A Major Project Report on

REMOTE MONITORING OF RADIATION AND ENVIRONMENTAL CONDITIONS WITH TELEGRAM RESPONDER BOT

Submitted in partial fulfillment of the requirements

for the award of degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

SUBMITTED

BY

AKSHAY K. RAJPUROHIT	**(17P61A0405)**
PODDUTOORU RAKESH	**(17C21A0419)**
BITTU NIKHIL	**(17P61A0425)**
PILLARISETTY SAI CHARAN	**(18P65A0419)**

Under the Esteemed Guidance of

Ms. P. SREEVANI

ASSISTANT PROFESSOR

Department of Electronics and Communication Engineering

VIGNANA BHARATHI Institute of Technology

AUTONOMOUS Under UGC

Accredited by NBA NATIONAL BOARD OF ACCREDITATION (ECE,EEE,CSE,IT,CE,ME) NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL A Grade

Counselling Code VBIT

(Affiliated to JNTU Hyderabad, Approved by APSCHE &AICTE)

Aushapur (v), Ghatkesar (m), Medchal Dist, Hyderabad-501301

2020- 2021

VIGNANA BHARATHI Institute of Technology

AUTONOMOUS Under UGC

Accredited by NBA NATIONAL BOARD OF ACCREDITATION (ECE,EEE,CSE,IT,CE,ME) NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL A Grade

Counselling Code VBIT

Department of Electronics & Communication Engineering

CERTIFICATE

This is to certify that the Major Project Report, "**REMOTE MONITORING OF RADIATION AND ENVIRONMENTAL CONDITIONS WITH TELEGRAM RESPONDER BOT**" being submitted by **AKSHAY K. RAJPUROHIT (17P61A0405), P. RAKESH (17C21A0419), B. NIKHIL (17P61A0425), P. SAICHARAN (18P65A0419)** in partial fulfillment for the award of the Degree of Bachelor of Technology in **ELECTRONICS & COMMUNICATION ENGINEERING** to Jawaharlal Nehru Technological University is a record of a bona fide work carried out by them under my guidance and supervision.

The result(s) embodied in this project report have not been submitted to any other University/Institution for the award of any Degree/Diploma.

Internal Guide

Ms. P. Sreevani
Assistant Professor

Head of the Department

Dr. U. Poorna Lakshmi

EXTERNAL EXAMINER

Department of Electronics & Communication Engineering

CANDIDATES DECLARATION

We here by declare that this Major Project report titled **"REMOTE MONITORING OF RADIATION AND ENVIRONMENTAL CONDITIONS WITH TELEGRAM RESPONDER BOT"** submitted by us to the Department of **Electronics and Communication Engineering,** VBIT, Aushapur, Under JNTUH, is a bonafide work undertaken by and it is not submitted to any other University or Institution for the award of any degree or diploma.

By

AKSHAY K. RAJPUROHIT	**(17P61A0405)**
PODDUTOORU RAKESH	**(17C21A0419)**
BITTU NIKHIL	**(17P61A0425)**
PILLARISETTY SAI CHARAN	**(18P65A0419)**

ACKNOWLEDGEMENT

At the outset we sincerely thank God for having got our Major project report completed in time.

Firstly, we would thank our parents who have been a motivating factor throughout our lives.

Secondly, we sincerely thank our principal, **Dr. P.V.S. SRINIVAS** and our Head of the department, **Dr. U. POORNA LAKSHMI** for their kind cooperation and Encouragement for the successful completion of Seminar work and providing the necessary facilities. We are most obliged and grateful to our project guide, **MS. P. SREEVANI**, for giving us guidance in completing this project successfully.

We express our sincere gratitude to our Project coordinators, Department of ECE and my other faculty for attending my project seminars and for their insightful comments and constructive suggestions to improve the quality of this project work.

By

AKSHAY K. RAJPUROHIT **(17P61A0405)**

PODDUTOORU RAKESH **(17C21A0419)**

BITTU NIKHIL **(17P61A0425)**

PILLARISETTY SAI CHARAN **(18P65A0419)**

ABSTRACT

Nowadays, the need to use radioactive sources has been increasing in every country and in India too; therefore, radiation safety systems, actuate and sensitive detection devices in several institutes, research laboratories, personal radiation monitoring and field of radiation protection essentially need radiation detection instruments. Among the radiation monitoring devices, Geiger Müller counters generally called the Geiger counters are a class of radiation detectors which are based on the phenomenon of ionization.

This project proposes to use low-cost hardware to build a monitor that measures Radiation levels and Environmental conditions (humidity and temperature) and relays the measurement readings to an endpoint on the internet, which would enable concerned parties to remotely keep a check on the system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Definition
DHT	Digital Humidity and Temperature
VPS	Virtual Private Server
SDK	Software Development Kit
SMS	Short Message Service
HDMI	High-Definition Multimedia Interface
USB	Universal Serial Bus
ARM	Advanced RISC Machine
CPU	Central Processing Unit
GPU	Graphical Processing Unit
RAM	Random Access Memory
I/O	Input/Output
GPIO	General Purpose Input/Output
GIC	General Interrupt Controller
LAN	Local Area Network
NTP	Network Time Protocol
SMPS	Switched-Mode Power Supply
SSH	Secure Shell
FTP	File Transfer Protocol
TCP	Transmission Control Protocol

CHAPTER 1

INTRODUCTION

1.1. OVERVIEW OF PROJECT

A Geiger counter (Geiger-Muller tube) is a device used for the detection and measurement of all types of radiation: alpha, beta and gamma radiation. Basically, it consists of a pair of electrodes surrounded by a gas. The electrodes have a high voltage across them. The gas used is usually Helium or Argon. When radiation enters the tube, it can ionize the gas. The ions (and electrons) are attracted to the electrodes and an electric current is produced. A scaler counts the current pulses, and one obtains a "count" whenever radiation ionizes the gas.

Nowadays, the need to use radioactive sources has been increasing in every country and in India too; therefore, radiation safety systems, actuate and sensitive detection devices in several institutes, research laboratories, personal radiation monitoring and field of radiation protection essentially need radiation detection instruments. Along with the measurement of radiation, we will also be measuring Temperature and Pressure using a simple DHT sensor module to provide more information about the environment our project is deployed in. Our project aims to be an inexpensive and easy-to-deploy solution for this unique problem.

1.2. MOTIVATION

We are living in the era of Digital Revolution, which is making dramatic changes to our lives every day. Leveraging these technologies can provide us with the means to take care of our health and the health of all those around us. The effects of radiation may not appear all at once. They can manifest themselves in decades to come in future generations, in the form of cancer, genetic mutations, etc. This is an issue that is often overlooked and has disastrous consequences.

There are many jobs that exist and involve the handling and use of radioactive materials. Often, the people who work these jobs are not aware that they are around radioactive materials and are exposed to the harmful radiations that these materials are always emitting. At once, they may be exposed to small and minute quantities of radiation, but over time, this builds up and leads to major health issues.

The various places where a Geiger Counter can be used is as follows:

- To detect radioactive rocks and minerals during mineral prospecting or as a mineral collector.
- For Fire and Police first responders to a scene for making an initial determination of radiation risk
- For HazMat personnel in checking for radiation danger in an emergency.

- To check for environmental levels of radioactivity near a nuclear power facility.
- To test for danger amidst a nuclear accident or leakage of radioactive coolant.
- To check for radioactive contamination of clothing and shoes in your workplace.
- You work in or near an X-ray lab in a medical facility and want to check for leaks or possible exposure.
- To check for irradiated gemstones in the jewellery trade.
- You are a cancer patient undergoing radiation therapy, including thyroid cancer patients checking Iodine 131 levels.
- You are near a uranium mine and want to test the soil and environment for dangerous levels of radioactivity.
- To test for radioactive contamination of food.
- To check materials in your anthropology or archaeology field.
- To check for radioactivity in metal objects in your home or office that could be made of recycled radioactive materials.

1.3 OBJECTIVES

The main objective of the project is to make radiation measurement easy and accessible to all. The Geiger counters in the market are expensive and are usually unavailable for common citizens. We can build a Geiger Counter from scratch, and this implementation will be relatively inexpensive, and can act as a good monitoring station that alerts us of any radiation present in our surroundings. People who work with hazardous materials can keep themselves safe by using our implementation.

- To monitor and measure the radiation levels and environmental conditions (temperature and humidity) and graphically represent the readings.
- To set up a Telegram Responder Bot for the authorities to interact with and to receive periodical updates about the radiation levels and environmental conditions and alert notifications if the readings cross predefined thresholds.
- To relay all the information (measurements) to an endpoint on the internet for remote monitoring and analysis.

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

Remote data-monitoring systems have been widely used in nearly all manufacturing industries for many years and with several benefits. For example, these systems allow for the tracking, recording, and ongoing monitoring of data from different sensors — such as for Radiation, Temperature, Pressure, etc. — in a central control room of a plant.

Various sensors are placed within one or more of such devices and their readings (meaning the values of the temperature, humidity, or pressure, etc.) are recorded.

These sensor readings (the data) are sent to a remote monitoring station in one of two ways: Either through wires or wirelessly. In some cases, a wired system is suitable and in others, a wireless one is more convenient. Both options have pros and cons, depending on the application. In a wireless system, a transmitter is attached to a sensor that is capable of transmitting sensor data. A remote receiver gets the data and can display and/or store it for later processing.

