IoT BASED STADIOMETER

A PROJECT REPORT

Submitted to

GONDWANA UNIVERSITY, GADCHIROLI

by

PRANAV KHATALE

ABHAY GAWALI

VENKATRAMANA KURA

SHRIKRUSHNA WAGH

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS & TELECOMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

GOVERNMENT COLLEGE OF ENGINEERING,

CHANDRAPUR-442403

2021-2022

IoT BASED STADIOMETER

A Project Report Submitted to

GONDWANA UNIVERSITY, GADCHIROLI

by

PRANAV KHATALE

ABHAY GAWALI

VENKATRAMANA KURA

SHRIKRUSHNA WAGH

under the guidance of

Prof. H. M. RAZA

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS & TELECOMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

GOVERNMENT COLLEGE OF ENGINEERING,

CHANDRAPUR-442403

2021-2022

CERTIFICATE

This is to certify that the project entitled “**IoT Based Stadiometer**” has been carried out by the team under my guidance in partial fulfillment of the degree of Bachelor of Engineering in Electronics & Telecommunication Engineering of Gondwana University, Gadchiroli during the academic year 2021-2022.

Team:

1. **Pranav Khatale**
2. **Abhay Gawali**
3. **Venkatramana Kura**
4. **Shrikrushna Wagh**

Date:

Place: Chandrapur

Guide	**Head of the Department**
(Prof. H. M. Raza)	(Prof. H. M. Raza)

Principal

(Dr. S. G. Akojwar)

GOVERNMENT COLLGE OF ENGINEERING,

CHANDRAPUR- 442 403

PROJECT APPROVAL SHEET

Following team has done the appropriate work related to the "**IoT Based Stadiometer**" in partial fulfillment for the award of Bachelor of Engineering in Electronics & Telecommunication Engineering of "Gondwana University, Gadchiroli" and is being submitted to the Government College of Engineering, Chandrapur.

Team:

1. **Pranav Khatale**
2. **Abhay Gawali**
3. **Venkatramana Kura**
4. **Shrikrushna Wagh**

Internal Examiner:

(Prof. H. M. Raza)

External Examiner:

(Prof. …………………………….)

Date:

Place: Chandrapur

GOVERNMENT COLLGE OF ENGINEERING,

CHANDRAPUR- 442 403

PAGE INDEX

TABLE INDEX

FIGURE INDEX

Abstract

A digital height-measurement device (IoT based Stadiometer) was developed to improve the speed and accuracy of the height measurement process and address the discomfort and inefficiency issues of traditional physical height-measurement devices currently used in hospitals and clinics. The digital stadiometer uses ultrasonic distance sensor to quickly and efficiently measure a patient's height. One key need for accurate height measurements from this device is to ensure the device is level to the ground. This project incorporated an ultrasonic distance sensor HC-SR04 along with a NodeMCU ESP8266 12-E Board, which allowed the user to 'self-adjust' the device and see the value of height on LCD screen. The device was prototyped and tested, then mounted onto a stadiometer stand for final testing and analysis. Limited testing proved that the device took quick height measurements, with height displayed on 16×2 LCD in about 5 seconds, was relatively accurate and further uploaded to IoT Cloud and data is collected in spreadsheet for more data analysis and storage, reducing paper work as well as human efforts. Future revisions to the prototype will strive to improve accuracy of the device through implementation of a temperature sensor and more thorough testing with improved algorithms.

Keywords: Height, Stadiometer, Ultrasonic Distance Sensor, NodeMCU ESP8266, IoT

Chapter 1

Introduction

Height measurement is an important metric for pediatricians and their patients because a person's height changes so rapidly throughout childhood and can indicate health concerns if height deviates from growth curves. The purpose of this project was to develop a digital version of the traditional physical height-measurement devices (stadiometers) currently used in hospitals and clinics. This device would have improved portability while striving to stay within the accuracy and speed of the current devices. This report was created to document the design process for creating this device from researching the current height-measuring process and brainstorming possible solutions to the final testing and analysis of the device. This introduction section of the following report consists of information on the traditional stadiometer, the need for accurate height measurement, the need for quick height measurement, the digital height-measurement device, and project goals.

1.1 Need for Improvement in Traditional Stadiometers

It is routine practice in pediatric hospitals and clinics to gather data on the patient's physical measurements before meeting with the doctor for any specific needs. One of these measurements is the height of the patient. A pediatrician will find the collection of height measurement data helpful so they can track the growth pattern of their patients. Most pediatricians will want to be able to have the most accurate data for their patients to ensure quality health care. Height is a critical indicator because it tracks how the patient is growing over time, comparing their growth to that of a healthy rate.

Traditional height measuring methods used in hospitals and clinics today consists of measuring the patient against a wall using a retractable ruler system called a stadiometer, shown in Figure 1. To measure the patient height accurately, the patient must stand up straight against the wall to be measured the nurse. The nurse then places the ruler on top of the patient's head and records the measurement reading marked on the ruler.

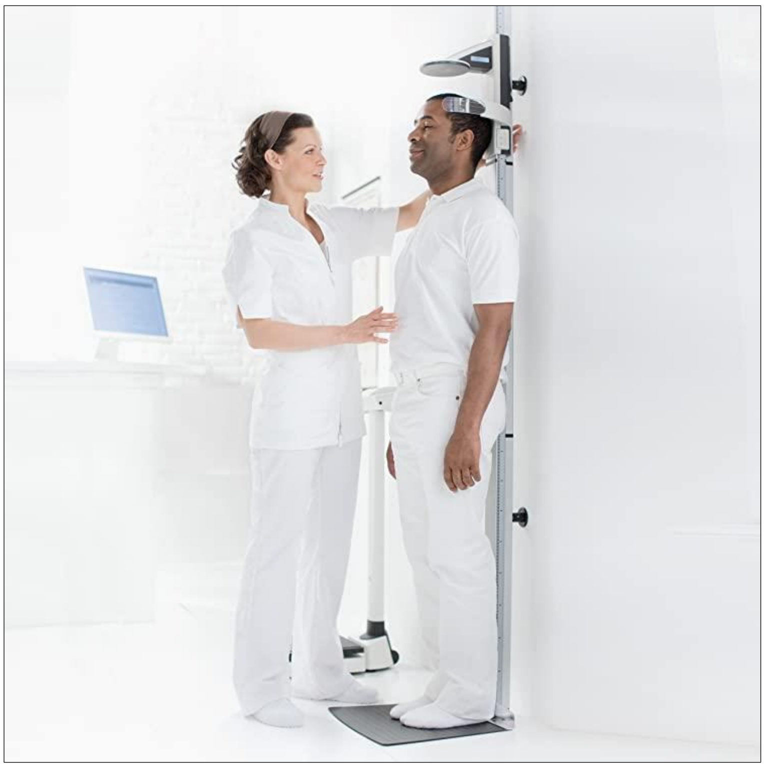


Figure 1.1: Traditional Stadiometer [1]

Height data is input manually by the nurse into the patient's file records in the medical database of the hospital or clinic. The height measurements are recorded at each visit, and over time, these data measurements create a trendline of the patient's growth pattern. The patient's growth pattern can be compared with the average growth patterns of patients with similar heights and weights to ensure that the patient is following the typical trend.

A stadiometer is a piece of medical equipment used for measuring human height. Height measurement is considered important when monitoring the health and growth of the human body. While standing erect, human height is measured as the distance from the bottom of the feet to the peak of the head. Stadiometer is used to collect accurate height data of a person and to help short nurses who could not accurately record height information for taller persons. Accurate height measurements help in the monitoring of height for any defect or factors that might affect the growth of an individual. Having a shorter height could be a very important indicator of affected growth. Regular accurate height measurements and follow-ups are necessary for the detection of abnormalities that may affect the normal growth rate in humans. For children, it is important to monitor that they are growing at a healthy rate, and for the elders, it is important to monitor whether or not their height decreasing. On average males are taller than females. The badly affected height in an individual's growth could be an indicator of osteoporosis. Height measurements coupled with weight measurements are used to calculate the Body Mass Index (BMI).

1.2 Features of IoT based Stadiometer Compared to Traditional Stadiometers

Manual height measuring devices (Traditional Stadiometers) give inaccurate readings, consume time for measurement, and constantly require expensive calibration services; therefore, the use of IoT based stadiometers turns out to be more reliable and enables paperless work. Time management is another problem since it takes a substantial amount of time to collect a lot of people’s height measurements. Stadiometers are used in routine medical examinations and also clinical tests and experiments in several places such as educational institutions, gymnasiums, medical diagnostics centers, screening centers, military training centers, etc.

Chapter 2

Problem Definition

Accuracy of height readings helps detect abnormalities in a child's growth patterns, which can be used to diagnose them for any diseases. For example, Figure 2 shows a chart featured in the article, “Clinical Dilemmas in Evaluating the Short Child” [2] by Melissa D. Garganta, MD, and Andrew A. Bremer, MD, Ph.D. shows the growth pattern of a male patient with acquired hypothyroidism as compared to the average male.

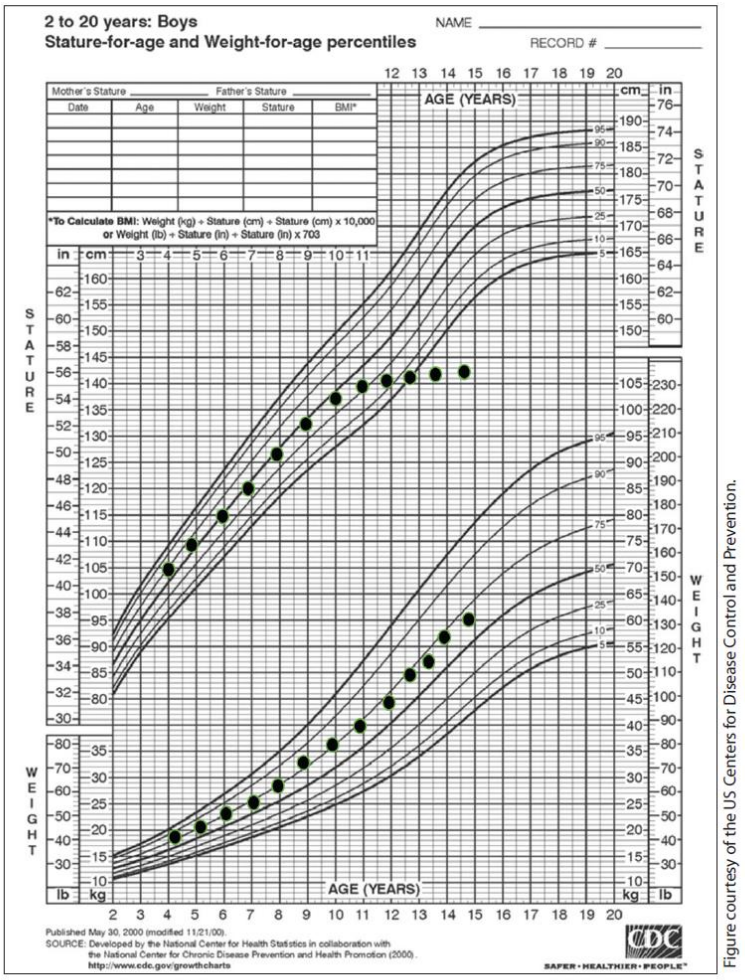


Figure 2.1: The Growth Pattern Over Time for a Patient with acquired hypothyroidism [2]

Figure 2 shows the growth pattern of the patient with hypothyroidism indicated by the dotted trendline and compared to the typical growth trendline of the average male. The patient’s trendline of height over the years is flattening out much faster than expected for his age. When measurements are accurate to the patient's height, and if it is clear that their growth pattern does not match what is the typical trend, they can be examined for any ailments that may be causing a typical pattern.

2.1. Need for Quick Height Measurements

The need for a quick height measurement is to increase the time efficiency of the patient's visit. According to the statistic chart, "Time U.S. physicians spent with each patient 2018" [3] published by John Elflein, 33% of doctors spend between 17-24 minutes with their patients.

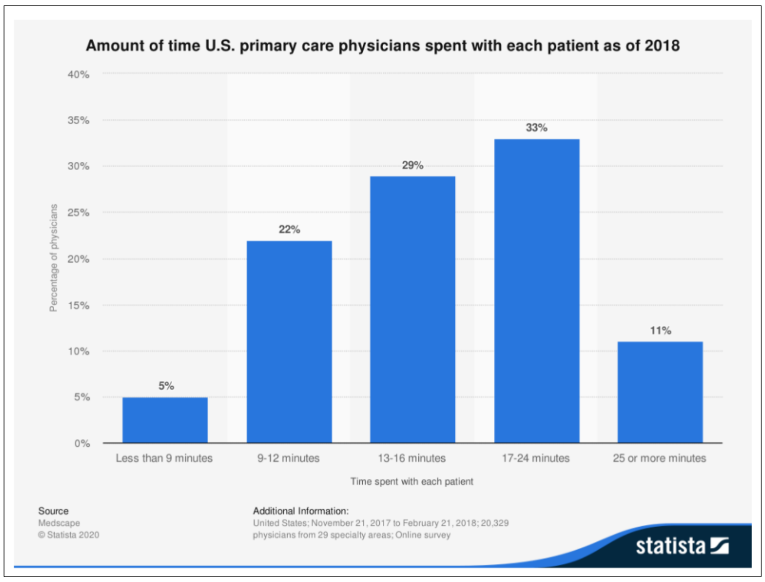


Figure 2.2: Time U.S. physicians spent with each patient in 2018 [3]

Since the patient only is allotted a certain amount of time during their visit, a faster height measurement system is important to allow for the most time to be spent addressing the specific needs of the patient. When the physical is fast, the doctor can spend more time with the patient. As discussed in the article "15-Minute Visits Take A Toll On The Doctor-Patient Relationship" [4] by Roni Caryn Rabin, there is an issue when doctor-patient meetings are too short. The more time spent between doctor and patient, the more that the patient will be actively involved in discussing their health. This discussion will help ensure the doctor understands the patient's condition to better assess their health.

2.2. Traditional Stadiometer Accuracy and Speed Statistics

Although traditional stadiometers are the standard device used by hospitals and clinics, they can sometimes result in inaccuracies. As stated in the article “Clinical height measurements are unreliable: a call for improvement” by A. L. Mikula, S. J. Hetzel, N. Binkley & P. A. Anderson [5], “When performing direct measurements on stadiometers, the mean difference from a gold standard length was 0.24 cm (SD 0.80). Nine percent of stadiometers examined had an error of >1.5 cm.” (A. L. Mikula, S. J. Hetzel, N. Binkley & P. A. Anderson) [5] The article states that, when testing traditional stadiometers, there are some errors in the height measurements. These errors, while relatively small, still negatively impact the assessment of the patient’s health, especially when a patient has an ailment such as osteoporosis in which even a small change of a few centimeters in a patient’s height is a crucial indicator used to assess their condition. [5]

The average time of a total appointment is 30 minutes. Out of this time, the physical examination with the nurse can take as little as 2 minutes, to as long as 20 minutes. Since the patients can be as young as toddlers, some may not cooperate because they are scared or upset. The average time of a physical is 5-7 minutes. The height measurement process specifically, given there are no distractions, takes about 10 seconds.

Chapter 3

Literature Survey

All the researchers have aimed to develop electronic digital stadiometers in Embedded Systems which can improve the quality and speed of height measurement. The various researches done for implementing the stadiometer shows that there's a need for automated data collection facility. So, our team proposed IoT based Stadiometer which is completely different and the world's first open-source stadiometer. Which is not only improved but also provides a completely new service of remote health monitoring. The summary of research work done is listed as follows:

3.1. Design and Development of a Microcontroller Based Stadiometer [6]

This paper presents the design and construction of a portable stadiometer for human measurements for up to approximately 200 cm, with the use of a microcontroller and ultrasonic sensor device (transducer). Ultrasonic transducers use ultrasound waves with a frequency greater than the upper limit of human hearing. The ultrasonic transducer is controlled by the PIC16F877A microcontroller to generate an ultrasound wave to be emitted through the transmitter, the pulse is reflected off an object and the echo received by the receiver derives the time traveled, calculates the distance, and then calculates the height which is then displayed on the Liquid Crystal Display. The use of the ultrasound wave finds its application in several sectors such as in the medical, industrial and security, electronic and navigational sectors.

