electrical Engineering* , vol. 96. New Developments in Sensing Technology for Structural Health Monitoring (Springer, 2011). ISBN 978-3-642-21098-3
33. H. Ghayvat, S. Mukhopadhyay, X. Gui, N. Suryadevara, WSN- and IOT-based smart homes and their extension to smart buildings. Sensors **15**, 10350–10379 (2015). http://www.mdpi.com/journal/sensors. https://doi.org/10.3390/s150510350
34. H. Ghayvat, S.C. Mukhopadhyay, X. Gui, Issues and mitigation of interference, attenuation and direction of arrival in IEEE 802.15.4/ZigBee to wireless sensors and networks based smart building. Measurement **86**, 209–226 (2016). https://doi.org/10.1016/j.measurement.2016.01.045
35. S.C. Mukhopadhyay, O. Postolache, in *Smart Sensors, Measurement and Instrumentation*, vol. 2. Pervasive and Mobile Sensing and Computing for Healthcare: Technological and Social Issues (Springer, 2012). ISBN 978-3-642-32537-3