In this project, we are going to be using a Raspberry Pi to read the sensor values and transmit them to an endpoint on the internet, which in our case is InfluxDB deployed on a Microsoft Azure VPS cloud instance.

On the same VPS, we will also be deploying a Telegram Bot that reads values from the InfluxDB and either sends alert messages when the sensor readings cross a threshold or sends on-demand sensor readings to the user if he interacts with the Bot on the client side.

As far as sensors go, we will be using the CAJOE Radiation Monitor to measure radiation (in μSv/h) and the DHT11 sensor to measure Temperature (in °C) and Humidity (in %). And finally, we will be using the Skyhook Precision Location SDK to determine the location of the system without using a physical GPS module.

IoT devices produce many types of information, including telemetry, metadata, state, and commands and responses. Telemetry data from devices can be used in short operational timeframes or for longer-term Many devices support local monitoring in the form of a buzzer or an alarm panel on-premises. This type of monitoring is valuable, but has limited scope for in-depth or long-term analysis. This article instead discusses remote monitoring, which involves gathering and analyzing monitoring information from a remote location using cloud resources.analytics and model building.

Operational and device performance data is often in the form of a time series, where each piece of information includes a time stamp. This data can be further enriched with dimensional labels (sometimes referred to as tags), such as labels that identify hardware revision, operating timezone, installation location, firmware version, and so on.

2.2 EXISTING SYSTEM

There are several Geiger Counters or Dosimeter available in the market. These devices enable professionals to measure radiations in different places. Geiger counters are used to detect radioactive emissions, most commonly beta particles and gamma rays. The counter consists of a tube filled with an inert gas that becomes conductive of electricity when it is impacted by a high- energy particle. When a Geiger counter is exposed to ionizing radiation, the particles penetrate the tube and collide with the gas, releasing more electrons. Positive ions exit the tube and the negatively charged electrons become attracted to a high-voltage middle wire.

When the number of electrons that build up around the wire reaches a threshold, it creates an electric current. This causes the temporary closing of a switch and generates an electric pulse that is registered on a meter, either acoustically as a click that increases in intensity as the ionizing radiation increases, or visually as the motion of a needle pointer.

There are two types of detected radiation readout: counts or radiation dose. The counts display is the simplest and is the number of ionizing events detected displayed either as a count rate, such as "counts per minute" or "counts per second", or as a total number of counts over a set time period (an integrated total). The counts readout is normally used when alpha or beta particles are being detected.

More complex to achieve is a display of radiation dose rate, displayed in a unit such as the sievert which is normally used for measuring gamma or X-ray dose rates. A Geiger– Müller tube can detect the presence of radiation, but not its energy, which influences the radiation's ionizing effect. Consequently, instruments measuring dose rate require the use of an energy compensated Geiger–Müller tube, so that the dose displayed relates to the counts detected. The electronics will apply known factors to make this conversion, which is specific to each instrument and is determined by design and calibration.

The first historical uses of the Geiger principle were for the detection of alpha and beta particles, and the instrument is still used for this purpose today. For alpha particles and low energy beta particles, the "end-window" type of a Geiger–Müller tube has to be used as these particles have a limited range and are easily stopped by a solid material. Therefore, the tube requires a window which is thin enough to allow as many as possible of these particles through to the fill gas. The window is usually made of mica with a density of about 1.5 - 2.0 mg/cm^2. End-window Geiger counters are still used as a general purpose, portable, radioactive contamination measurement and detection instrument, owing to their relatively low cost, robustness and their relatively high detection efficiency, particularly with high energy beta particles.

2.3 DISADVANTAGE OF EXISTING SYSTEM

The existing system is not accessible to everyone and is generally expensive for people who are not professionals. People like miners and farmers need a radiation measurement device just as much as professionals do. The devices are also bulky, non-extensible and difficult to repair. We cannot add more features or hardware and extend it to make it more useful, and the non-open-source nature of the hardware is a roadblock for anyone who wishes to repair a faulty equipment. The users cannot monitor these readings remotely if they wish to do so. If alerts occur via text messages (SMS), then they have the obvious disadvantage of character-limit and lack of interactivity. Not to mention the costs that are incurred when sending text messages, the number for which cannot be absolutely determined.

2.4 PROPOSED SYSTEM

To overcome the disadvantages of the existing system, a low-cost Geiger Counter using Raspberry Pi as the computer is proposed. This system is relatively inexpensive, easy to repair and replace and is highly extensible using different components.

For example, we can attach various components to it, such as a temperature sensor, a humidity sensor, to collect environmental data from our surroundings, and have the Raspberry Pi process that data. Once it is processed, the data can be transmitted to an endpoint on the internet, which in our case is InfluxDB. Text alerts can be sent via the Telegram Responder Bot. Skyhook Precision Location Module can be used to obtain geographical location coordinates. All of this data can then be mapped to a location, and multiple instances of this low-cost and efficient Radiation and Environmental Conditions Monitor implementation can be deployed in various places of interest.

2.5 CONCLUSION

Our proposed project provides far more extensibility than any proprietary systems that exist out there. We also believe that that our project will prove to be helpful to those who need it as a Radiation and Environmental Conditions Monitor.

CHAPTER 3

ANALYSIS

3.1 HARDWARE REQUIREMENT

3.1.1. RASPBERRY PI 4B

Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The Raspberry Pi project originally leaned towards the promotion of teaching basic computer science in schools and in developing countries. The original model became more popular than anticipated, selling outside its target market for uses such as robotics. It is widely used in many areas, such as for weather monitoring, because of its low cost, modularity, and open design. It is typically used by computer and electronic hobbyists, due to its adoption of HDMI and USB devices.

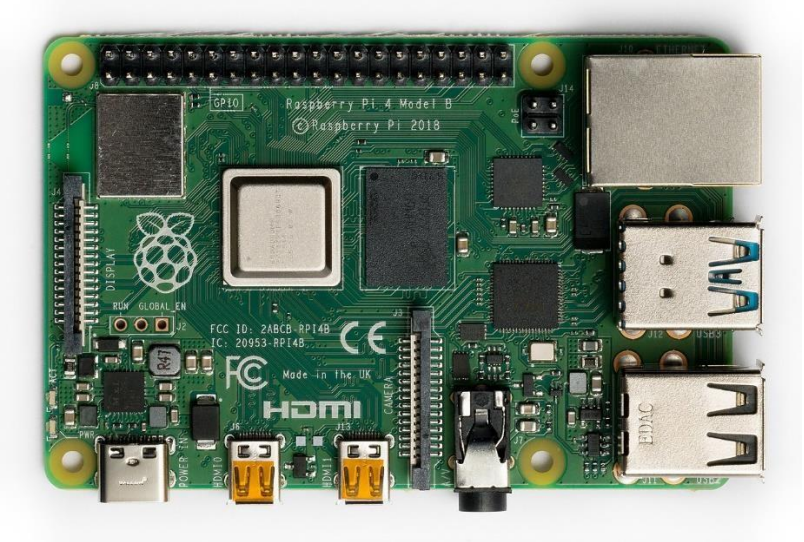

Fig 3.1 Raspberry Pi 4 Model B

Raspberry Pi 4 Model B was released in June 2019 with a 1.5 GHz 64-bit quad core ARM CortexA72 processor, on-board 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet (throughput not limited), two USB 2.0 ports, two USB 3.0 ports, and dual-monitor support via a pair of micro-HDMI (HDMI Type D) ports for up to 4K resolution. The Pi 4 is also powered via a USB-C port, enabling additional power to be provided to downstream peripherals, when used with an appropriate PSU. The initial Raspberry Pi 4 board has a design flaw where third-party e-marked USB cables, such as those used on Apple MacBooks, incorrectly identify it and refuse to provide power. Tom's Hardware tested 14 different cables and found that 11 of them turned on and powered the Pi without issue. The design flaw was fixed in revision 1.2 of the board, released in late 2019.
The Raspberry Pi hardware has evolved through several versions that feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral device support.

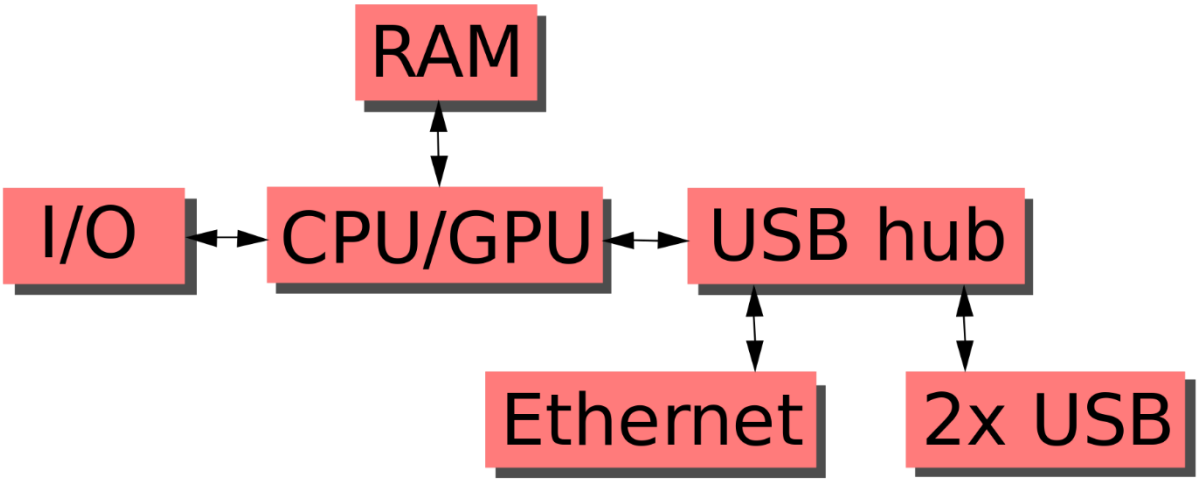

Fig 3.2 Block Diagram

This block diagram describes models B, B+, A and A+. The Pi Zero models are similar, but lack the Ethernet and USB hub components. The Ethernet adapter is internally connected to an additional USB port. In Model A, A+, and the Pi Zero, the USB port is connected directly to the system on a chip (SoC). On the Pi 1 Model B+ and later models the USB/Ethernet chip contains a five-port USB hub, of which four ports are available, while the Pi 1 Model B only provides two. On the Pi Zero, the USB port is also connected directly to the SoC, but it uses a micro USB (OTG) port. Unlike all other Pi models, the 40 pin GPIO connector is omitted on the Pi Zero, with solderable through-holes only in the pin locations. The Pi Zero WH remedies this.

Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ or 1.5 GHz for the Pi 4; on-board memory ranges from 256 MB to 1 GB random-access memory (RAM), with up to 8 GB available on the Pi 4. Secure Digital (SD) cards in MicroSDHC form factor (SDHC on early models) are used to store the operating system and program memory. The boards have one to five USB ports. For video output, HDMI and composite video are supported, with a standard 3.5 mm tip-ring-sleeve jack for audio output. Lower-level output is provided by a number of GPIO pins, which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3, Pi 4 and Pi Zero W have on-board Wi-Fi 802.11n and Bluetooth.

The Raspberry Pi Foundation provides Raspberry Pi OS (formerly called Raspbian), a Debian-based (32-bit) Linux distribution for download, as well as third-party Ubuntu, Windows 10 IoT Core, RISC OS, and LibreELEC (specialised media centre distribution).[146] It promotes Python and Scratch as the main programming languages, with support for many other languages. The default firmware is closed source, while unofficial open source is available. Many other operating systems can also run on the Raspberry Pi.

3.1.1.1 PROCESSOR

A central processing unit (CPU), also called a central processor, main processor or just processor, is the electronic circuitry that executes instructions comprising a computer program. The CPU performs basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions in the program. This contrasts with external components such as main memory and I/O circuitry, and specialized processors such as graphics processing units (GPUs).

The form, design, and implementation of CPUs have changed over time, but their fundamental operation remains almost unchanged. Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that orchestrates the fetching (from memory) and execution of instructions by directing the coordinated operations of the ALU, registers and other components.

The Broadcom BCM2835 SoC used in the first-generation Raspberry Pi includes a 700 MHz ARM1176JZF-S processor, VideoCore IV graphics processing unit (GPU), and RAM. It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible. The ARM1176JZ(F)-S is the same CPU used in the original iPhone, although at a higher clock rate, and mated with a much faster GPU.

The earlier V1.1 model of the Raspberry Pi 2 used a Broadcom BCM2836 SoC with a 900 MHz 32-bit, quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache.

The Raspberry Pi 2 V1.2 was upgraded to a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, the same SoC which is used on the Raspberry Pi 3, but underclocked (by default) to the same 900 MHz CPU clock speed as the V1.1. The BCM2836 SoC is no longer in production as of late 2016.

The Raspberry Pi 3 Model B uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache. The Model A+ and B+ are 1.4 GHz.

The Raspberry Pi 4 uses a Broadcom BCM2711 SoC with a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor, with 1 MB shared L2 cache. Unlike previous models, which all used a custom interrupt controller poorly suited for virtualization, the interrupt controller on this SoC is compatible with the ARM Generic Interrupt Controller (GIC) architecture 2.0, providing hardware support for interrupt distribution when using ARM virtualization capabilities.

The Raspberry Pi Zero and Zero W use the same Broadcom BCM2835 SoC as the first-generation Raspberry Pi, although now running at 1 GHz CPU clock speed.

The Raspberry Pi Pico uses the RP2040 running at 133 MHz.

3.1.1.1.1 PERFORMANCE

While operating at 700 MHz by default, the first-generation Raspberry Pi provided a real-world performance roughly equivalent to 0.041 GFLOPS. On the CPU level the performance is similar to a 300 MHz Pentium II of 1997–99. The GPU provides 1 Gpixel/s or 1.5 Gtexel/s of graphics processing or 24 GFLOPS of general-purpose computing performance. The graphical capabilities of the Raspberry Pi are roughly equivalent to the performance of the Xbox of 2001.

Raspberry Pi 2 V1.1 included a quad-core Cortex-A7 CPU running at 900 MHz and 1 GB RAM. It was described as 4–6 times more powerful than its predecessor. The GPU was identical to the original. In parallelized benchmarks, the Raspberry Pi 2 V1.1 could be up to 14 times faster than a Raspberry Pi 1 Model B+.

The Raspberry Pi 3, with a quad-core Cortex-A53 processor, is described as having ten times the performance of a Raspberry Pi 1. Benchmarks showed the Raspberry Pi 3 to be approximately 80% faster than the Raspberry Pi 2 in parallelized tasks.

The Raspberry Pi 4, with a quad-core Cortex-A72 processor, is described as having three times the performance of a Raspberry Pi 3.

3.1.1.1.2 OVERCLOCKING

In computing, overclocking is the practice of increasing the clock rate of a computer to exceed that certified by the manufacturer. Commonly, operating voltage is also increased to maintain a component's operational stability at accelerated speeds. Semiconductor devices operated at higher frequencies and voltages increase power consumption and heat. An overclocked device may be unreliable or fail completely if the additional heat load is not removed or power delivery components cannot meet increased power demands. Many device warranties state that overclocking or over-specification voids any warranty, however there are an increasing number of manufacturers that will allow overclocking as long as performed (relatively) safely.

While most modern devices are fairly tolerant of overclocking, all devices have finite limits. Generally for any given voltage most parts will have a maximum "stable" speed where they still operate correctly. Past this speed, the device starts giving incorrect results, which can cause malfunctions and sporadic behaviour in any system depending on it. While in a PC context the usual result is a system crash, more subtle errors can go undetected, which over a long enough time can give unpleasant surprises such as data corruption (incorrectly calculated results, or worse writing to storage incorrectly) or the system failing only during certain specific tasks (general usage such as internet browsing and word processing appear fine, but any application wanting advanced graphics crashes the system).

At this point, an increase in operating voltage of a part may allow more headroom for further increases in clock speed, but the increased voltage can also significantly increase heat output, as well as shorten the lifespan further. At some point, there will be a limit imposed by the ability to supply the device with sufficient power, the user's ability to cool the part, and the device's own maximum voltage

tolerance before it achieves destructive failure. Overzealous use of voltage or inadequate cooling can rapidly degrade a device's performance to the point of failure, or in extreme cases outright destroy it.

Most Raspberry Pi systems-on-chip could be overclocked to 800 MHz, and some to 1000 MHz.

There are reports the Raspberry Pi 2 can be similarly overclocked, in extreme cases, even to 1500 MHz (discarding all safety features and over-voltage limitations). In the Raspbian Linux distro the overclocking options on boot can be done by a software command running "sudo raspi-config" without voiding the warranty. In those cases, the Pi automatically shuts the overclocking down if the chip temperature reaches 85 °C (185 °F), but it is possible to override automatic over-voltage and overclocking settings (voiding the warranty); an appropriately sized heat sink is needed to protect the chip from serious overheating.

Newer versions of the firmware contain the option to choose between five overclock ("turbo") presets that, when used, attempt to maximize the performance of the SoC without impairing the lifetime of the board. This is done by monitoring the core temperature of the chip and the CPU load, and dynamically adjusting clock speeds and the core voltage.

When the demand is low on the CPU or it is running too hot, the performance is throttled, but if the CPU has much to do and the chip's temperature is acceptable, performance is temporarily increased with clock speeds of up to 1 GHz, depending on the board version and on which of the turbo settings is used.

The overclocking modes are:

- none; 700 MHz ARM, 250 MHz core, 400 MHz SDRAM, 0 overvolting,
- modest; 800 MHz ARM, 250 MHz core, 400 MHz SDRAM, 0 overvolting,
- medium; 900 MHz ARM, 250 MHz core, 450 MHz SDRAM, 2 overvolting,
- high; 950 MHz ARM, 250 MHz core, 450 MHz SDRAM, 6 overvolting,
- turbo; 1000 MHz ARM, 500 MHz core, 600 MHz SDRAM, 6 overvolting,
- Pi 2; 1000 MHz ARM, 500 MHz core, 500 MHz SDRAM, 2 overvolting,
- Pi 3; 1100 MHz ARM, 550 MHz core, 500 MHz SDRAM, 6 overvolting.

In system information the CPU speed appears as 1200 MHz. When idling, speed lowers to 600 MHz. In the highest (turbo) mode the SDRAM clock speed was originally 500 MHz, but this was later changed to 600 MHz because of occasional SD card corruption. Simultaneously, in high mode the core clock speed was lowered from 450 to 250 MHz, and in medium mode from 333 to 250 MHz.

The CPU of the first- and second-generation Raspberry Pi board did not require cooling with a heat sink or fan, even when overclocked, but the Raspberry Pi 3 may generate more heat when overclocked.