3.2. Digital Height-Measuring Sensor Device [7]

A digital height-measurement device (stadiometer) was developed to improve the speed and accuracy of the height measurement process and address the discomfort and inefficiency issues of traditional physical height-measurement devices currently used in hospitals and clinics. The digital stadiometer uses sensor components to quickly and efficiently measure a patient's height. One key need for accurate height measurements from this device is to ensure the device is level with the ground. Therefore, this project incorporated an accelerometer along with a "bubble level game" feature, which allowed the user to 'self-level' the device in two axes by moving a black circle into a ring as viewed by the LCD screen. The device was prototyped and tested, then mounted onto a helmet for final testing and analysis. The aim is for the helmet-mounted device to be incorporated into the clinic’s routine physical measurements that they perform for each patient. Unfortunately, due to the COVID-19 lockdown, prototype testing was unable to be completed in the clinic setting with patients. However, limited testing proved that the device took quick height measurements, with height displayed in about 6 seconds, and was relatively accurate, though it did not meet the ¼” accuracy desired by the customer. Future revisions to the device will strive to improve the accuracy of the device through the implementation of a temperature sensor and more thorough testing.

3.3. Parametric Design of Height and Weight Measuring System [8]

Anthropometry medical measuring system is very important in the health sector in taking the physical measurement of patients. The research focused on the need to consider a parametric design of height and weight measuring system with the enlightenment of basic design principles. The materials involved in the parametric design of this medical measuring system under study are Load Cells with Strain Gauge (weight sensor), Sonar with Ultrasonic Sensors (height sensor), Liquid Crystal Display, Medium Density Fibreboard, Rubber Mounting, Microcontroller Arduino Board, and other important accessories. The results from the design analysis revealed that the system weight sensor platform has a sheer force of 630 N, and a maximum bending moment of 126 Nm. The compressive stress that could be induced during the measurement of maximum human weight is 6 kN/m^2 and stress that could be induced due to maximum bending moment is amount to 605 N/m^2. The amount by which the weight sensor platform could deflect when subjected to bending moment is 0.000004 m. These values indicate that the design is safe for an operation that is far lesser than the property parameters of the materials used for the weight sensor platform. The designed height of the system for measuring the height is about 2.1 m (7 Feet), and the area of the stand platform for measuring the weight is about 0.2 m 2. The weight sensor for the design could measure from 1kg to 200 kg with an accuracy of about ± 0.1 kg and the height measuring sensor could measure through a scanning range of 0.4 to 2.5 m with an accuracy of about ±1 mm.

3.4. An Ultrasonic Sensor Based Portable Height Measuring Device [9]

This project is to design a portable device that utilizes ultrasonic sensors to measure height. The device can be used as an alternative to the more bulky and expensive height measuring products found either at home or in doctor's offices. It is designed to allow users to easily measure their heights on their own. The objective is to make it both aesthetically pleasing, easy to use, and as cost-efficient as possible.

3.5. A Non-contact Human Body Height and Weight Measurement Approach Using Ultrasonic Sensor [10]

Human body height and weight are the key factors for personal health monitoring and many works have been conducted to accurately and conveniently measure these two physiology parameters. However, in many conditions such as in outer space, measuring human body weight becomes impossible for traditional approaches such as using a weight scale. In this paper, we present a medical instrument, called the Ultrasonic Measurer to measure human body height and weight using a non-contact approach. This is a low-cost, efficient device to estimate human body sizes using an ultrasonic sensor, the SRF04. This sensor provides good measurements at a close distance over a range of up to 300 cm. Moreover, we present a convenient approach to estimating human body weight from the measured sizes using the close relationship between a human body's weight and its sizes, despite the variation in shapes. The measurements and estimations were implemented on a set of 50 men and women with different body shapes. The proposed method proved to give good results as the sizes are estimated with an average error of less than 0.3% and weight is estimated with an average error of less than 2.8 %. The module is currently designed to be used as a standalone device. In the future, the module is equipped with the Internet of Things (IoT) for various remote health monitoring applications.

Chapter 4

Methodology

4.1. Internet of Things (IoT)

Figure 4.1: Things connected to IoT network

The Internet of things (IoT) describes physical objects (or groups of such objects) with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks. The Internet of things has been considered a misnomer because devices do not need to be connected to the public internet, they only need to be connected to a network and be individually addressable.

The field has evolved due to the convergence of multiple technologies, including ubiquitous computing, commodity sensors, increasingly powerful embedded systems, and machine learning. Traditional fields of embedded systems, wireless sensor networks, control systems, and automation (including home and building automation), independently and collectively enable the Internet of things. In the consumer market, IoT technology is most synonymous with products about the concept of the "smart home", including devices and appliances (such as lighting fixtures, thermostats, home security systems, cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers. IoT is also used in healthcare systems.

There are several concerns about the risks in the growth of IoT technologies and products, especially in the areas of privacy and security, and consequently, industry and governmental moves to address these concerns have begun, including the development of international and local standards, guidelines, and regulatory frameworks.

Organizational applications: Medical and healthcare

Figure 4.2: Medical things connected to IoT network

The Internet of Medical Things (IoMT) is an application of the IoT for medical and health-related purposes, data collection and analysis for research, and monitoring. The IoMT has been referenced as "Smart Healthcare", as the technology for creating a digitized healthcare system, connecting available medical resources and healthcare services.

IoT devices can be used to enable remote health monitoring and emergency notification systems. These health monitoring devices can range from blood pressure and heart rate monitors to advanced devices capable of monitoring specialized implants, such as pacemakers, Fitbit electronic wristbands, or advanced hearing aids. Some hospitals have begun implementing "smart beds" that can detect when they are occupied and when a patient is attempting to get up. It can also adjust itself to ensure appropriate pressure and support are applied to the patient without the manual interaction of nurses. A 2015 Goldman Sachs report indicated that healthcare IoT devices "can save the United States more than $300 billion in annual healthcare expenditures by increasing revenue and decreasing cost." Moreover, the use of mobile devices to support medical follow-up led to the creation of 'm-health', used analyzed health statistics."

Specialized sensors can also be equipped within living spaces to monitor the health and general well-being of senior citizens, while also ensuring that proper treatment is being administered and assisting people to regain lost mobility via therapy as well. These sensors create a network of intelligent sensors that can collect, process, transfer, and analyze valuable information in different environments, such as connecting in-home monitoring devices to hospital-based systems. Other consumer devices to encourage healthy living, such as connected scales or wearable heart monitors, are also a possibility with the IoT. End-to-end health monitoring IoT platforms are also available for antenatal and chronic patients, helping one manage health vitals and recurring medication requirements. Advances in plastic and fabric electronics fabrication methods have enabled ultra-low-cost, use-and-throw IoMT sensors. These sensors, along with the required RFID electronics, can be fabricated on paper or e-textiles for wireless-powered disposable sensing devices. Applications have been established for point-of-care medical diagnostics, where portability and low system complexity is essential.

As of 2018 IoMT was not only being applied in the clinical laboratory industry but also in the healthcare and health insurance industries. IoMT in the healthcare industry is now permitting doctors, patients, and others, such as guardians of patients, nurses, families, and similar, to be part of a system, where patient records are saved in a database, allowing doctors and the rest of the medical staff to have access to patient information. Moreover, IoT-based systems are patient-centered, which involves being flexible to the patient's medical conditions. IoMT in the insurance industry provides access to better and new types of dynamic information. This includes sensor-based solutions such as biosensors, wearables, connected health devices, and mobile apps to track customer behavior. This can lead to more accurate underwriting and new pricing models.

The application of the IoT in healthcare plays a fundamental role in managing chronic diseases and in disease prevention and control. Remote monitoring is made possible through the connection of powerful wireless solutions. The connectivity enables health practitioners to capture patients' data and apply complex algorithms in health data analysis.

4.2 IoT Cloud

Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. Large clouds often have functions distributed over multiple locations, each location being a data center. Cloud computing relies on sharing of resources to achieve coherence and typically uses a "pay-as-you-go" model which can help in reducing capital expenses but may also lead to unexpected operating expenses for unaware users. An IoT cloud is a massive network that supports IoT devices and applications. This includes the underlying infrastructure, servers, and storage, needed for real-time operations and processing.

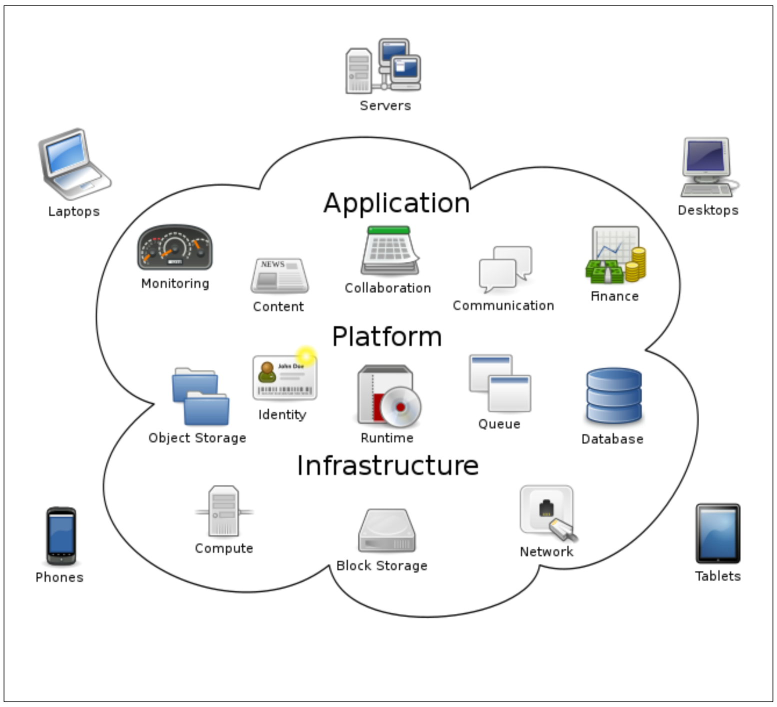


Figure 4.3: Cloud computing metaphor - the group of networked elements providing services need not be individually addressed or managed by users; instead, the entire provider-managed suite of hardware and software can be thought of as an amorphous cloud.

An IoT cloud also includes the services and standards necessary for connecting, managing, and securing different IoT devices and applications. IoT clouds offer an efficient, flexible, and scalable model for delivering the infrastructure and services needed to power IoT devices and applications for businesses with limited resources. IoT clouds offer on-demand, cost-efficient hyper-scale so organizations can leverage the significant potential of IoT without having to build the underlying infrastructure and services from scratch.

Service Models:

Though service-oriented architecture advocates "Everything as a service" (with the acronyms EaaS or XaaS, or simply aas), cloud-computing providers offer their "services" according to different models, of which the three standard models per NIST are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These models offer increasing abstraction; they are thus often portrayed as layers in a stack: infrastructure-, platform- and software-as-a-service, but these need not be related. For example, one can provide SaaS implemented on physical machines (bare metal), without using underlying PaaS or IaaS layers, and conversely, one can run a program on IaaS and access it directly, without wrapping it as SaaS.

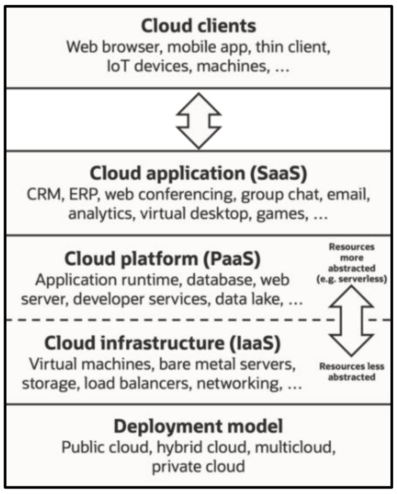

Cloud clients
Web browser, mobile app, thin client, IoT devices, machines, ...
Cloud application (SaaS)
CRM, ERP, web conferencing, group chat, email, analytics, virtual desktop, games, ...
Cloud platform (PaaS)
Application runtime, database, web server, developer services, data lake, ...
Resources more abstracted (e.g. serverless)
Cloud infrastructure (IaaS)
Virtual machines, bare metal servers, storage, load balancers, networking, ...
Resources less abstracted
Deployment model
Public cloud, hybrid cloud, multicloud, private cloud

Figure 4.4: Cloud computing service models arranged as layers in a stack

Infrastructure as a service (IaaS)

"Infrastructure as a service" (IaaS) refers to online services that provide high-level APIs used to abstract various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning, scaling, security, backup, etc. A hypervisor runs the virtual machines as guests. Pools of hypervisors within the cloud operational system can support large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements. Linux containers run in isolated partitions of a single Linux kernel running directly on the physical hardware. Linux cgroups and namespaces are the underlying Linux kernel technologies used to isolate, secure, and manage the containers. Containerisation offers higher performance than virtualization because there is no hypervisor overhead. IaaS clouds often offer additional resources such as a virtual-machine disk-image library, raw block storage, file or object storage, firewalls, load balancers, IP addresses, and virtual local area networks (VLANs), and software bundles.

Platform as a service (PaaS)

The NIST's definition of cloud computing defines Platform as a Service as:

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Software as a service (SaaS)

The NIST's definition of cloud computing defines Software as a Service as:

The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

In the software as a service (SaaS) model, users gain access to application software and databases. Cloud providers manage the infrastructure and platforms that run the applications. SaaS is sometimes referred to as "on-demand software" and is usually priced on a pay-per-use basis or using a subscription fee. In the SaaS model, cloud providers install and operate application software in the cloud and cloud users access the software from cloud clients. Cloud users do not manage the cloud infrastructure and platform where the application runs. This eliminates the need to install and run the application on the cloud user's computers, which simplifies maintenance and support. Cloud applications differ from other applications in their scalability—which can be achieved by cloning tasks onto multiple virtual machines at run-time to meet changing work demands. Load balancers distribute the work over the set of virtual machines. This process is transparent to the cloud user, who sees only a single access point. To accommodate a large number of cloud users, cloud applications can be multitenant, meaning that any machine may serve more than one cloud-user organization.

Deployment models:

Private cloud

A private cloud is a cloud infrastructure operated solely for a single organization, whether managed internally or by a third party and hosted either internally or externally. Undertaking a private cloud project requires significant engagement to virtualize the business environment, and requires the organization to reevaluate decisions about existing resources. It can improve business, but every step in the project raises security issues that must be addressed to prevent serious vulnerabilities. Self-run data centers are generally capital intensive. They have a significant physical footprint, requiring allocations of space, hardware, and environmental controls. These assets have to be refreshed periodically, resulting in additional capital expenditures. They have attracted criticism because users "still have to buy, build, and manage them" and thus do not benefit from less hands-on management, essentially "(lacking) the economic model that makes cloud computing such an intriguing concept".

Public cloud

Cloud services are considered "public" when they are delivered over the public Internet, and they may be offered as a paid subscription, or free of charge. Architecturally, there are few differences between public- and private cloud services, but security concerns increase substantially when services (applications, storage, and other resources) are shared by multiple customers. Most public cloud providers offer direct-connection services that allow customers to securely link their legacy data centers to their cloud-resident applications.

Hybrid cloud

A hybrid cloud is a composition of a public cloud and a private environment, such as a private cloud or on-premises resources, that remain distinct entities but are bound together, offering the benefits of multiple deployment models. A hybrid cloud can also mean the ability to connect collocation, managed, and/or dedicated services with cloud resources. Gartner defines a hybrid cloud service as a cloud computing service that is composed of some combination of private, public, and community cloud services, from different service providers. A hybrid cloud service crosses isolation and provider boundaries so that it can't be simply put in one category of private, public, or community cloud service. It allows one to extend either the capacity or the capability of a cloud service, by aggregation, integration, or customization with another cloud service.

4.3 Wi-Fi Protocol

Wi-Fi or WiFi is a family of wireless network protocols, based on the IEEE 802.11 family of standards, which are commonly used for local area networking of devices and Internet access, allowing nearby digital devices to exchange data by radio waves. These are the most widely used computer networks in the world, used globally in home and small office networks to link desktop and laptop computers, tablet computers, smartphones, smart TVs, printers, and smart speakers together and to a wireless router to connect them to the Internet, and in wireless access points in public places like coffee shops, hotels, libraries and airports to provide the public Internet access for mobile devices.

Wi-Fi is a trademark of the non-profit Wi-Fi Alliance, which restricts the use of the term Wi-Fi Certified to products that complete interoperability certification testing. As of 2017, the Wi-Fi Alliance consisted of more than 800 companies from around the world. As of 2019, over 3.05 billion Wi-Fi-enabled devices are shipped globally each year. Wi-Fi uses multiple parts of the IEEE 802 protocol family and is designed to interwork seamlessly with its wired sibling, Ethernet. Compatible devices can network through wireless access points to each other as well as to wired devices and the Internet. The different versions of Wi-Fi are specified by various IEEE 802.11 protocol standards, with the different radio technologies determining radio bands, and the maximum ranges, and speeds that may be achieved. Wi-Fi most commonly uses the 2.4 gigahertz (120 mm) UHF and 5 gigahertz (60 mm) SHF radio bands; these bands are subdivided into multiple channels. Channels can be shared between networks but only one transmitter can locally transmit on a channel at any moment in time.

4.4 MQTT IoT Protocol

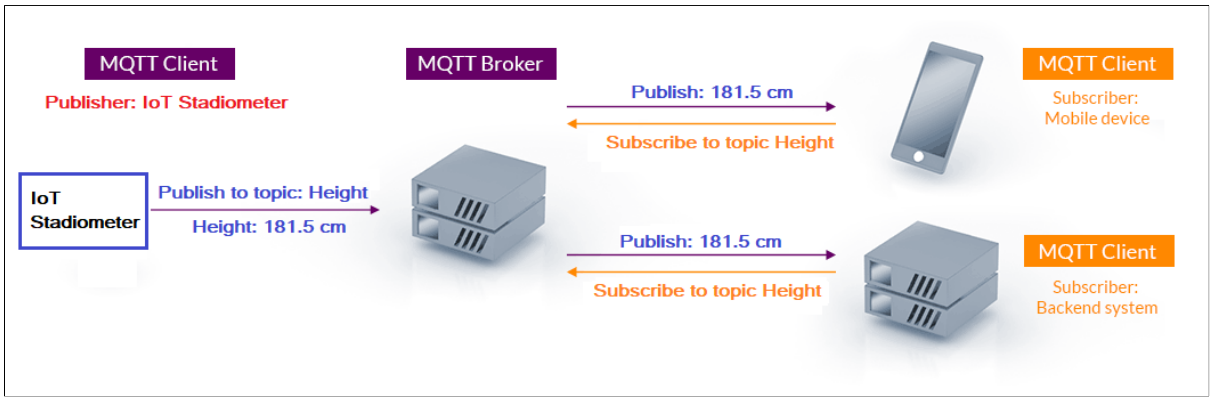


Figure 4.5: Structure of MQTT publisher and subscriber model

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc. MQTT (originally an initialism of MQ Telemetry Transport) is a lightweight, publish-subscribe, machine-to-machine network protocol. It is designed for connections with remote locations that have devices with resource constraints or limited network bandwidth. It must run over a transport protocol that provides ordered, lossless, bi-directional connections—typically, TCP/IP. It is an open OASIS standard and an ISO recommendation (ISO/IEC 20922).

The MQTT protocol defines two types of network entities: a message broker and several clients. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device (from a microcontroller up to a fully-fledged server) that runs an MQTT library and connects to an MQTT broker over a network. Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number of locations of subscribers, and subscribers, in turn, do not have to be configured with any data about the publishers.

If a broker receives a message on a topic for which there are no current subscribers, the broker discards the message unless the publisher of the message designated the message as a retained message. A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding QoS for the selected topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic. This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher.

When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker. Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics. A minimal MQTT control message can be as little as two bytes of data. A control message can carry nearly 256 megabytes of data if needed. There are fourteen defined message types used to connect and disconnect a client from a broker, publish data, acknowledge receipt of data, and supervise the connection between client and server. MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth. MQTT sends connection credentials in plain text format and does not include any measures for security or authentication. This can be provided by using TLS to encrypt and protect the transferred information against interception, modification, or forgery. The default unencrypted MQTT port is 1883. The encrypted port is 8883.

MQTT broker

The MQTT broker is a piece of software running on a computer (running on-premises or in the cloud) and could be self-built or hosted by a third party. It is available in both open source and proprietary implementations. The broker acts as a post office. MQTT clients don't use a direct connection address of the intended recipient but use the subject line called "Topic". Anyone who subscribes receives a copy of all messages for that topic. Multiple clients can subscribe to a topic from a single broker (one to many capabilities), and a single client can register subscriptions to topics with multiple brokers (many to one). Each client can both produce and receive data by both publishing and subscribing, i.e., the devices can publish sensor data and still be able to receive the configuration information or control commands (MQTT is a bi-directional communication protocol). This helps in both sharing data and managing and controlling devices. A client cannot broadcast the same data to a range of topics and must publish multiple messages to the broker, each with a single topic given.

With MQTT broker architecture, the client devices and server applications become decoupled. In this way, the clients are kept unaware of each other's information. MQTT uses TLS encryption with username and password-protected connections. Optionally, the connection may require certification, in the form of a certificate file that a client provides and must match with the server's copy. In case of failure, broker software and clients can automatically hand over to a Redundant/automatic backup broker. Backup brokers can also be set up to share a load of clients across multiple servers onsite, in the cloud, or a combination of these.

The main advantages of MQTT broker are:

1. Eliminates vulnerable and insecure client connections.
2. Can easily scale from a single device to thousands.
3. Manages and tracks all client connection status, including security credentials and certificates.
4. Reduced network strain without compromising the security (cellular or satellite network)

Message types

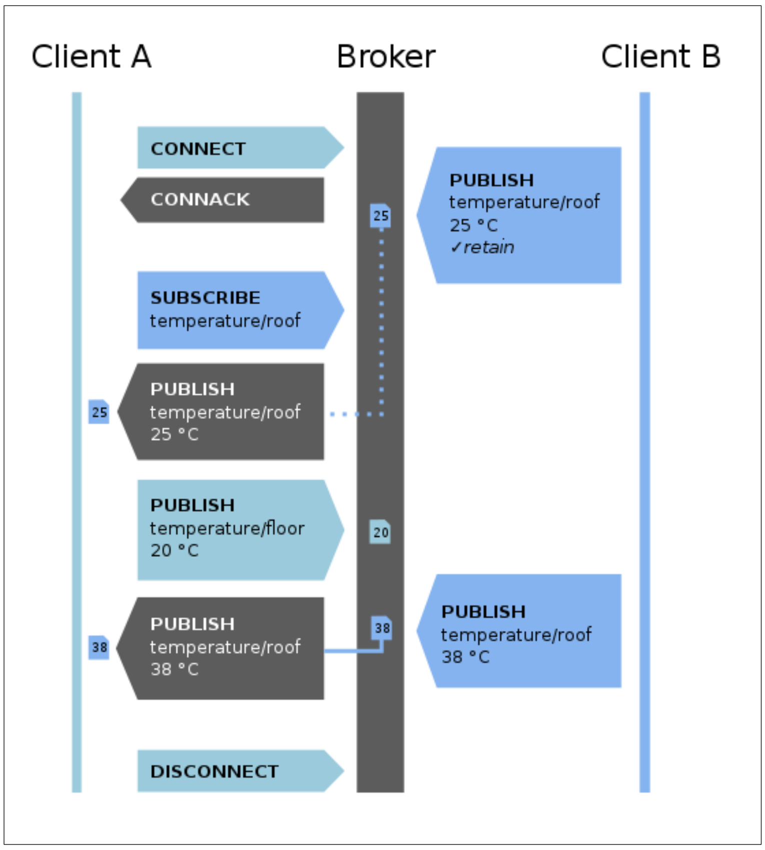

Client A
Broker
Client B
CONNECT
CONNACK
SUBSCRIBE temperature/roof
PUBLISH temperature/roof 25 °C
PUBLISH temperature/floor 20 °C
PUBLISH temperature/roof 38 °C
DISCONNECT
PUBLISH temperature/roof 25 °C ✓retain
PUBLISH temperature/roof 38 °C
25
20
38

Figure 4.6: Example of an MQTT connection (QoS 0) with connect, publish/subscribe, and disconnect. The first message from client B is stored due to the retain flag.

- Connect: Waits for a connection to be established with the server and creates a link between the nodes.
- Disconnect: Waits for the MQTT client to finish any work it must do, and for the TCP/IP session to disconnect.
- Publish Returns immediately to the application thread after passing the request to the MQTT client.

Key benefits of MQTT

- Lightweight and Efficient: MQTT clients are very small, and require minimal resources so can be used on small microcontrollers. MQTT message headers are small to optimize network bandwidth.
- Bi-directional Communications: MQTT allows for messaging between the device to the cloud and the cloud to the device. This makes for easy broadcasting messages to groups of things.
- Scale to Millions of Things: MQTT can scale to connect with millions of IoT devices.

- Reliable Message Delivery: Reliability of message delivery is important for many IoT use cases. This is why MQTT has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once
- Support for Unreliable Networks: Many IoT devices connects over unreliable cellular networks. MQTT's support for persistent sessions reduces the time to reconnect the client with the broker.
- Security Enabled: MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.

Quality of service

Each connection to the broker can specify a quality of service (QoS) measure. These are classified in increasing order of overhead:

- At most once – the message is sent only once and the client and broker take no additional steps to acknowledge delivery (fire and forget).
- At least once – the message is re-tried by the sender multiple times until acknowledgment is received (acknowledged delivery).
- Exactly once – the sender and receiver engage in a two-level handshake to ensure only one copy of the message is received (assured delivery).

  This field does not affect the handling of the underlying TCP data transmissions; it is only used between MQTT senders and receivers.

Applications

Several projects implement MQTT, for example:

- OpenHAB the Open-source software home automation platform embeds an MQTT binding.
- The Open Geospatial Consortium SensorThings API standard specification has an MQTT extension in the standard as an additional message protocol binding. It was demonstrated in a US Department of Homeland Security IoT Pilot.
- Node-RED supports MQTT with TLS nodes as of version 0.14.
- Home Assistant the Open-source software home automation platform is MQTT enabled and offers a Mosquitto broker add-on.
- ejabberd supports MQTT as of version 19.02.
- Eclipse Foundation manages a Sparkplug protocol specification compatible with MQTT. It builds on top of MQTT adding requirements needed in real-time industrial applications.

Chapter 5

Experimentation

5.1 Requirement Analysis: Hardware Requirement

5.1.a NodeMCU ESP8266 12-E Devkit v1.0

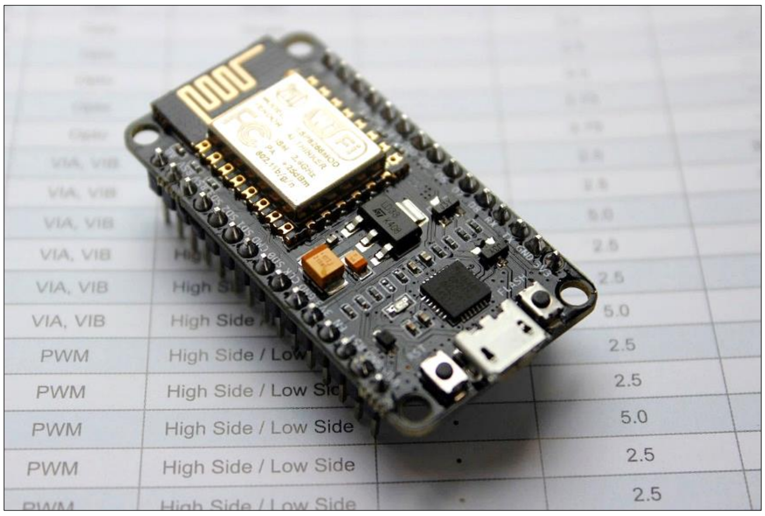


Figure 5.1: NodeMCU DEVKIT 1.0

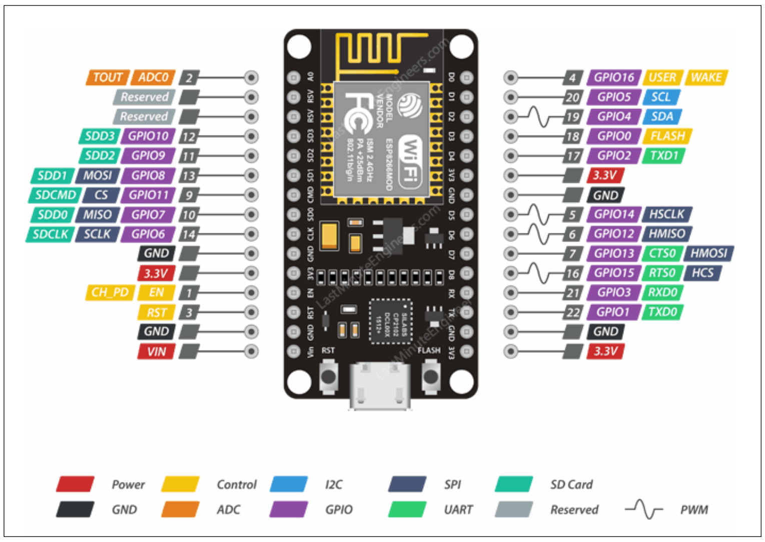


Figure 5.2: Pinout diagram of NodeMCU

NodeMCU is a low-cost open-source IoT platform. It initially included firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems and hardware that was based on the

ESP-12 module. NodeMCU is implemented in C and is layered on the Espressif NON-OS SDK. NodeMCU is an open-source firmware for which open-source prototyping board designs are available. The name "NodeMCU" combines "node" and "MCU" (micro-controller unit). The term "NodeMCU" strictly speaking refers to the firmware rather than the associated development kits. Both the firmware and prototyping board designs are open source. The firmware uses the Lua scripting language. The firmware is based on the eLua project and built on the Espressif Non-OS SDK for ESP8266. It uses many open-source projects, such as lua-cjson and SPIFFS. Due to resource constraints, users need to select the modules relevant to their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented.

The prototyping hardware typically used is a circuit board functioning as a dual in-line package (DIP) which integrates a USB controller with a smaller surface-mounted board containing the MCU and antenna. The choice of the DIP format allows for easy prototyping on breadboards. The design was initially based on the ESP-12 module of the ESP8266, which is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications. There are two available versions of NodeMCU version 0.9 & 1.0 where the version 0.9 contains ESP-12 and version 1.0 contains ESP-12E where E stands for "Enhanced". The Development Kit based on ESP8266 integrates GPIO, PWM, IIC, 1-Wire, and ADC all in one board. It powers the development in the fastest way combine with NodeMCU Firmware!

5.1.b Ultrasonic Distance Sensor HC-SR04

Ultrasound is high-pitched sound waves with frequencies higher than the audible limit of human hearing. Human ears can hear sound waves that vibrate in the range from about 20 times a second (a deep rumbling noise) to about 20,000 times a second (a high-pitched whistling). However, ultrasound has a frequency of over 20,000 Hz and is therefore inaudible to humans.