3.1.1.2 RAM

Random-access memory (RAM; /ræm/) is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older magnetic tapes and drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.

RAM contains multiplexing and demultiplexing circuitry, to connect the data lines to the addressed storage for reading or writing the entry. Usually more than one bit of storage is accessed by the same address, and RAM devices often have multiple data lines and are said to be "8-bit" or "16-bit", etc. devices.

The early designs of the Raspberry Pi Model A and B boards included only 256 MB of random access memory (RAM). Of this, the early beta Model B boards allocated 128 MB to the GPU by default, leaving only 128 MB for the CPU. On the early 256 MB releases of models A and B, three different splits were possible. The default split was 192 MB for the CPU, which should be sufficient for standalone 1080p video decoding, or for simple 3D processing. 224 MB was for Linux processing only, with only a 1080p framebuffer, and was likely to fail for any video or 3D.

128 MB was for heavy 3D processing, possibly also with video decoding. In comparison, the Nokia 701 uses 128 MB for the Broadcom VideoCore IV.

The later Model B with 512 MB RAM, was released on 15 October 2012 and was initially released with new standard memory split files (arm256_start.elf, arm384_start.elf, arm496_start.elf) with 256 MB, 384 MB, and 496 MB CPU RAM, and with 256 MB, 128 MB, and 16 MB video RAM, respectively. But about one week later, the foundation released a new version of start.elf that could read a new entry in config.txt (gpu_mem=xx) and could dynamically assign an amount of RAM (from 16 to 256 MB in 8 MB steps) to the GPU, obsoleting the older method of splitting memory, and a single start.elf worked the same for 256 MB and 512 MB Raspberry Pis.

The Raspberry Pi 2 has 1 GB of RAM. The Raspberry Pi 3 has 1 GB of RAM in the B and B+ models, and 512 MB of RAM in the A+ model. The Raspberry Pi Zero and Zero W have 512 MB of RAM. The Raspberry Pi 4 is available with 2, 4 or 8 GB of RAM. A 1 GB model was originally available at launch in June 2019 but was discontinued in March 2020, and the 8 GB model was introduced in May 2020.

3.1.1.3 NETWORKING

The Model A, A+ and Pi Zero have no Ethernet circuitry and are commonly connected to a network using an external user-supplied USB Ethernet or Wi-Fi adapter. On the Model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter using the SMSC LAN9514 chip. The Raspberry Pi 3 and Pi Zero W (wireless) are equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) based on the Broadcom BCM43438 FullMAC chip with no official support for monitor mode (though it was implemented through unofficial firmware patching) and the Pi 3 also has a 10/100 Mbit/s Ethernet port. The Raspberry Pi 3B+ features dualband IEEE 802.11b/g/n/ac WiFi, Bluetooth 4.2, and Gigabit Ethernet (limited to approximately 300 Mbit/s by the USB 2.0 bus between it and the SoC). The Raspberry Pi 4 has full gigabit Ethernet (throughput is not limited as it is not funnelled via the USB chip.)

3.1.1.4 REAL TIME CLOCK

A real-time clock (RTC) is an electronic device (most often in the form of an integrated circuit) that measures the passage of time.

Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in almost any electronic device which needs to keep accurate time of day.

The term real-time clock is used to avoid confusion with ordinary hardware clocks which are only signals that govern digital electronics, and do not count time in human units. RTC should not be confused with real-time computing, which shares its three-letter acronym but does not directly relate to time of day.

Most RTCs use a crystal oscillator, but some have the option of using the power line frequency. The crystal frequency is usually 32.768 kHz, the same frequency used in quartz clocks and watches. Being exactly 215 cycles per second, it is a convenient rate to use with simple binary counter circuits. The low frequency saves power, while remaining above human hearing range. The quartz tuning fork of these crystals does not change size much from temperature, so temperature does not change its frequency much.

When booting, the time defaults to being set over the network using the Network Time Protocol (NTP). The source of time information can be another computer on the local network that does have a real-time clock, or to a NTP server on the internet. If no network connection is available, the time may be set manually or configured to assume that no time passed during the shutdown. In the latter case, the time is monotonic (files saved later in time always have later timestamps) but may be considerably earlier than the actual time. For systems that require a built-in real-time clock, several small, low-cost add-on boards with real-time clocks are available.

The RP2040 microcontroller has a built-in real-time clock but this cannot be set automatically without some form of user entry or network facility being added.

3.1.1.5 RASPBERRY PI 4B PINOUT

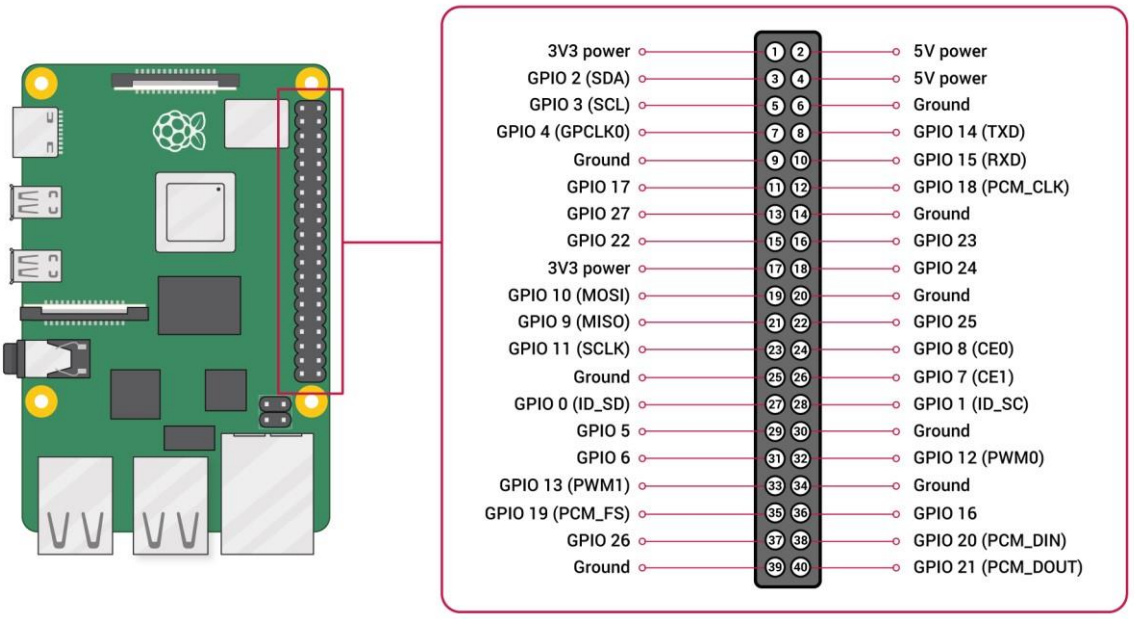

Fig 3.3 Raspberry Pi 4B Pinout

In electronics, a pinout (sometimes written "pin-out") is a cross-reference between the contacts, or pins, of an electrical connector or electronic component, and their functions. "Pinout" now supersedes the term "basing diagram" that was the standard terminology used by the manufacturers of vacuum tubes and the RMA. The RMA started its standardization in 1934, collecting and correlating tube data for registration at what was to become the EIA. The EIA (Electronic Industries Alliance) now has many sectors reporting to it and sets what are known as EIA standards where all registered pinouts and registered jacks can be found.

The functions of contacts in electrical connectors, be they power- or signalling-related, must be specified for connectors to be interchangeable. When connected, each contact of a connector must mate with the contact on the other connector that has the same function. If contacts of disparate functions are allowed to make contact, the connection may fail and damage may result. Therefore, pinouts are a vital reference when building and testing connectors, cables, and adapters.

All Raspberry Pi’s with the standard 40 GPIO pins will have two 5V pins and two 3.3V pins, always in the same place. Along with the 5V and 3.3V pins, 8 ground pins are available. Power and ground pins are what let your Raspberry Pi power components like LEDs and motors.

In many ways, the Raspberry Pi 4 improves upon the features set by the Pi models before it. Not only does the single-board computer support more RAM, a faster processor speed, and expanded peripherals, but the GPIO pins retain their standard functions set by previous models along with extra functions for existing pins.

3.1.2 GEIGER MÜLLER TUBE

A G-M tube consists of a chamber filled with a gas mixture at a low pressure of about 0.1 atmosphere. The chamber contains two electrodes, between which there is a potential difference of several hundred volts. The walls of the tube are either metal or have their inside surface coated with a conducting material or a spiral wire to form the cathode, while the anode is a wire mounted axially in the centre of the chamber.

When ionizing radiation strikes the tube, some molecules of the fill gas are ionized directly by the incident radiation, and if the tube cathode is an electrical conductor, such as stainless steel, indirectly by means of secondary electrons produced in the walls of the tube, which migrate into the gas. This creates positively charged ions and free electrons, known as ion pairs, in the gas. The strong electric field created by the voltage across the tube's electrodes accelerates the positive ions towards the cathode and the electrons towards the anode. Close to the anode in the "avalanche region" where the electric field strength rises inversely proportional to radial distance as the anode is approached, free electrons gain sufficient energy to ionize additional gas molecules by collision and create a large number of electron avalanches. These spread along the anode and effectively throughout the avalanche region. This is the "gas multiplication" effect which gives the tube its key characteristic of being able to produce a significant output pulse from a single original ionising event.