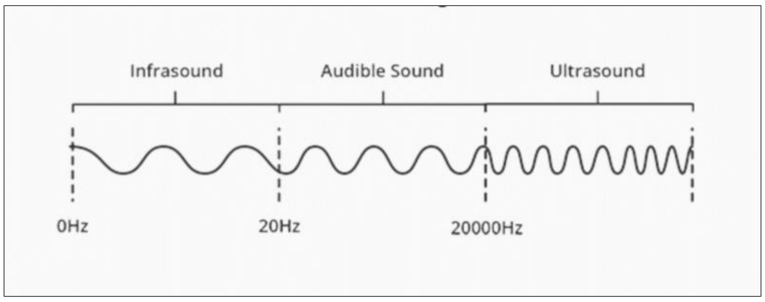


Figure 5.3: Sound frequency density over a range

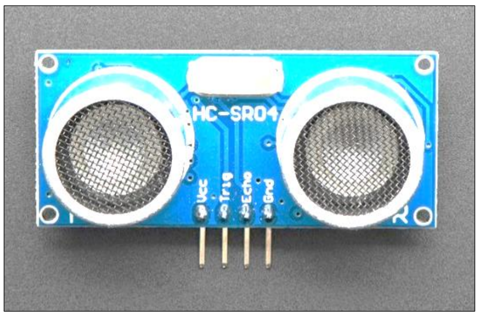


Figure 5.4: HC-SR04 Sensor Module

HC-SR04 Ultrasonic Sonar Distance Sensor is used for automation, interactive art, and motion sensing. This sensor is fast, easy to use, low cost, and is common in robotics projects. At its core, the HC-SR04 Ultrasonic distance sensor consists of two ultrasonic transducers. The one acts as a transmitter that converts an electrical signal into 40 kHz ultrasonic sound pulses. The receiver listens for the transmitted pulses. If it receives them, it produces an output pulse whose width can be used to determine the distance the pulse traveled. The device's Trigger signal can be 3V or 5V, while the "return" Echo signal is 5V logic. For that reason, two 10K resistors are included for use as a divider to convert the 5V logic level to a safe 2.5V for 3V devices. The HC-SR04 sensor works at about 2 cm to 400 cm away, but 10 cm - 250 cm can provide ideal results. The ranging accuracy can reach 3 mm. The modules include ultrasonic transmitters, receivers, and control circuits.

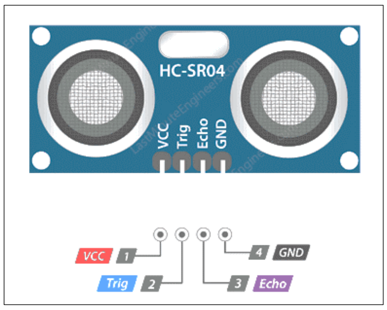


Figure 5.5: HC-SR04 Ultrasonic Sensor Pinout

- VCC - Power pin - this sensor requires 5V power for best results.
- Trig - Signal can be 3V or 5V. Setting the Trig pin too high for 10µs causes the sensor to initiate an ultrasonic burst.
- Echo - "Return" signal is 5 V logic - the sensor comes with two 10k resistors to use as a divider to convert the 5 V logic level to a safe 2.5 V that you can read with your 3V device. The Echo pin goes high when the ultrasonic burst is transmitted (in response to

Trig) and stays high until the sensor receives an echo of its burst, at which point it goes low. By measuring the time, the Echo pin is high, the distance can be computed.

- GND - Ground pin.

The basic principle of work:

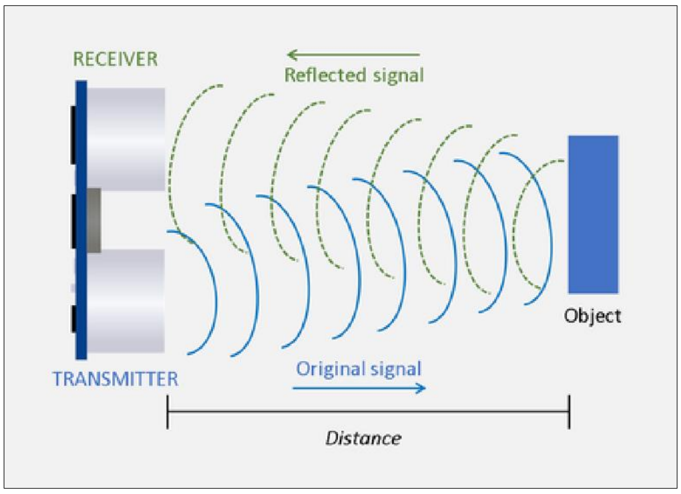


Figure 5.6: Working Principle of HC-SR04

1. Using I/O trigger for at least 10 us a high-level signal,

2. Module automatically sends eight 40 kHz and detects whether there is a pulse signal back.

3. If the signal back, through a high level, the time of high output I/O duration is the time from sending ultrasonic to returning.

The time between the transmission and reception of the signal allows us to calculate the distance to an object. This is possible because we know the sound's velocity in the air.

Test distance = (high level time × velocity of sound (340 m/s) / 2

Working Voltage	DC +5V
Working Current	15 mA
Quiescent Current	< 2 mA
Working Frequency	40 kHz
Maximum Range	400 cm
Minimum Range	2 cm
Resolution	0.3 cm
Effective Angle	<15°
Measuring Angle	30°

Trigger Input Signal Pulse Width	10 uS high TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45×20×15 mm
Weight	8.7 g

Table 5.1: Technical Parameters of HC-SR04

Timing diagram:

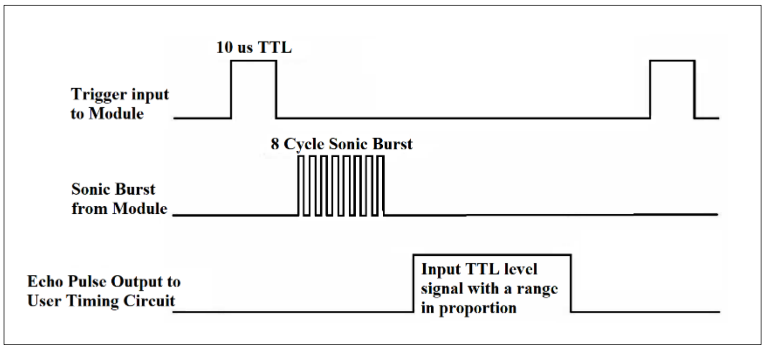

10 us TTL
Trigger input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timing Circuit
Input TTL level signal with a range in proportion

Figure 5.7: Timing Diagram of Ultrasonic distance sensor

We only need to supply a short 10 uS pulse to the trigger input to start the ranging, and then the module will send out an 8-cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion. You can calculate the range through the time interval between sending the trigger signal and receiving the echo signal. The best is to use over 60 ms measurement cycle, to prevent trigger signal to the echo signal.

Formula: uS / 58 = centimetres or uS / 148 = inch

Features:

- Measures the distance within a wide range of 2 cm to 400 cm with stable performance
- Accurate distance measurement
- High-density and small blind distance

5.1.c 16×2 I2C LCD Module

I2C interface 16x2 LCD module is a high-quality 2-line 16-character LCD module with an onboard contrast control adjustment, backlight, and I2C communication interface. LCD1602 Parallel LCD Display provides a simple and cost-effective solution for adding a 16×2 white text on RGB Liquid Crystal Display into a project. The display is 16 character 2-line display that has a very clear and high contrast white text upon a blue backlight. This display is very easy to interface with Arduino or other microcontrollers. The I2C is a type of serial bus developed by Philips, which uses two bidirectional lines, called SDA (Serial Data Line) and SCL (Serial Clock Line). Both must be connected via pulled-up resistors. The usage voltages are standard as 5V and 3.3V.

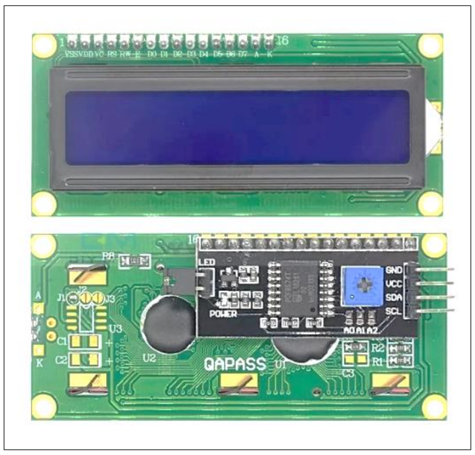


Figure 5.8: 16×2 I2C LCD Module

I2C (Inter-Integrated Circuit, eye-squared-C), alternatively known as I2C or IIC, is an asynchronous, multi-controller/multi-target (controller/target), packet-switched, single-ended, serial communication bus invented in 1982 by Philips Semiconductors. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. Several competitors, such as Siemens, NEC, Texas Instruments, STMicroelectronics, Motorola, Nordic Semiconductor, and Intersil, have introduced compatible I2C products to the market since the mid-1990s. System Management Bus (SMBus), defined by Intel in 1995, is a subset of I2C, defining a stricter usage. One purpose of SMBus is to promote robustness and interoperability. Accordingly, modern I2C systems incorporate some policies and rules from SMBus, sometimes supporting both I2C and SMBus, requiring only minimal reconfiguration either by commanding or output pin use.

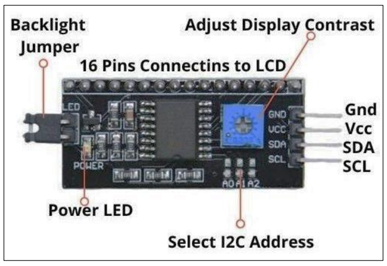


Figure 5.9: I2C Module of 16×2 LCD

Features:

- I2C Reduces the overall wirings.
- 16 characters wide, 2 rows.
- I2C adapter allows flexibility in connections.
- The Single LED backlight included can be dimmed easily with a resistor or PWM.
- IIC / I2C interface was developed to reduce the IO port usage on the NodeMCU.
- Display Type: Negative white on the blue backlight.
- Interface: I2C to 4 bits LCD data and control lines.
- Contrast Adjustment: built-in Potentiometer.
- Backlight Control: Firmware or jumper wire.

Model	**LCD1602**
Characters	16
Character Colour	White
Backlight	Blue
Input Voltage	5 V
Length	36 mm
Width	80 mm
Height	18 mm
Weight	35 gm

Table 5.2: Technical Specifications of I2C Module of 16×2 LCD

5.1.d Mini Buzzer

Figure 5.10: Buzzer

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers and beepers include alarm devices, timers, training, and confirmation of user input such as a mouse click or keystroke. Piezoelectric buzzers, or piezo buzzers, as they are sometimes called, were invented by Japanese manufacturers and fitted into a wide array of products during the 1970s to 1980s. This advancement mainly came about because of cooperative efforts by Japanese manufacturing companies. In 1951, they established the Barium Titanate Application Research

Committee, which allowed the companies to be "competitively cooperative" and bring about several piezoelectric innovations and inventions. A piezoelectric element may be driven by an oscillating electronic circuit or other audio signal sources, driven with a piezoelectric audio amplifier. Sounds commonly used to indicate that a button has been pressed are a click, a ring, or a beep.

5.1.e Red LED

Figure 5.11: 5 mm Red LED

A light-emitting diode (LED) is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons. The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the bandgap of the semiconductor. White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.

Specifications:

- Dice material: GaAlAs
- Emitted color: Super Red
- Lens color: Red Transparent
- Peak wavelength: 660 nm
- Viewing angle: 16°
- Luminous intensity (IV): 250 mcd

5.1.f Resistor

The resistor is a passive two-terminal electrical component that implements electrical resistance as a circuit element. In electronic circuits, resistors are used to reduce current flow, adjust signal levels, divide voltages, bias active elements, and terminate transmission lines, among other uses. High-power resistors that can dissipate many watts of electrical power as heat may be used as part of motor controls, in power distribution systems, or as test loads for generators.

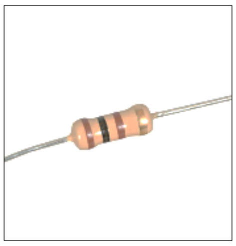


Figure 5.12: 100 Ω resistor

Fixed resistors have resistances that only change slightly with temperature, time, or operating voltage. Variable resistors can be used to adjust circuit elements (such as volume control or a lamp dimmer), or as sensing devices for heat, light, humidity, force, or chemical activity.

5.1.g Perfboard

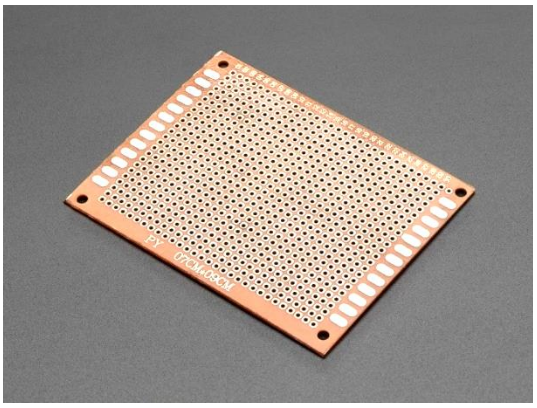


Figure 5.13: Top of a copper-clad Perfboard with solder pads for each hole.

Perfboard is material for prototyping electronic circuits (also called DOT PCB). It is a thin, rigid sheet with holes pre-drilled at standard intervals across a grid, usually a square grid of 0.1 inches (2.54 mm) spacing. These holes are ringed by round or square copper pads, though bare boards are also available. Inexpensive perfboard may have pads on only one side of the board, while better quality perfboard can have pads on both sides (plated-through holes). Since each pad is electrically isolated, the builder makes all connections with either wire wrap or miniature point-to-point wiring techniques. Discrete components are soldered to the prototype board such as resistors, capacitors, and integrated circuits.

5.1.h Power Supply

A power supply is an electrical device that supplies electric power to an electrical load. The main purpose of a power supply is to convert electric current from a source to the correct voltage, current, and frequency to power the load. As a result, power supplies are sometimes referred to as electric power converters. Some power supplies are separate standalone pieces of equipment, while others are built into the load appliances that they power. Examples of the latter include power supplies found in desktop computers and consumer electronics devices. Other functions that power supplies may perform include limiting the current drawn by the load to safe levels, shutting off the current in the event of an electrical fault, power conditioning to prevent electronic noise or voltage surges on the input from reaching the load, power-factor correction, and storing energy so it can continue to power the load in the event of a temporary interruption in the source power (uninterruptible power supply).

Switched-mode Power Supply:

Figure 5.14: SMPS Power Supply Adapter

In a switched-mode power supply (SMPS), the AC mains input is directly rectified and then filtered to obtain a DC voltage. The resulting DC voltage is then switched on and off at a high frequency by electronic switching circuitry, thus producing an AC that will pass through a high-frequency transformer or inductor. Switching occurs at a very high frequency (typically 10 kHz — 1 MHz), thereby enabling the use of transformers and filter capacitors that are much smaller, lighter, and less expensive than those found in linear power supplies operating at mains frequency. After the inductor or transformer is secondary, the high-frequency AC is rectified and filtered to produce the DC output voltage.

Features:

- Input voltage: 100-240 V, AC 50-60 Hz
- Stabilized output, 5.5 mm output DC plug, low ripple, low interference
- Noise output for device safety, High efficiency, and low no-load power consumption
- Power red LED indicator, short circuit, and overload protection

5.1.i Stadiometer Stand

A mechanical support structure for making the stadiometer structure is built using steel rods having a metal sheet thickness of 2 mm. The rods are square with 2.5×2.5 cm of cross-sectional area. The stand structure is hollow and light in weight. The footpad is built using a square steel frame with thin metal (thickness: 2 mm) sheet having a 45×45 cm cross-sectional area on top of the frame. The length or height of the IoT Stadiometer is 200 cm.

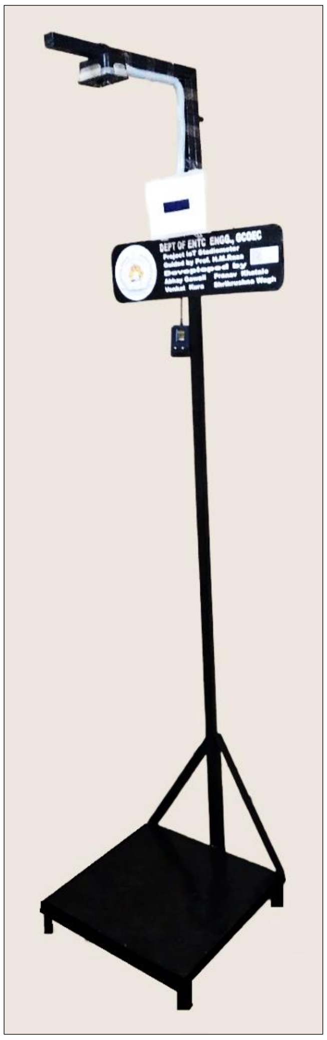
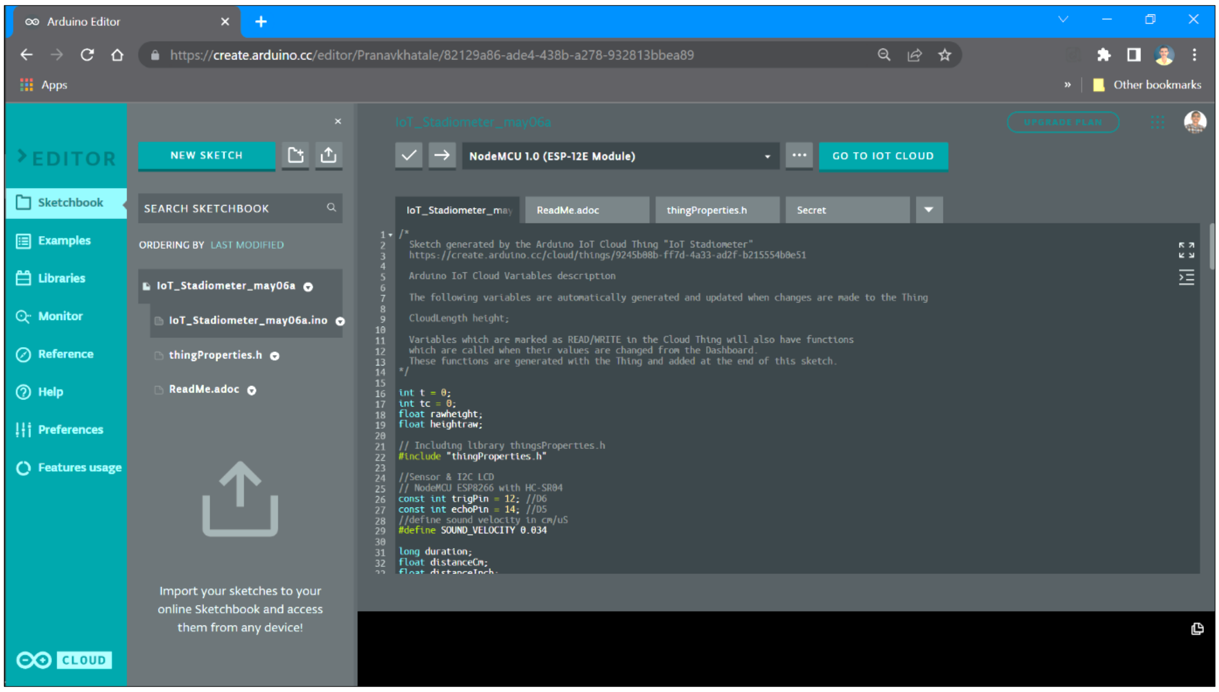


Figure 5.15: IoT Stadiometer

5.2 Software Requirement

5.2.a Arduino Web Editor



```
/*
  Sketch generated by the Arduino IoT Cloud Thing "IoT Stadiometer"
  https://create.arduino.cc/cloud/things/9245b08b-ff7d-4a33-ad2f-b215554b0e51

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes are made to the Thing

  CloudLength height;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/

int t = 0;
int tc = 0;
float rawheight;
float heightraw;

// Including library thingsProperties.h
#include "thingProperties.h"

//Sensor & I2C LCD
// NodeMCU ESP8266 with HC-SR04
const int trigPin = 12; //D6
const int echoPin = 14; //D5
//define sound velocity in cm/uS
#define SOUND_VELOCITY 0.034

long duration;
float distanceCm;
```

Figure 5.16: User Interface of Arduino Web Editor

The Arduino Web Editor allows to write the source code (a.k.a sketches), compile the sketches, and upload sketches to the NodeMCU board after installing a simple plug-in i.e Arduino Create Agent, sketchbook will be stored in the Arduino IoT Cloud and accessible from any device. Users can even import the Sketchbook via a .zip file! What's more, sharing a sketch is now as easy as sharing a link. The Web Editor is part of Arduino Create, a platform that simplifies making a project as a whole, without having to switch between many different tools to manage the various aspects of whatever you are making. It includes a code editor with features such as text cutting and pasting, searching and replacing text, automatic indenting, brace matching, and syntax highlighting, and provides simple one-click mechanisms to compile and upload programs to an Arduino board. It also contains a message area, a text console, a toolbar with buttons for common functions, and a hierarchy of operation menus. It supports the languages C and C++ using special rules of code structuring. User-written code only requires two basic functions, for starting the sketch and the main program loop, which is compiled and linked with a program stub main () into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution.

Sketch: A sketch is a program written with the Arduino IDE. Sketches are saved on the development computer as text files with the file extension .ino

A minimal Arduino C/C++ program consists of only two functions:

setup(): This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch. It is analogous to the function main().

loop(): After setup() function exits (ends), the loop() function is executed repeatedly in the main program. It controls the board until the board is powered off or reset. It is analogous to the function while(1)

Libraries: The open-source nature of the Arduino project has facilitated the publication of many free software libraries that other developers use to augment their projects.

5.2.b Arduino IoT Cloud

The Arduino IoT Cloud is an online platform that makes it easy to create, deploy and monitor IoT projects. Connected devices around the world are increasing by billions every year. The Arduino IoT Cloud is a platform that allows anyone to create IoT projects, with a user-friendly interface, and an all-in-one solution for configuration, writing code, uploading, and visualization.

Arduino IoT Cloud features:

- Data Monitoring - Useful for easily monitoring the IoT board’s sensor values through a dashboard.
- Variable Synchronisation - Variable synchronization allows to sync of variables across devices, enabling communication between devices with minimal coding.
- Scheduler - Schedule jobs to go on/off for a specific amount of time (seconds, minutes, hours).
- Over-The-Air (OTA) Uploads - Upload code to devices not connected to the computer.
- Webhooks - Integrate the project with another service, such as IFTTT.
- Amazon Alexa Support - Make the project voice-controlled with the Amazon Alexa integration.
- Dashboard Sharing - Share the data with other people around the world.

To use the Arduino IoT Cloud, a cloud-compatible board is required. We can choose between using an official Arduino board, or a third-party board based on the ESP32 / ESP8266 microcontroller with support for Wi-Fi. The Arduino IoT Cloud currently supports connection via Wi-Fi, LoRaWAN (via The Things Network), and mobile networks. To set them up, the third-party option in the device setup is available.

5.2.c Webhooks with Arduino IoT Cloud

Webhooks allow to send and receive automated messages to and from other services. For example, a developer can use webhooks to receive a notification when a property of our Thing changes. There are third-party platforms like IFTTT, and Zapier that link the properties from Arduino Cloud projects to the desired trigger action.

5.2.d IFTTT

If This Then That (commonly known as IFTTT) is a private commercial company that runs services that allow a user to program a response to events in the world. IFTTT has partnerships with different service providers that supply event notifications to IFTTT and execute commands that implement the responses. Some event and command interfaces are simply public APIs.[6]

IFTTT employs the following concepts:

- Services (formerly known as channels) are the basic building blocks of IFTTT. They mainly describe a series of data from a certain web service such as YouTube or eBay. Services can also describe actions controlled with certain APIs, like SMS. Sometimes, they can represent information in terms of whether or stocks. Each service has a particular set of triggers and actions.
- Triggers are the "this" part of an applet. They are the items that trigger the action. For example, from an RSS feed, you can receive a notification based on a keyword or phrase.
- Actions are the "that" part of an applet. They are the output that results from the input of the trigger.
- Applets (formerly known as recipes) are the predicates made from Triggers and Actions. For example, if you like a picture on Instagram (trigger), and IFTTT app can send the photo to your Dropbox account (action).

- Ingredients are basic data available from a trigger - from the email trigger, for example; subject, body, attachment, received date, and sender's address.

5.2.e Google Sheets

Google Sheets is a spreadsheet program included as part of the free, web-based Google Docs Editors suite offered by Google. Google Sheets is available as a web application, mobile app for Android, iOS, Microsoft Windows, BlackBerry OS, and as a desktop application on Google's Chrome OS. The app is compatible with Microsoft Excel file formats. The app allows users and APIs to create and edit files online while collaborating with other users in real-time. Edits are tracked by a user with a revision history presenting changes. An editor's position is highlighted with an editor-specific color and cursor and a permissions system regulates what users can do. Updates have introduced features using machine learning, including "Explore", offering answers based on natural language questions in a spreadsheet.

5.2.f ThingSpeak IoT Cloud

ThingSpeak is an IoT Cloud platform where users can send sensor data to the cloud. Users can also analyze and visualize the data with MATLAB or other software, including making their applications. The ThingSpeak service is operated by MathWorks. To sign up for ThingSpeak, a user must create a new MathWorks Account or log in to an existing MathWorks Account. ThingSpeak is free for small non-commercial projects. ThingSpeak includes a Web Service (REST API) that lets helps users collect and store sensor data in the cloud and develop Internet of Things applications. It works with NodeMCU and MATLAB (premade libraries and APIs exists) But it should work with all kind of Programming Languages since it uses a REST API and HTTP.

5.2.g HTTP

The Hypertext Transfer Protocol (HTTP) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access. HTTP functions as a request-response protocol in the client-server model. A web browser, for example, may be the client whereas a process, named a web server, running on a computer hosting one or more websites may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

5.3 System Design and Implementation

The major components in the design of the Internet of Things based Stadiometer are the ultrasonic distance sensor HC-SR04, NodeMCU ESP8266 Wi-Fi SoC having a Tensilica 32-bit RISC CPU Xtensa LX106 microcontroller, buzzer, red indicator LED, Wi-Fi router, power supply, 16×2 I2C liquid crystal display, stadiometer stand, Arduino IoT Cloud, IFTTT API Engine, Google Sheets and Subscriber Client.

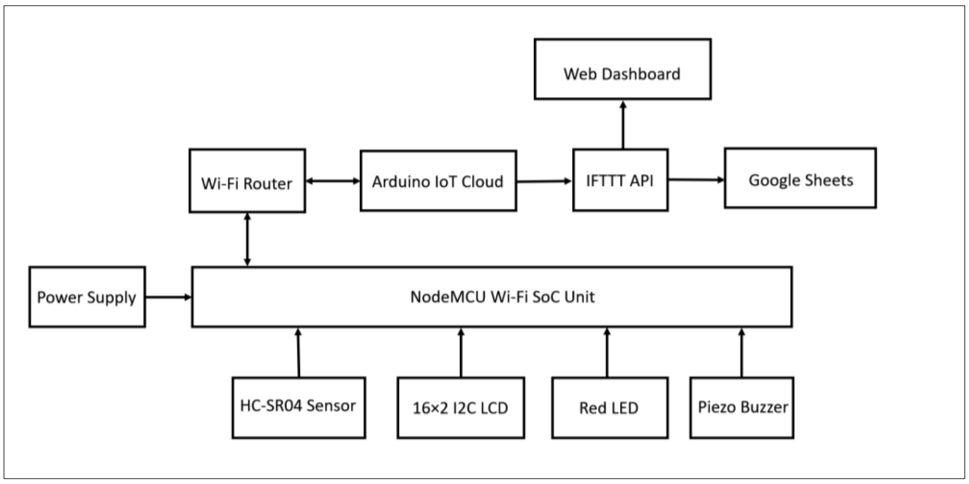


Figure 5.17: Block diagram of IoT Stadiometer

The project uses NodeMCU ESP8266 Wi-Fi SoC to connect to the Wi-Fi network and calculate the height using the ultrasonic distance sensor HC-SR04. The system is powered using a 5V 1A DC Power Supply. The Piezo Buzzer, 5mm Red LED is added to give audio-visual notifications when the data is successfully uploaded to IoT Cloud. Figure 5.17 shows the algorithm of the project.

The device works in 3 stages:

1. Once the system’s circuit is powered up, the NodeMCU ESP8266 Wi-Fi SoC searches for nearby Wi-Fi networks and tries to connect with the known Wi-Fi network, it also establishes the connection with the Arduino IoT Cloud server.

2. Once the person stands on the stadiometer stand, an ultrasonic distance sensor interfaced with NodeMCU ESP8266 Wi-Fi SoC measures the height and performs computations with the

help of an algorithm implemented on the Tensilica 32-bit RISC CPU Xtensa LX106 Microcontroller.

3. As the NodeMCU ESP8266 Wi-Fi SoC is connected to the Internet it uploads the data to Arduino IoT Cloud. The IFTTT API acts as a data path between IoT Cloud and Google Sheets. It fetches the data from Arduino IoT Cloud and uploads it to Google Sheets, for further data storage and analysis.