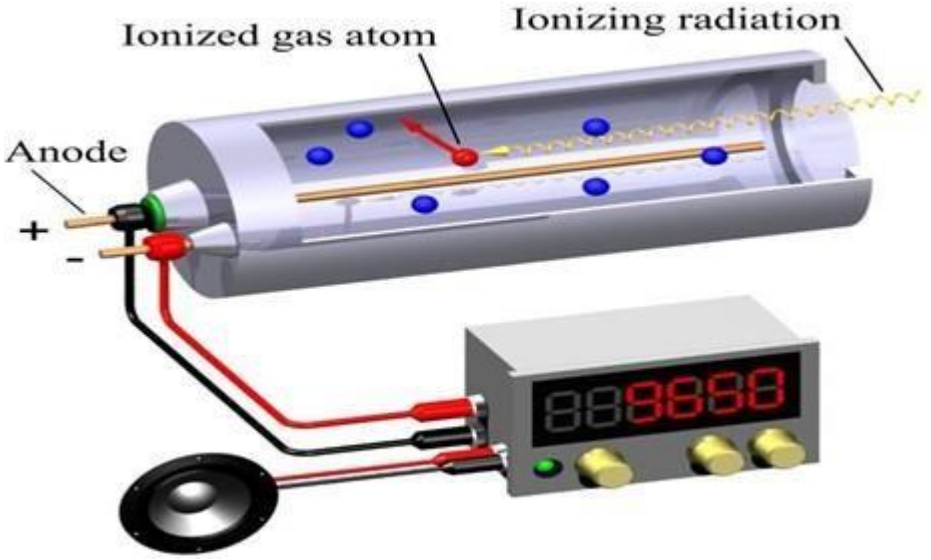

Fig 3.4 Geiger–Müller Tube

3.1.2. DC-DC BOOST CONVERTER AND INTERFACE TO RASPBERRY PI

A boost converter (step-up converter) is a DC-to-DC power converter that steps up voltage (while stepping down current) from its input (supply) to its output (load). It is a class of switched-mode power supply (SMPS) containing at least two semiconductors (a diode and a transistor) and at least one energy storage element: a capacitor, inductor, or the two in combination. To reduce voltage ripple, filters made of capacitors (sometimes in combination with inductors) are normally added to such a converter's output (load-side filter) and input (supply-side filter).

Power for the boost converter can come from any suitable DC source, such as batteries, solar panels, rectifiers, and DC generators. A process that changes one DC voltage to a different

DC voltage is called DC to DC conversion. A boost converter is a DC-to-DC converter with an output voltage greater than the source voltage. A boost converter is sometimes called a step-up converter since it "steps up" the source voltage. Since power (P=VI) must be conserved, the output current is lower than the source current.

Fig 3.5 DC-DC Boost Converter with interface to Raspberry Pi

The pins used on the DC-DC Boost Converter board are as follows:

- **Vin:** This is an output pin that we connect to D2 on Arduino and collect the count-rate from. This count rate is then converted to radiation units.

- **5V:** This is an input pin where we supply 5V input. The Boost Converter uses this input and boosts the voltage to 300-400V, which is the operating voltage for the GM Tube.

- **GND**: Ground pins.

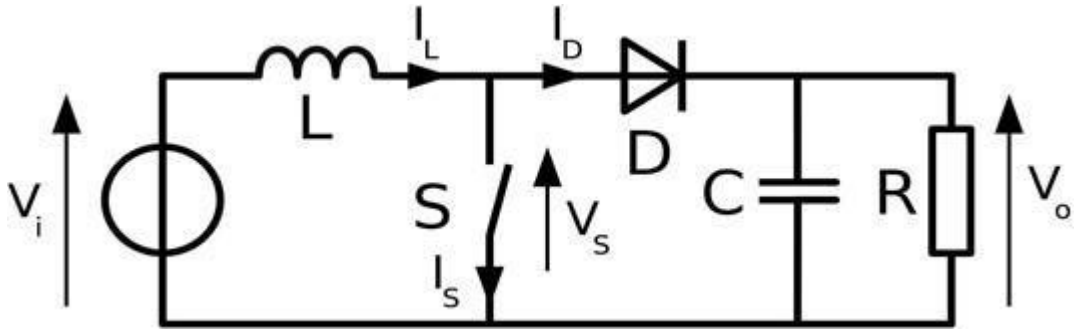

Fig 3.6 Boost Converter Schematic

3.1.4 DHT11 SENSOR MODULE

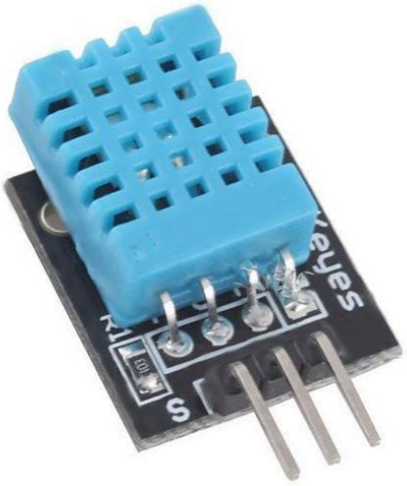

Fig 3.7 DHT11 Sensor Module

Temperature is a physical quantity that expresses hot and cold. It is the manifestation of thermal energy, present in all matter, which is the source of the occurrence of heat, a flow of energy, when a body is in contact with another that is colder or hotter.

Humidity is the concentration of water vapour present in the air. Water vapor, the gaseous state of water, is generally invisible to the human eye. Humidity indicates the likelihood for precipitation, dew, or fog to be present.

The DHT11 is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers.

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measures, processes this changed resistance values and changes them into a digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. Humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz .i.e. it gives one reading for every second. DHT11 is small in size with operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA.

Pin Name	Description
VCC	Power supply 3.5V to 5.5V
Data	Outputs both Temperature and Humidity through serial data
Ground	Connected to the ground of the circuit

Table 1 : DHT11 Sensor Module Pinout:

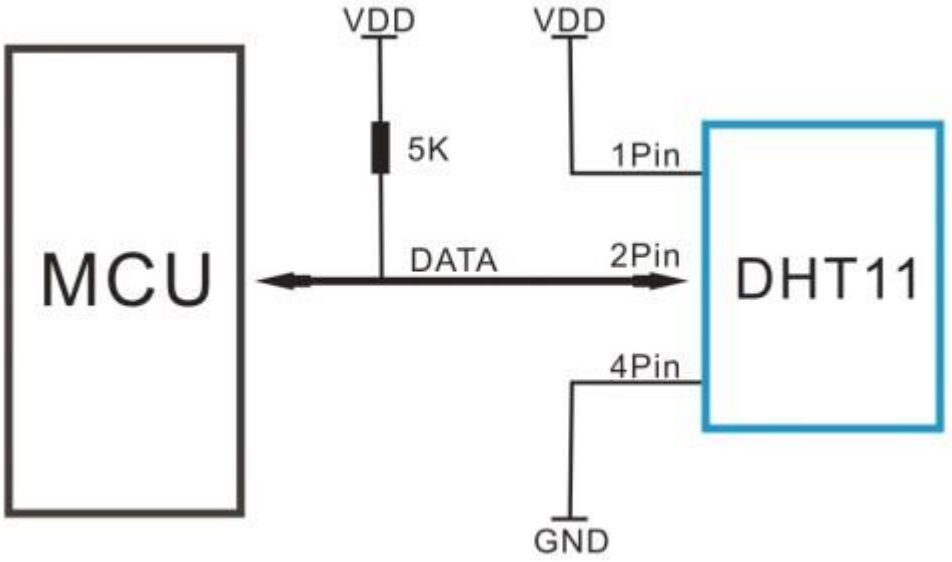

Fig 3.8 Connection Diagram for DHT11 Sensor Module

3.2 SOFTWARE REQUIREMENT

3.2.1 PYTHON 3.8 OR HIGHER

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed, and garbage collected. Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system.[10] Its implementation began in December 1989. Python is usually installed along with a Linux distribution. You can simply check if it is installed using the following command:

```
$ python3 --version
```

This command will either print the version number that is installed on your computer or will give you an error, in which case you need to look up how to install Python for your operating system.

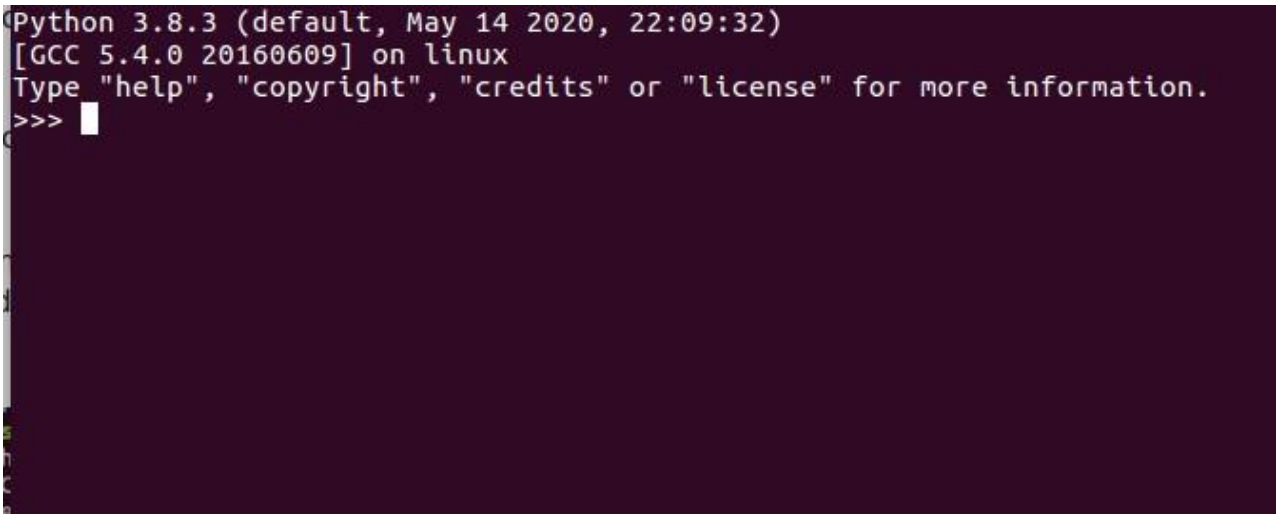

Fig 3.9: The Python 3.8.3 Prompt

Once, Python has been successfully installed, ensure that you have also installed Pip, which is Python's official package manager. You can check if Pip is successfully installed the same way you checked it for Python:

```
$ pip3 --version
```

Again, if it is not installed already, look up how to install it for your operating system.