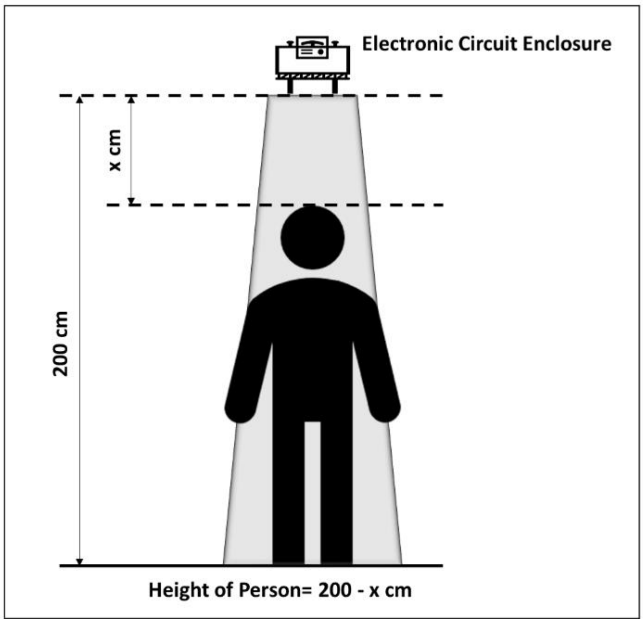


Figure 5.18: Sensor-electronics enclosure and human position below stadiometer

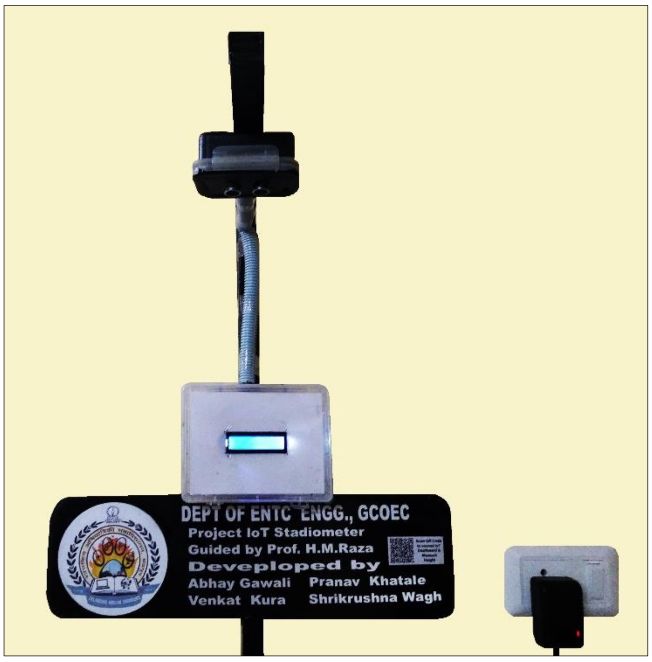


Figure 5.19: Actual implementation of IoT Stadiometer

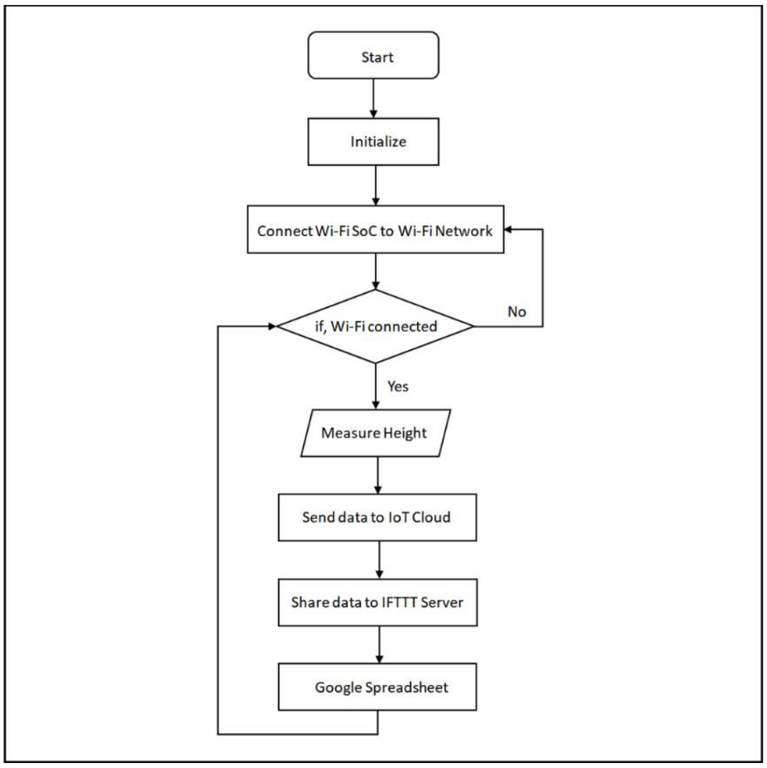


Figure 5.20: Flowchart of IoT Stadiometer

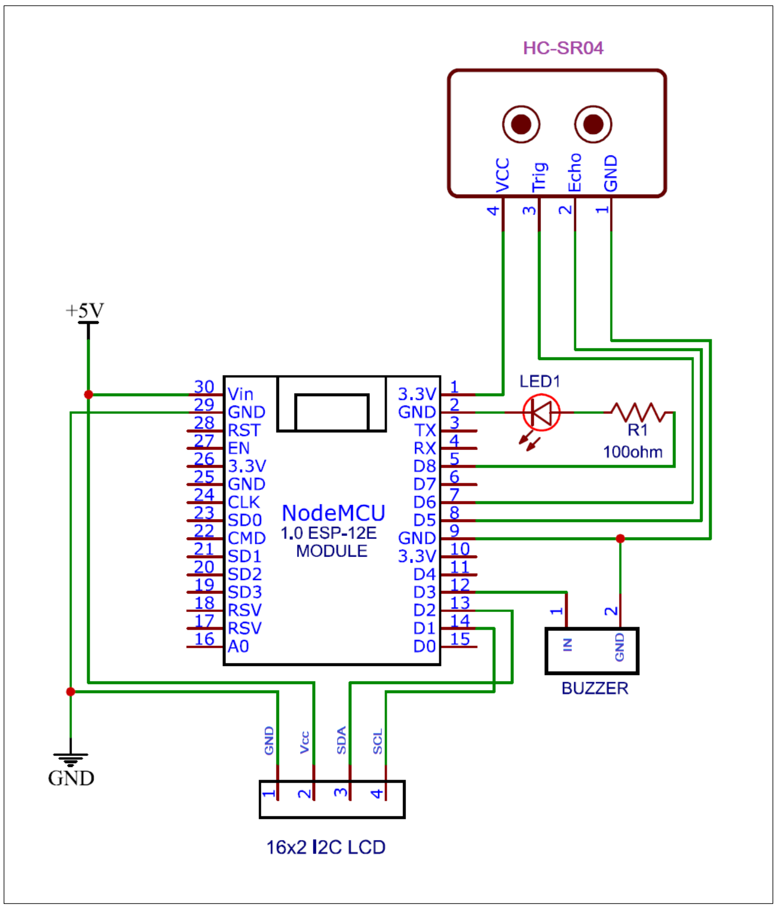


Figure 5.21: Circuit diagram of IoT Stadiometer

Figure 5.18 shows the circuit diagram of the IoT Stadiometer electronics circuit. The NodeMCU, 16×2 I2C LCD is working on an external 5V DC power supply. The HC-SR04 sensor is consuming 3.3 V from NodeMCU's 3V3 pin. The 16×2 LCD is using I2C protocol for data communication. The buzzer and red LED are deriving power from NodeMCU's GPIO pin. The resistor of 100 Ω is used to protect the LED from getting damaged due to excessive power at the GPIO pin.

5.4 System Execution in Detail

1. Users need to set up a Portable Wi-Fi Hotspot Network with the below credentials and turn on the power supply of the IoT Stadiometer -

SSID	- IoTWiFi1
Password	- IoTWiFi1
Security	- WPA2-Personal
AP Band	- 2.4 GHz

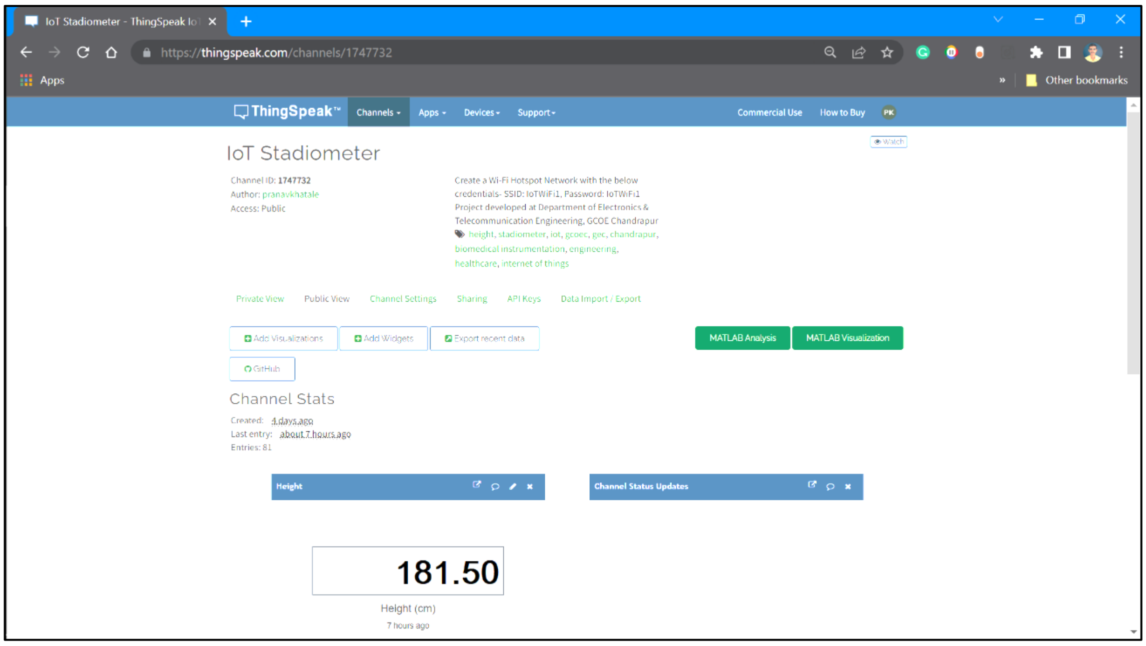
   Note: Create only 1 Wi-Fi Hotspot with the above credentials at a time.
2. Turn on the Internet connection. The IoT Stadiometer will try to connect to the Wi-Fi hotspot.
3. The user needs to scan the QR code printed on the front panel of the IoT Stadiometer or the below given, to access the ThingSpeak IoT Cloud web dashboard.

Figure 5.22: QR code for accessing the public web dashboard of ThingSpeak

4. After 30 seconds, the IoT Stadiometer will connect with Wi-Fi Hotspot.
5. Now user needs to measure the height.
6. Once a person hears a 5-second long buzzer beep, he\she must need to leave the IoT Stadiometer.
7. Now users can check the result on ThingSpeak IoT Cloud Public Web Dashboard by visiting https://thingspeak.com/channels/1747732 or by entering the channel ID as 1747732.

8. Users can check the result on Arduino IoT Cloud Web Dashboard by visiting the account dashboard opened earlier from personal login, if Arduino IoT Cloud, with IFTTT API and Google Sheets application, is deployed.
9. The “Export recent data” button can be used to download the spreadsheet file in .csv format on the ThingSpeak dashboard.

Desktop Screenshots:


https://thingspeak.com/channels/1747732

IoT Stadiometer

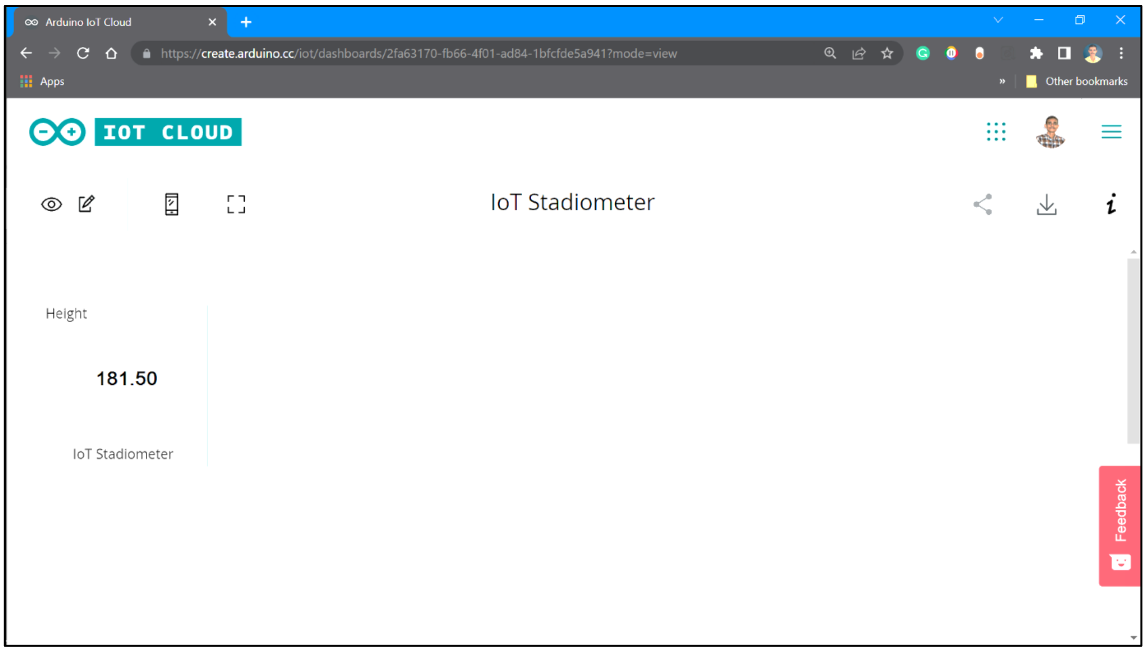
Channel ID: 1747732
Author: pranavkhatale
Access: Public

Create a Wi-Fi Hotspot Network with the below credentials- SSID: IoTWiFi1, Password: IoTWiFi1
Project developed at Department of Electronics & Telecommunication Engineering, GCOE Chandrapur
height, stadiometer, iot, gcoec, gec, chandrapur, biomedical instrumentation, engineering, healthcare, internet of things

Channel Stats
Created: 4 days ago
Last entry: about 7 hours ago
Entries: 81

Height
181.50
Height (cm)
7 hours ago

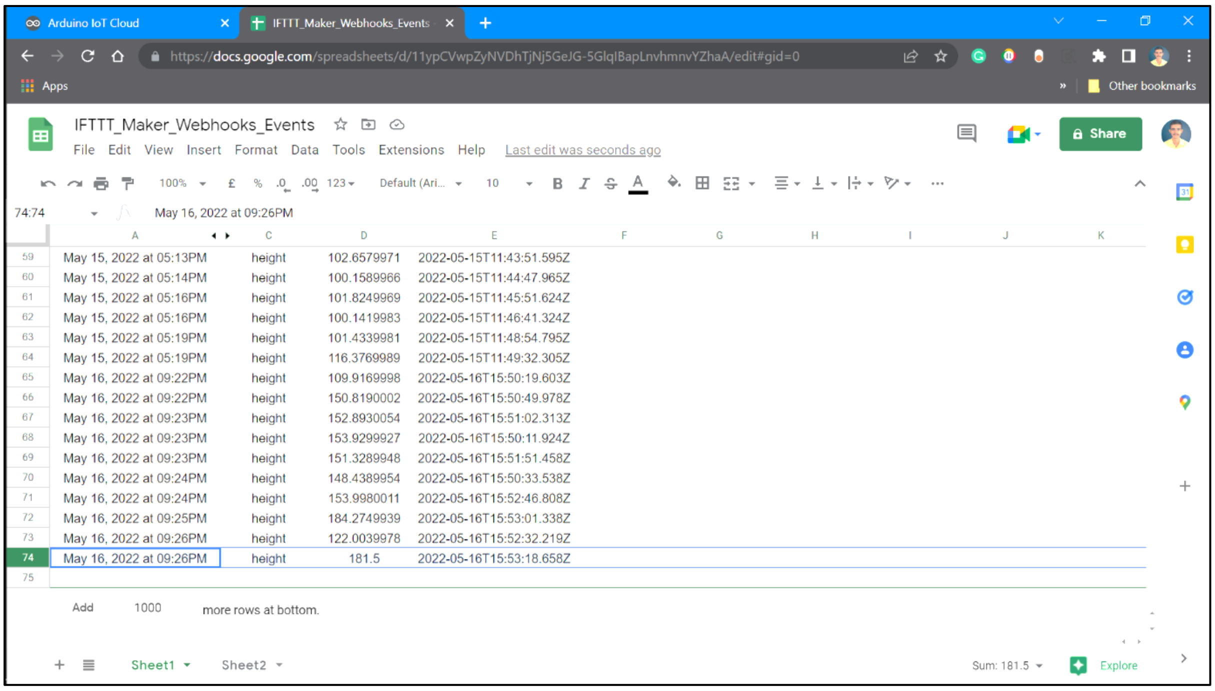
Figure 5.23: ThingSpeak IoT Cloud Public View Dashboard


https://create.arduino.cc/iot/dashboards/2fa63170-fb66-4f01-ad84-1bfcfde5a941?mode=view

IoT Stadiometer

Height
181.50
IoT Stadiometer

Figure 5.24: Arduino IoT Cloud Web Dashboard


Arduino IoT Cloud
IFTTT_Maker_Webhooks_Events
https://docs.google.com/spreadsheets/d/11ypCVwpZyNVDhTjNj5GeJG-5GlqlBapLnvhmnvYZhaA/edit#gid=0
IFTTT_Maker_Webhooks_Events
File Edit View Insert Format Data Tools Extensions Help
Last edit was seconds ago
Share
74:74
May 16, 2022 at 09:26PM
Sheet1
Sheet2
Sum: 181.5
Explore

	A	C	D	E
59	May 15, 2022 at 05:13PM	height	102.6579971	2022-05-15T11:43:51.595Z
60	May 15, 2022 at 05:14PM	height	100.1589966	2022-05-15T11:44:47.965Z
61	May 15, 2022 at 05:16PM	height	101.8249969	2022-05-15T11:45:51.624Z
62	May 15, 2022 at 05:16PM	height	100.1419983	2022-05-15T11:46:41.324Z
63	May 15, 2022 at 05:19PM	height	101.4339981	2022-05-15T11:48:54.795Z
64	May 15, 2022 at 05:19PM	height	116.3769989	2022-05-15T11:49:32.305Z
65	May 16, 2022 at 09:22PM	height	109.9169998	2022-05-16T15:50:19.603Z
66	May 16, 2022 at 09:22PM	height	150.8190002	2022-05-16T15:50:49.978Z
67	May 16, 2022 at 09:23PM	height	152.8930054	2022-05-16T15:51:02.313Z
68	May 16, 2022 at 09:23PM	height	153.9299927	2022-05-16T15:50:11.924Z
69	May 16, 2022 at 09:23PM	height	151.3289948	2022-05-16T15:51:51.458Z
70	May 16, 2022 at 09:24PM	height	148.4389954	2022-05-16T15:50:33.538Z
71	May 16, 2022 at 09:24PM	height	153.9980011	2022-05-16T15:52:46.808Z
72	May 16, 2022 at 09:25PM	height	184.2749939	2022-05-16T15:53:01.338Z
73	May 16, 2022 at 09:26PM	height	122.0039978	2022-05-16T15:52:32.219Z
74	May 16, 2022 at 09:26PM	height	181.5	2022-05-16T15:53:18.658Z
75				

Add 1000 more rows at bottom.