3.2.2 OPENSSH

ssh (SSH client) is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections, arbitrary TCP ports and UNIX-domain sockets can also be forwarded over the secure channel.

ssh connects and logs into the specified destination, which may be specified as either [user@]hostname or a URI of the form ssh://[user@]hostname[:port].

OpenSSH (also known as OpenBSD Secure Shell) is a suite of secure networking utilities based on the Secure Shell (SSH) protocol, which provides a secure channel over an unsecured network in a client–server architecture.

OpenSSH started as a fork of the free SSH program developed by Tatu Ylönen; later versions of Ylönen's SSH were proprietary software offered by SSH Communications Security. OpenSSH was first released in 1999 and is currently developed as part of the OpenBSD operating system.

OpenSSH is not a single computer program, but rather a suite of programs that serve as alternatives to unencrypted protocols like Telnet and FTP. OpenSSH is integrated into several operating systems, namely Microsoft Windows, macOS and most Linux operating systems, while the portable version is available as a package in other systems.

OpenSSH includes the ability to set up a secured channel through which data sent to local, clientside Unix domain sockets or local, client-side TCP ports may be "forwarded" (sent across the secured channel) for routing on the server side; when this forwarding is set up, the server is instructed to send that forwarded data to some socket or TCP host/port (the host could be the server itself, "localhost"; or, the host may be some other computer, so that it appears to the other computer that the server is the originator of the data). The forwarding of data is bidirectional, meaning that any return communication is itself forwarded back to the client-side in the same manner; this is known as an "SSH tunnel", and it can be used to multiplex additional TCP connections over a single SSH connection since 2004, to conceal connections, to encrypt protocols that are otherwise unsecured, and to circumvent firewalls by sending/receiving all manner of data through one port that is allowed by the firewall. For example, an X Window System tunnel may be created automatically when using OpenSSH to connect to a remote host, and other protocols, such as HTTP and VNC, may be forwarded easily.

The OpenSSH server can authenticate users using the standard methods supported by the ssh protocol: with a password; public-key authentication, using per-user keys; host-based authentication, which is a secure version of rlogin's host trust relationships using public keys; keyboard-interactive, a generic challenge–response mechanism, which is often used for simple password authentication, but which can also make use of stronger authenticators such as tokens; and Kerberos/GSSAPI.

Tunneling a TCP-encapsulating payload (such as PPP) over a TCP-based connection (such as SSH's port forwarding) is known as "TCP-over-TCP", and doing so can induce a dramatic loss in

transmission performance (a problem known as "TCP meltdown"), which is why virtual private network software may instead use for the tunnel connection a protocol simpler than TCP.

However, this is often not a problem when using OpenSSH's port forwarding, because many use cases do not entail TCP-over-TCP tunneling; the meltdown is avoided because the OpenSSH client processes the local, client-side TCP connection in order to get to the actual payload that is being sent, and then sends that payload directly through the tunnel's own TCP connection to the server side, where the OpenSSH server similarly "unwraps" the payload in order to "wrap" it up again for routing to its final destination.

In addition, some third-party software includes support for tunnelling over SSH. These include DistCC, CVS, rsync, and Fetchmail. On some operating systems, remote file systems can be mounted over SSH using tools such as sshfs (using FUSE).

An ad hoc SOCKS proxy server may be created using OpenSSH. This allows more flexible proxying than is possible with ordinary port forwarding.

Beginning with version 4.3, OpenSSH implements an OSI layer 2/3 tun-based VPN. This is the most flexible of OpenSSH's tunnelling capabilities, allowing applications to transparently access remote network resources without modifications to make use of SOCKS.

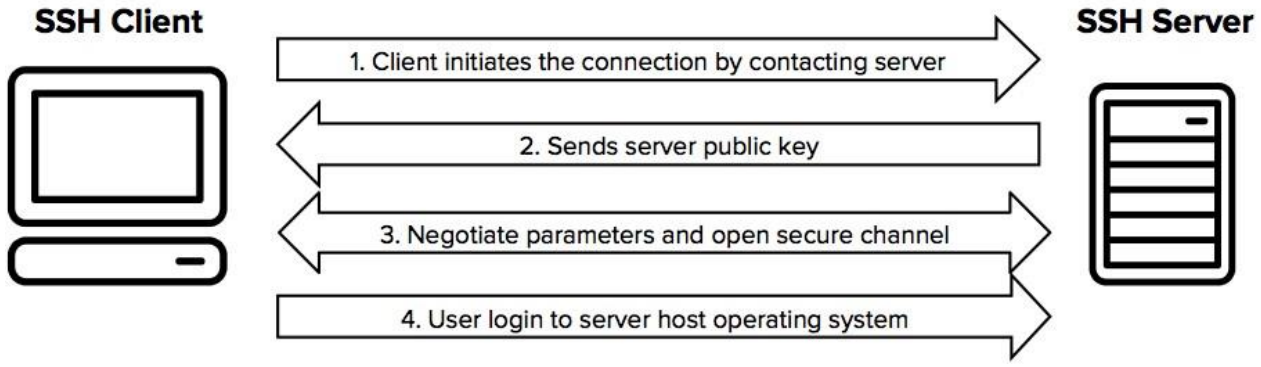

Fig 3.10 SSH Connection Steps

SSH protocol uses symmetric encryption, asymmetric encryption and hashing in order to secure transmission of information. The SSH connection between the client and the server happens in three stages:

- Verification of the server by the client.
- Generation of a session key to encrypt all the communication.
- Authentication of the client.

3.2.3. SSH FILE TRANSFER PROTOCOL (SFTP)

In computing, the SSH File Transfer Protocol (also Secure File Transfer Protocol, or SFTP) is a network protocol that provides file access, file transfer, and file management over any reliable data stream. It was designed by the Internet Engineering Task Force (IETF) as an extension of the Secure Shell protocol (SSH) version 2.0 to provide secure file transfer capabilities. The IETF Internet Draft states that, even though this protocol is described in the context of the SSH-2 protocol, it could be used in a number of different applications, such as secure file transfer over Transport Layer Security (TLS) and transfer of management information in VPN applications.

This protocol assumes that it is run over a secure channel, such as SSH, that the server has already authenticated the client, and that the identity of the client user is available to the protocol.

Compared to the SCP protocol, which only allows file transfers, the SFTP protocol allows for a range of operations on remote files which make it more like a remote file system protocol. An SFTP client's extra capabilities include resuming interrupted transfers, directory listings, and remote file removal. Compared to the SCP protocol, which only allows file transfers, the SFTP protocol allows for a range of operations on remote files which make it more like a remote file system protocol. An SFTP client's extra capabilities include resuming interrupted transfers, directory listings, and remote file removal.

SFTP attempts to be more platform-independent than SCP; with SCP, for instance, the expansion of wildcards specified by the client is up to the server, whereas SFTP's design avoids this problem. While SCP is most frequently implemented on Unix platforms, SFTP servers are commonly available on most platforms. The file transfer is fast in SCP when compared to the SFTP protocol due to the back-and-forth nature of SFTP protocol. In SFTP, the file transfer can be easily terminated without terminating a session like other mechanisms do.

SFTP is not FTP run over SSH, but rather a new protocol designed from the ground up by the IETF SECSH working group. It is sometimes confused with Simple File Transfer Protocol.

The protocol itself does not provide authentication and security; it expects the underlying protocol to secure this. SFTP is most often used as subsystem of SSH protocol version 2 implementations, having been designed by the same working group. It is possible, however, to run it over SSH-1 (and some implementations support this) or other data streams. Running an SFTP server over SSH-1 is not platform-independent as SSH-1 does not support the concept of subsystems. An SFTP client willing to connect to an SSH-1 server needs to know the path to the SFTP server binary on the server side.