Figure 5.25 Google Sheets linked with Arduino IoT Cloud using IFTTT APIs

Smartphone screenshots of similar services are attached in Appendix B along with this project report.

Chapter 6

Result

The sensitivity and accuracy of the stadiometer project have been tested for height measurement and compared with the measurement made on the wall for the same.

Stadiometer Reading (X1)	Wall Reading (X2)	Differences (X1-X2)	$(X1-X2)^2$
165.1	165	0.1	0.01
167.5	167.5	0	0.00
169.2	169	0.2	0.04
171.6	171.5	0.1	0.01
173.2	173	0.2	0.04
175.6	175.5	0.1	0.01
177.1	177	0.1	0.01
179.8	179.5	0.3	0.09
181.4	181	0.4	0.16
183.7	183.5	0.2	0.04
$\sum(X1-X2)^2$ =			0.41

Figure 6.1 - Table of Stadiometer measured readings and wall readings

To determine the accuracy of the measurements noted, Root Mean Square is given by,

$RMSE = [\ \sum(X1-X2)^2 \div N\ ]^{1/2}$

$= [\ 0.41 \div 10\ ]^{1/2}$

RMSE = 0.2024

Advantages

- IoT Stadiometer operates at a faster speed (< 4 height measurements per minute)
- It uses precise contactless sensing technology to reduce human contact with a stadiometer.
- Short nurses can also use it for measuring a patient's height, during regular check-ups.
- It can be powered up with a battery or external domestic power supply.
- It consumes very low power.
- The complete Stadiometer is built using open-source hardware and software tools.
- It gives real-time quick and accurate results on LCD as well as on cloud dashboard or smartphone app.
- The price is very less, which makes it affordable to adopt quickly without heavy investment for installation.
- Calibration and maintenance aren't required.
- Useful for people with any age group except babies.
- No, another person is required to measure and record the height of a person
- It uses paperless digital technology for data storage to reduce paper and human efforts.

Applications

IoT Stadiometers can use in routine medical examinations and also clinical tests and experiments in several places such as:

- Educational institutions
- Gymnasiums
- Medical diagnostics centers
- Screening centers
- Military training centers

Chapter 7

Conclusions and Future Scope

Using a system of ultrasonic distance sensors and NodeMCU Wi-Fi SoC development platform, user himself can measure the height. The design of the IoT stadiometer system has been completed and the aim was achieved. This project work has successfully presented a functional and accurate IoT-based electronic stadiometer, with complete IoT automation, reliability, data storage in Arduino IoT Cloud, Google Sheets, ThingSpeak IoT Cloud, analysis, and even output on liquid crystal display.

Future Scope

- This stadiometer can be upgraded with better algorithms to provide person-to-person healthcare service like monitoring the height and alerting in case of imperfections such that a person can grow at healthy rate.
- The temperature around the stadiometer impacts the height measurements so a perfect algorithm and addition of few sensors like temperature and humidity sensor is required to get maximum accuracy.

BIBLIOGRAPHY

[1] Health and Care blog - https://www.healthandcare.co.uk/stadiometers/seca-264-wireless-stadiometer-patient-height-measuring-system.html

[2] Garganta, Melissa & Bremer, Andrew. (2014). Clinical Dilemmas in Evaluating the Short Child. Pediatric annals. 43. 321-327. 10.3928/00904481-20140723-11.

[3] Elflein, J. (2018, April). Amount of time U.S. primary care physicians spent with each patient as of 2018. Retrieved June 4, 2020

[4] Rabin, R. C. (2014, April 21). "15-Minute Visits Take A Toll On The Doctor-Patient Relationship". Retrieved from https://khn.org/news/15-minute-doctor-visits/

[5] Mikula, A., Hetzel, S., Binkley, N., & Anderson, P. (2016). "Clinical height measurements are unreliable: a call for improvement". Osteoporosis International, 27 (10), 3041–3047. https://doi-org.libproxy.scu.edu/10.1007/s00198-016-3635-2

[6] Shoewu, O., Duduyemi, Samuel, J. (June 2015) "Design and Development of a Microcontroller Based Stadiometer"

[7] Valerie Woo (Spring 2020), "Digital Height-Measuring Sensor Device"

[8] Tunji John Erinle1, Dayo Hephzibah Oladebeye, Ibrahim Bunmi Ademiloye (July 2020), "Parametric Design of Height and Weight Measuring System" DOI:10.17148/IJIREEICE.2020

[9] Alexander Nguyen, Michael Heath, Anthony Messina, Jiang Wu, and Ying Suna (March 2017), "An Ultrasonic Sensor Based Portable Height Measuring Device" Conference: 43rd Annual Northeast Bioengineering ConferenceAt: New Jersey Institute of Technology, Newark, NJ, USA.

[10] Minh H. Ly, Nguyen M. Khang, Tran T. Nhi, Tin T. Dang, and Anh Dinh, "A Non-contact Human Body Height and Weight Measurement Approach Using Ultrasonic Sensor" Springer Nature Singapore Pte Ltd. 2020, V. Van Toi et al. (eds.), 7th International Conference on the Development of Biomedical Engineering in Vietnam (BME7), IFMBE Proceedings 69.

APPENDIX – A Source Code

Arduino IoT Cloud specific sketch for NodeMCU:

```
/*
 : Author:   Pranav Khatale
 : Date:     01/05/2022
 : Revision: Version 1.0
 : License:  Public Domain
= Project: IoT Stadiometer
  Documentation of this project visit:   https://github.com/pranavkhatale/IoT-Stadiometer,
  sketch generated by the Arduino IoT Cloud Thing "IoT Stadiometer"
  https://create.arduino.cc/cloud/things/9245b08b-ff7d-4a33-ad2f-b215554b0e51
  Arduino IoT Cloud Variables description the following variables are automatically
generated and updated when changes are made to the Thing
  CloudLength height;
  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
which are called when their values are changed from the Dashboard. These functions are
generated with the Thing and added at the end of this sketch.

*/
// including libraries
#include "thingProperties.h"
#include <LiquidCrystal_I2C.h>
// NodeMCU and HC-SR04 circuit connections
const int trigPin = 12;                 // Pin D6 or GPIO 12
const int echoPin = 14;                 // Pin D5 OR GPIO 14
#define SOUND_VELOCITY 0.034            // define sound velocity in cm/uS
int t = 0;                              // Wi-Fi connection successful timer variable
int tc = 0;                             // Wi-Fi connection failure timer variable
float rawheight;
float heightraw;
long duration;
float distanceCm;
float distanceInch;
int lcdColumns = 16;                    // set the LCD number of columns
int lcdRows = 2;                        // set the LCD number of rows
// set LCD address, number of columns and rows
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
//Below loop executes once
void setup() {
  Serial.begin(115200);                 // Initialize serial and wait for port to open:
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);
  initProperties();                              // Defined in thingProperties.h
  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  /* The following function allows you to obtain more information related to the state of
network and IoT Cloud connection and errors the higher number the more granular
information you’ll get. The default is 0 (only errors). Maximum is 4 */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
  lcd.init();                                  // initialize LCD
  lcd.backlight();                             // turn on LCD backlight
  pinMode(trigPin, OUTPUT);                    // Sets the trigPin as an Output
  pinMode(echoPin, INPUT);                     // Sets the echoPin as an Input
  pinMode(D3, OUTPUT);                         // Buzzer is connected to D3 Pin
  pinMode(D4, OUTPUT);                         // Onboard Blue LED is declared as output device
  pinMode(D8, OUTPUT);                         // Red LED is connected to D8 Pin
  lcd.clear();                                 // Clear the contents on LCD
  lcd.setCursor(0, 0);                         // Sets the cursor on column 1, row 1
  lcd.print("      IoT");                      // Print message
  lcd.setCursor(0, 1);
  lcd.print("  Stadiometer");                  // Sets the cursor on column 1, row 2
  Serial.println("IoT Stadiometer");
  delay(2000);                                 // Delay of 2 seconds
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(" Connecting to");
  lcd.setCursor(0, 1);
  lcd.print("WiFi & IoT Cloud");
  delay(2000);
  lcd.clear();
}
//Below loop executes multiple times
void loop() {
  ArduinoCloud.update();                         // Gets update of the Arduino IoT Cloud
  // Executes the loop if NodeMCU is not connected to Wi-Fi and Arduino IoT Cloud
  if (ArduinoCloud.connected() == 0)
  {
    ArduinoCloud.update();
    Serial.println("Waiting for connection to Arduino IoT Cloud");
    lcd.setCursor(0, 0);
    lcd.print("...");
    lcd.setCursor(0, 1);
    lcd.print("             ...");
    delay(1000);
    lcd.clear();
    tc = tc + 1;                         // Incrementing the time span by 1 seconds in past value
    // Executes if NodeMCU fails to connect to Wi-Fi router in 15 seconds
    for (; tc >= 15;)
    {
      lcd.setCursor(0, 0);
      lcd.print("   Networking");
      lcd.setCursor(0, 1);
      lcd.print("     Issues");
      digitalWrite(D3, HIGH);            // turn on buzzer for 5 seconds
      delay(5000);
      lcd.clear();
      digitalWrite(D3, LOW);             // turn off buzzer for 5 seconds
      tc = 0;
      break;
    }
  }
 // Executes the loop if NodeMCU is connected to both Wi-Fi and Arduino IoT Cloud
  else
  {
    // Executes the below loop if NodeMCU establishes the connection to Cloud
    if (t == 0)
    {
      t = t + 1;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("  Connection");
      lcd.setCursor(0, 1);
      lcd.print("  Established");
      digitalWrite(D8, HIGH);
     // Red LED blinks and Buzzers beeps 2 times to indicate that everything is going     well!
      digitalWrite(D3, HIGH);   // turn on buzzer for half seconds
      digitalWrite(D4, HIGH);   // turn on onboard Blue LED for half seconds
      digitalWrite(D8, HIGH);   // turn on Red LED for half seconds
      delay(500);
      digitalWrite(D3, LOW);    // turn off buzzer for half seconds
      digitalWrite(D4, LOW);    // turn off onboard Blue LED for half seconds
      digitalWrite(D8, LOW);    // turn off Red LED for half seconds
      delay(500);
      digitalWrite(D3, HIGH);   // turn on buzzer for half seconds
      digitalWrite(D4, HIGH);   // turn on onboard Blue LED for half seconds
      digitalWrite(D8, HIGH);   // turn on Red LED for half seconds
      delay(500);
      digitalWrite(D3, LOW);    // turn off buzzer for half seconds
      digitalWrite(D4, LOW);    // turn off onboard Blue LED for half seconds
      digitalWrite(D8, LOW);    // turn off Red LED for half seconds
      delay(500);
      lcd.clear();
    }
    // Start printing the height value on LCD
    lcd.setCursor(0, 0);
    lcd.print("    Reading");
    lcd.setCursor(0, 1);
    lcd.print("    Height");
    delay(500);
    lcd.clear();
    // Clears the trigger Pin
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    // Sets the trigger pin on HIGH state for 10 micro seconds
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echo pin, returns the sound wave travel time in microseconds
    duration = pulseIn(echoPin, HIGH);
    // Calculate the distance in cm
    distanceCm = duration * SOUND_VELOCITY / 2;
    // Prints the distance on the Serial Monitor
    Serial.print("Distance (cm): ");
    Serial.println(distanceCm);
    // Calculate the height of person by subtracting the measured distance from 200 cm
    rawheight = 200 - distanceCm;
    //Serial.print("rawheight (cm): ");
    //Serial.println(rawheight);
    // If the value is positive then execute the loop
    if (rawheight >= 100)
    {
      // Read the sensor value to detect if the person left the stadiometer or is present there to
  measure the height
      delay(5000);
      // Clears the trigPin
      digitalWrite(trigPin, LOW);
      delayMicroseconds(2);
      // Sets the trigPin on HIGH state for 10 micro seconds
      digitalWrite(trigPin, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin, LOW);
      // Reads the echoPin, returns the sound wave travel time in microseconds
```

```
      duration = pulseIn(echoPin, HIGH);
      // Calculate the distance
      distanceCm = duration * SOUND_VELOCITY / 2;
      // Prints the distance on the Serial Monitor
      Serial.print("Distance (cm): ");
      Serial.println(distanceCm);
      heightraw = 200 - distanceCm;
      if (heightraw >= 100)
      {
        height = heightraw;
        // Serial.print("rawheight (cm): ");
        // Serial.println(rawheight);
        // Prints the distance on the Serial Monitor
        Serial.print("Height (cm): ");
        Serial.println(height);
        Serial.println("Data Uploaded Sucessfully.");
        Serial.println("                              ");
        lcd.clear();
        lcd.setCursor(0, 0);     // set cursor to first column, first row
        lcd.print("Height:");
        lcd.print(height);                          // print height value on LCD
        lcd.print(" cm");
        lcd.setCursor(0, 1);
        lcd.print(" Data Uploaded");
        digitalWrite(D3, HIGH);              // turn on buzzer
        digitalWrite(D8, HIGH);              // turn on Red LED
        ArduinoCloud.update();              // upload the measured height data to cloud
        delay(5000);
        lcd.clear();                                  // clears the display to print new message
        digitalWrite(D3, LOW);              // turn off buzzer
        digitalWrite(D8, LOW);              // turn off Red LED
      }
    }
  }
}
```

Souce code in “arduino_secrets.h” file:

```
#define SECRET_SSID "IoTWiFi1"
#define SECRET_PASS "IoTWiFi1"
// Enter the device specific secret key from the Arduino IoT Cloud login dashboard
#define SECRET_DEVICE_KEY ""
```

Source code in “thingProperties.h” file:

```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>
// Enter the device login name
const char DEVICE_LOGIN_NAME[] = "DEVICE_LOGIN_NAME";
const char SSID[] = SECRET_SSID;          // Network SSID (name)
// Network password (use for WPA, or use as key for WEP)
const char PASS[] = SECRET_PASS;
const char DEVICE_KEY[] = SECRET_DEVICE_KEY;          // Secret device password
CloudLength height;
void initProperties(){
  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(height, READ, ON_CHANGE, NULL, 1);
}
WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

ThingSpeak IoT Cloud specific sketch for NodeMCU:

```
/*
 : Author:   Pranav Khatale
 : Date:     01/05/2022
 : Revision: Version 1.0
 : License:  Public Domain
= Project: IoT Stadiometer
  Documentation of this project visit: https://github.com/pranavkhatale/IoT-Stadiometer
*/
#include <ESP8266WiFi.h>
#include <LiquidCrystal_I2C.h>
int t = 0;
int cloudtimer=11;
int i;
float rawheight;
float heightraw;
float height;
// NodeMCU ESP8266 circuit connections with HC-SR04
const int trigPin = 12; // Pin D6
const int echoPin = 14; // Pin D5
//define sound velocity in cm/uS
#define SOUND_VELOCITY 0.034
```