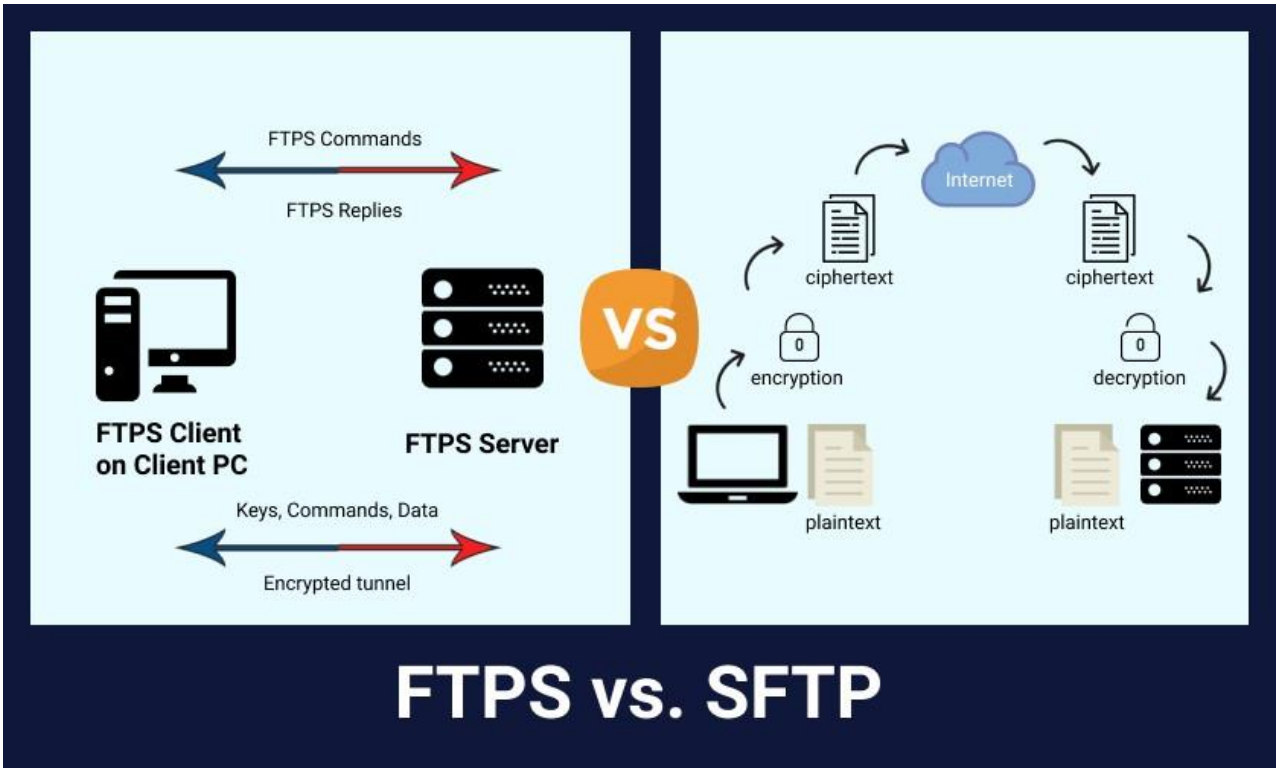

Fig 3.11 FTPS vs SFTP

Uploaded files may be associated with their basic attributes, such as time stamps. This is an advantage over the common FTP protocol.

SFTP attempts to be more platform-independent than SCP; with SCP, for instance, the expansion of wildcards specified by the client is up to the server, whereas SFTP's design avoids this problem. While SCP is most frequently implemented on Unix platforms, SFTP servers are commonly available on most platforms. The file transfer is fast in SCP when compared to the SFTP protocol due to the back-and-forth nature of SFTP protocol. In SFTP, the file transfer can be easily terminated without terminating a session like other mechanisms do.

SFTP is not FTP run over SSH, but rather a new protocol designed from the ground up by the IETF SECSH working group. It is sometimes confused with Simple File Transfer Protocol.

The protocol itself does not provide authentication and security; it expects the underlying protocol to secure this. SFTP is most often used as subsystem of SSH protocol version 2 implementations, having been designed by the same working group. It is possible, however, to run it over SSH-1 (and some implementations support this) or other data streams. Running an SFTP server over SSH-1 is not platform-independent as SSH-1 does not support the concept of subsystems. An SFTP client willing to connect to an SSH-1 server needs to know the path to the SFTP server binary on the server side. Uploaded files may be associated with their basic attributes, such as time stamps. This is an advantage over the common FTP protocol.

3.2.4. INFLUX DB

Influx DB is an open-source time series database (TSDB) developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. It also has support for processing data from Graphite.

Influx DB has no external dependencies and provides an SQL-like language, listening on port 8086, with built-in time-centric functions for querying a data structure composed of measurements, series, and points. Each point consists of several key-value pairs called the field set and a timestamp. When grouped together by a set of key-value pairs called the tag set, these define a series. Finally, series are grouped together by a string identifier to form a measurement.

Values can be 64-bit integers, 64-bit floating points, strings, and Booleans. Points are indexed by their time and tag set. Retention policies are defined on a measurement and control how data is down sampled and deleted. Continuous Queries run periodically, storing results in a target measurement.

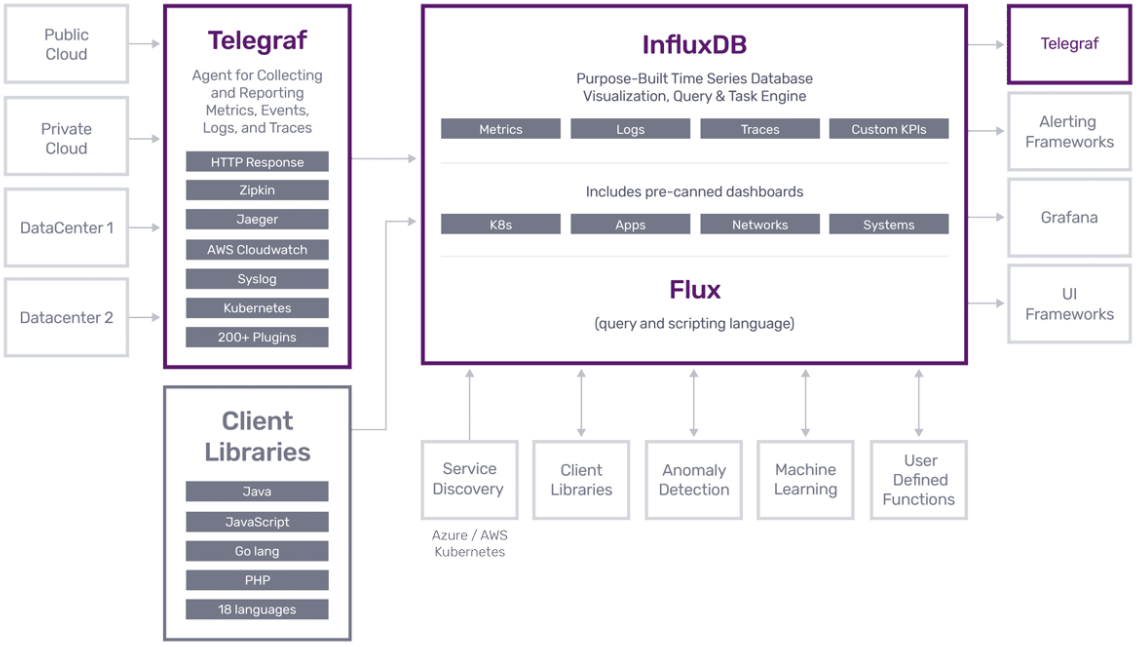

Fig 3.12 Influx DB Architecture

Fig 3.13 Example of data collected in Influx DB

3.2.5 GRAFANA

Grafana is a multi-platform open-source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources. A licensed Grafana Enterprise version with additional capabilities is also available as a self-hosted installation or an account on the Grafana Labs cloud service. It is expandable through a plug-in system. End users can create complex monitoring dashboards using interactive query builders. Grafana is divided into a front end and back end, written in TypeScript and Go, respectively.

Analytics is the systematic computational analysis of data or statistics.[1] It is used for the discovery, interpretation, and communication of meaningful patterns in data. It also entails applying data patterns towards effective decision-making. It can be valuable in areas rich with recorded information; analytics relies on the simultaneous application of statistics, computer programming and operations research to quantify performance.

Organizations may apply analytics to business data to describe, predict, and improve business performance. Specifically, areas within analytics include predictive analytics, prescriptive analytics, descriptive analytics, cognitive analytics, Big Data Analytics, retail analytics, supply chain analytics, store assortment and stock-keeping unit optimization, marketing optimization and marketing mix modeling, web analytics, call analytics, speech analytics, sales force sizing and optimization, price and promotion modeling, predictive science, graph analytics, credit risk analysis, and fraud analytics. Since analytics can require extensive computation (see big data), the algorithms and software used for analytics harness the most current methods in computer science, statistics, and mathematics.

As a visualization tool, Grafana is a popular component in monitoring stacks, often used in combination with time series databases such as Influx DB, Prometheus and Graphite; monitoring platforms such as Sensu, Icinga, Checkmk, Zabbix, Netdata, and PRTG; SIEMs such as Elasticsearch and Splunk; and other data sources.

Fig 3.14 Grafana Dashboard

Fig 3.15 Graphical representation of *Radiation vs. Time* in Grafana

3.2.6 SKYHOOK PRECISION LOCATION

Skyhook's Precision Location hybrid positioning system locates devices using Wi-Fi, GNSS and Cell signals, ensuring that all devices can be located in all environments. The solution provides the ability to locate devices even when offline, in an extremely power efficient manner - offering integration options suitable for the entire spectrum of connected devices. This system enables Skyhook to provide the most accurate and precise location available.

Geo-positioning, also known as geo-tracking, geo-localization, geolocating, geolocation, or geo-position fixing is the process of determining or estimating the geographic position of an object.