```
long duration;
float distanceCm;
// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;
// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
String apiKey = "";     //  Enter your Write API key from ThingSpeak IoT Analytics Cloud
platform.
const char *ssid =  "IoTWiFi1";     // Replace with your Wi-Fi ssid.
const char *pass =  "IoTWiFi1";     // Replace with your Wi-Fi wpa2 key.
const char *server = "api.thingspeak.com"; //Initializing Server.
WiFiClient client;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT);  // Sets the echoPin as an Input
  pinMode(D3, OUTPUT);      //Buzzer is declared as output
  pinMode(D4, OUTPUT);      //Onboard Blue LED is declared as output
  pinMode(D8, OUTPUT);      //Red LED is declared as output
  // Program written in setup will run only once for connecting to Wi-Fi network.
  Serial.begin(115200);     // Selecting the baud rate for Serial Communication.
  delay(500);
  Serial.println("IoT Stadiometer");
  lcd.init();                    // initialize LCD
  lcd.backlight();               // turn on LCD backlight
  lcd.setCursor(0, 0);
  lcd.print("      IoT");
  lcd.setCursor(0, 1);
  lcd.print("  Stadiometer");
  delay(2000);
  lcd.clear();
  Serial.println("Connecting to ");
  Serial.println(ssid);
  lcd.setCursor(0, 0);
  lcd.print("Connecting to");
  lcd.setCursor(0, 1);
  lcd.print("Wi-Fi Network...");
  delay(2000);
  lcd.clear();
  WiFi.begin(ssid, pass); // Wi-Fi network searching starts.
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
    }
  Serial.println("");
  Serial.println("Wi-Fi connected."); //Prints on serial monitor if Wi-Fi gets connected
sucessfully.
  lcd.setCursor(0, 0);
  lcd.print("Connected to");
  lcd.setCursor(0, 1);
  lcd.print("IoT WiFi Network");
  delay(2000);
  lcd.clear();
}
void loop()
{
  if (client.connect(server,80))   // "184.106.153.149" or api.thingspeak.com
  {
      if(t<1)
      {
        digitalWrite(D3, HIGH);   // turn on buzzer
        digitalWrite(D4, HIGH);   // turn on onboard Blue LED
        digitalWrite(D8, HIGH);   // turn on Red LED
        delay(500);
        digitalWrite(D3, LOW);    // turn off buzzer
        digitalWrite(D4, LOW);    // turn off onboard Blue LED
        digitalWrite(D8, LOW);    // turn off Red LED
        delay(500);
        digitalWrite(D3, HIGH);   // turn on buzzer
        digitalWrite(D4, HIGH);   // turn on onboard Blue LED
        digitalWrite(D8, HIGH);   // turn on Red LED
        delay(500);
        digitalWrite(D3, LOW);    // turn off buzzer
        digitalWrite(D4, LOW);    // turn off onboard Blue LED
        digitalWrite(D8, LOW);    // turn off Red LED
        delay(500);
        t=t+1;
      }
    lcd.setCursor(0, 0);
    lcd.print("    Reading");
    lcd.setCursor(0, 1);
    lcd.print("    Height");
    delay(500);
    lcd.clear();
    // Clears the trigPin
    digitalWrite(trigPin, LOW);
```

```
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
// Calculate the distance
distanceCm = duration * SOUND_VELOCITY / 2;
// Prints the distance on the Serial Monitor
Serial.print("Distance (cm): ");
Serial.println(distanceCm);
rawheight = 200 - distanceCm;
if (rawheight >= 100)
{
  delay(5000);
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculate the distance
  distanceCm = duration * SOUND_VELOCITY / 2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance (cm): ");
  Serial.println(distanceCm);
  heightraw = 200 - distanceCm;
  if (heightraw >= 100)
  {
      height = heightraw;
      String postStr = apiKey;
      postStr +="&field1=";
      postStr += String(height);
      postStr += "\r\n\r\n";
      client.print("POST /update HTTP/1.1\n");
      client.print("Host: api.thingspeak.com\n");
      client.print("Connection: close\n");
      client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
      client.print("Content-Type: application/x-www-form-urlencoded\n");
      client.print("Content-Length: ");
```

```
        client.print(postStr.length());
        client.print("\n\n");
        client.print(postStr);
        Serial.println("Data sent to Thingspeak IoT Analytics Cloud platform sucessfully.");
        Serial.print("Height (cm): ");          // Prints the distance on the Serial Monitor
        Serial.println(height);
        Serial.println("Data Uploaded Sucessfully.");
        Serial.println("                              ");
        lcd.setCursor(0, 0);                    // set cursor to first column, first row
        lcd.print("Height:");                   // print message
        lcd.print(height);
        lcd.print(" cm");
        lcd.setCursor(0, 1);
        lcd.print(" Data Uploaded");
        digitalWrite(D3, HIGH);                  // turn on buzzer
        digitalWrite(D8, HIGH);                  // turn on Red LED
        delay(5000);
        lcd.clear();                             // clears the display to print new message
        digitalWrite(D3, LOW);                   // turn off buzzer
        digitalWrite(D8, LOW);                  // turn off Red LED
        for(i=6;i--;i<=0)
        {
              lcd.setCursor(0,0);
              lcd.print("Wait for");
              lcd.setCursor(0,1);
              lcd.print(i);
              lcd.print(" seconds");
              delay(1000);
              lcd.clear();
        }
      }
    }
  }
  client.stop();
}
```

Scan the QR code to access all the above source codes or sketches alongwith required header from GitHub repository: https://github.com/pranavkhatale/IoT-Stadiometer

APPENDIX-B Result Screenshots of Smartphone Applications

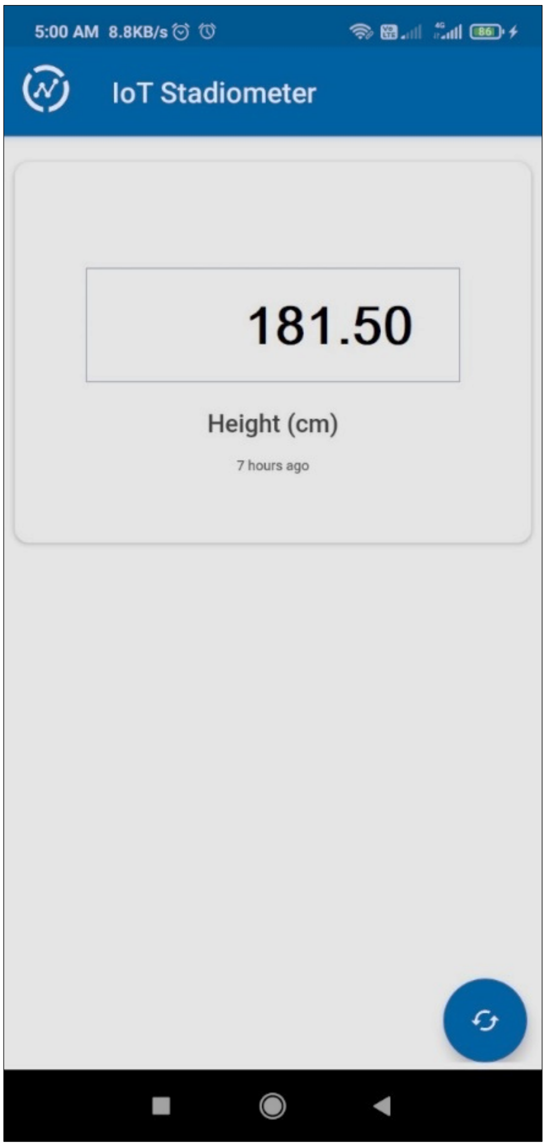


Figure 5.23: ThingSpeak IoT Cloud dashboard on ThingView Smartphone App

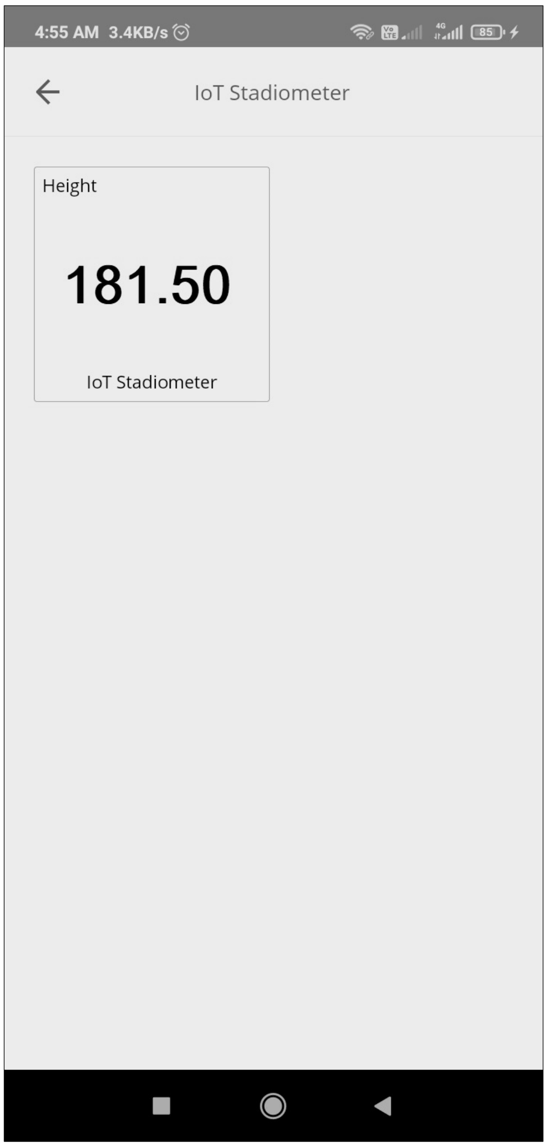
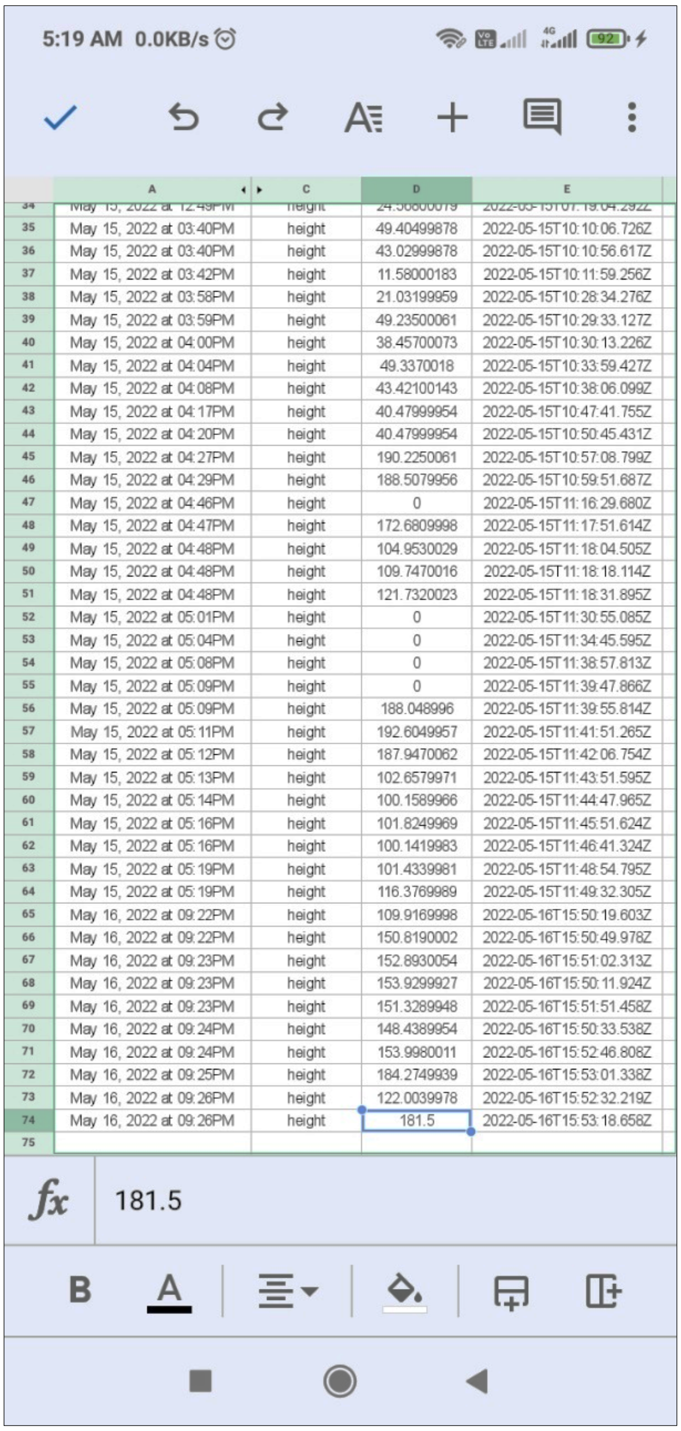


Figure 5.24: Arduino IoT Cloud Dashboard accessed on IoT Remote Smartphone App



	A	C	D	E
34	May 15, 2022 at 12:49PM	height	[illegible]	2022-05-15T07:19:04.292Z
35	May 15, 2022 at 03:40PM	height	49.40499878	2022-05-15T10:10:06.726Z
36	May 15, 2022 at 03:40PM	height	43.02999878	2022-05-15T10:10:56.617Z
37	May 15, 2022 at 03:42PM	height	11.58000183	2022-05-15T10:11:59.256Z
38	May 15, 2022 at 03:58PM	height	21.03199959	2022-05-15T10:28:34.276Z
39	May 15, 2022 at 03:59PM	height	49.23500061	2022-05-15T10:29:33.127Z
40	May 15, 2022 at 04:00PM	height	38.45700073	2022-05-15T10:30:13.226Z
41	May 15, 2022 at 04:04PM	height	49.3370018	2022-05-15T10:33:59.427Z
42	May 15, 2022 at 04:08PM	height	43.42100143	2022-05-15T10:38:06.099Z
43	May 15, 2022 at 04:17PM	height	40.47999954	2022-05-15T10:47:41.755Z
44	May 15, 2022 at 04:20PM	height	40.47999954	2022-05-15T10:50:45.431Z
45	May 15, 2022 at 04:27PM	height	190.2250061	2022-05-15T10:57:08.799Z
46	May 15, 2022 at 04:29PM	height	188.5079956	2022-05-15T10:59:51.687Z
47	May 15, 2022 at 04:46PM	height	0	2022-05-15T11:16:29.680Z
48	May 15, 2022 at 04:47PM	height	172.6809998	2022-05-15T11:17:51.614Z
49	May 15, 2022 at 04:48PM	height	104.9530029	2022-05-15T11:18:04.505Z
50	May 15, 2022 at 04:48PM	height	109.7470016	2022-05-15T11:18:18.114Z
51	May 15, 2022 at 04:48PM	height	121.7320023	2022-05-15T11:18:31.895Z
52	May 15, 2022 at 05:01PM	height	0	2022-05-15T11:30:55.085Z
53	May 15, 2022 at 05:04PM	height	0	2022-05-15T11:34:45.595Z
54	May 15, 2022 at 05:08PM	height	0	2022-05-15T11:38:57.813Z
55	May 15, 2022 at 05:09PM	height	0	2022-05-15T11:39:47.866Z
56	May 15, 2022 at 05:09PM	height	188.048996	2022-05-15T11:39:55.814Z
57	May 15, 2022 at 05:11PM	height	192.6049957	2022-05-15T11:41:51.265Z
58	May 15, 2022 at 05:12PM	height	187.9470062	2022-05-15T11:42:06.754Z
59	May 15, 2022 at 05:13PM	height	102.6579971	2022-05-15T11:43:51.595Z
60	May 15, 2022 at 05:14PM	height	100.1589966	2022-05-15T11:44:47.965Z
61	May 15, 2022 at 05:16PM	height	101.8249969	2022-05-15T11:45:51.624Z
62	May 15, 2022 at 05:16PM	height	100.1419983	2022-05-15T11:46:41.324Z
63	May 15, 2022 at 05:19PM	height	101.4339981	2022-05-15T11:48:54.795Z
64	May 15, 2022 at 05:19PM	height	116.3769989	2022-05-15T11:49:32.305Z
65	May 16, 2022 at 09:22PM	height	109.9169998	2022-05-16T15:50:19.603Z
66	May 16, 2022 at 09:22PM	height	150.8190002	2022-05-16T15:50:49.978Z
67	May 16, 2022 at 09:23PM	height	152.8930054	2022-05-16T15:51:02.313Z
68	May 16, 2022 at 09:23PM	height	153.9299927	2022-05-16T15:50:11.924Z
69	May 16, 2022 at 09:23PM	height	151.3289948	2022-05-16T15:51:51.458Z
70	May 16, 2022 at 09:24PM	height	148.4389954	2022-05-16T15:50:33.538Z
71	May 16, 2022 at 09:24PM	height	153.9980011	2022-05-16T15:52:46.808Z
72	May 16, 2022 at 09:25PM	height	184.2749939	2022-05-16T15:53:01.338Z
73	May 16, 2022 at 09:26PM	height	122.0039978	2022-05-16T15:52:32.219Z
74	May 16, 2022 at 09:26PM	height	181.5	2022-05-16T15:53:18.658Z
75				

fx 181.5

Figure 5.25. Google Sheets linked with Arduino IoT Cloud using IFTTT APIs