Geo-positioning yields a set of geographic coordinates (such as latitude and longitude) in a given map datum; positions may also be expressed as a bearing and range from a known landmark. In turn, positions can determine a meaningful location, such as a street address.

Geo-positioning uses various visual and electronic methods including position lines and position circles, celestial navigation, radio navigation, and the use of satellite navigation systems.

The calculation requires measurements or observations of distances or angles to reference points whose positions are known. In 2D surveys, observations of three reference points are enough to compute a position in a two dimensional plane. In practice, observations are subject to errors resulting from various physical and atmospheric factors that influence the measurement of distances and angles.

Specific instances include animal geotracking, the process of inferring the location of animals; positioning systems, the mechanisms for the determination of geographic positions in general; internet geolocation, geolocating a device connected to the internet; and mobile phone tracking.

A practical example of obtaining a position fix would be for a ship to take bearing measurements on three lighthouses positioned along the coast. These measurements could be made visually using a hand bearing compass, or in poor visibility, electronically using radar or radio direction finding. Since all physical observations are subject to errors, the resulting position fix is also subject to inaccuracy.

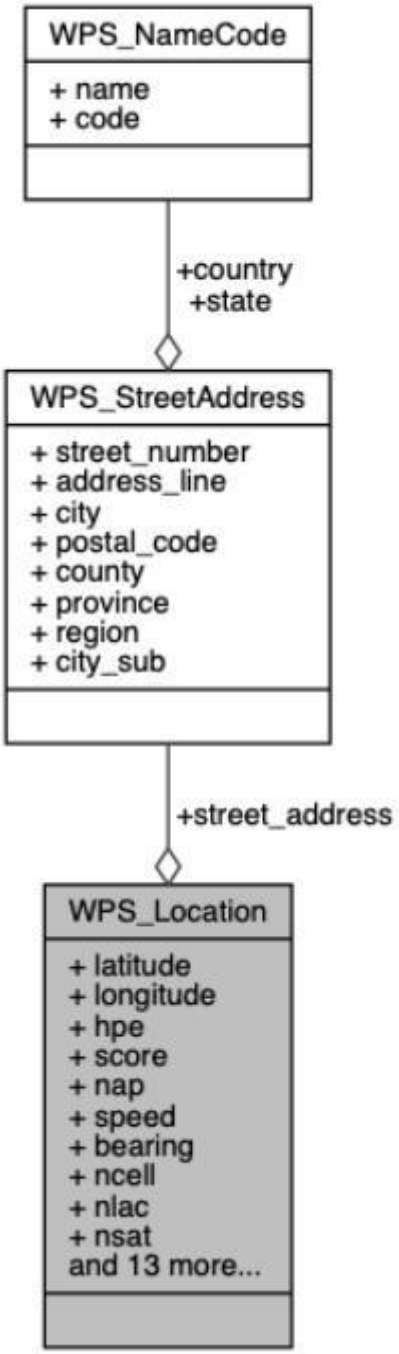

Fig 3.16 Skyhook Precision Location Data Structure

3.3 BLOCK DIAGRAM

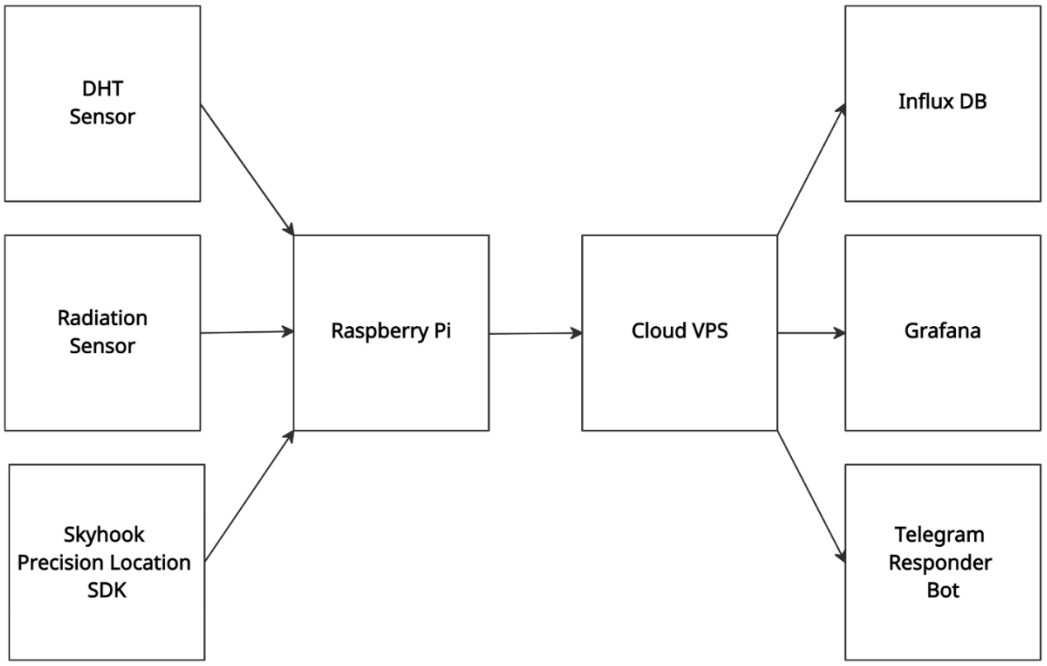

Fig 3.17 Block diagram

3.5 CONCLUSION

Thus, these are the hardware and software components that are required for developing the project. These components which are used for this project are designed for basic use. For ab advanced implementation we would require a more expensive VPS with larger memory and storage capacity and a more powerful processor.

CHAPTER 4

DESIGN

4.1. MODULE DESIGN

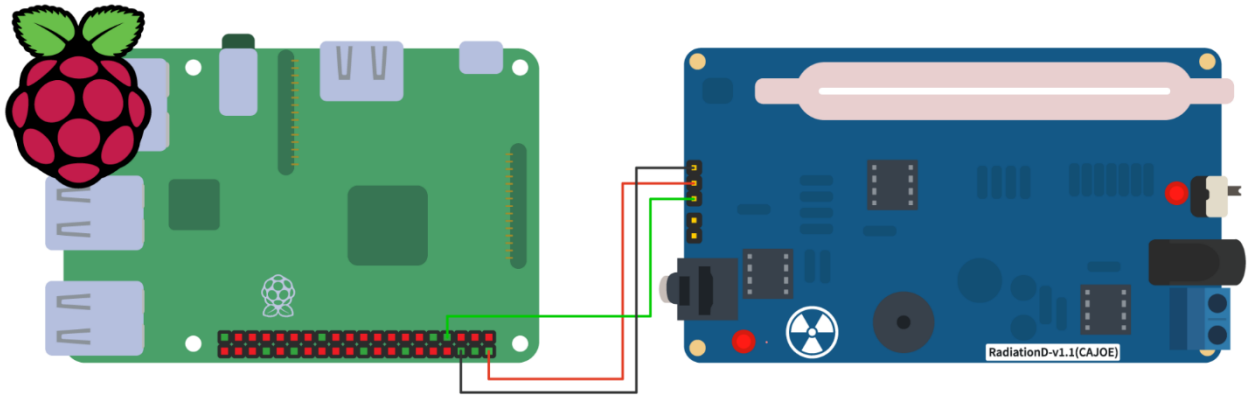

Fig 4.1 Connection of Geiger Counter with Raspberry Pi

In this design, the Radiation Monitor (Geiger Counter) is interfaced with Raspberry Pi. For this circuit the input required is (5V). VCC (5V) of the Radiation Monitor is connected to 5V Pin of the Raspberry Pi, GND (Ground) pin of the Radiation Monitor is connected to the ground pin of the Raspberry Pi and the VIN (Data) pin of the Radiation Monitor is connected to the GPIO 4 pin of the Raspberry Pi.

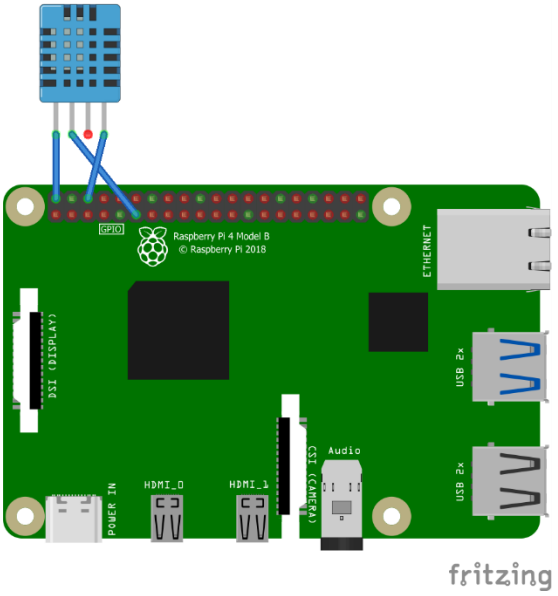

Fig 4.2 Connection of DHT11 Module with Raspberry Pi

In this design, The Raspberry Pi is interfaced to the DHT11 Module. The VCC (5V) pin is conncected to the 5V pin of the Raspberry Pi. GND (Ground) pin of the DHT11 Sensor Module is connected to

the ground pin of the Raspberry Pi and the DAT (Data) pin of the DHT11 Sensor Module is connected to the GPIO 4 pin of the Raspberry Pi.

4.2 FLOW DIAGRAM

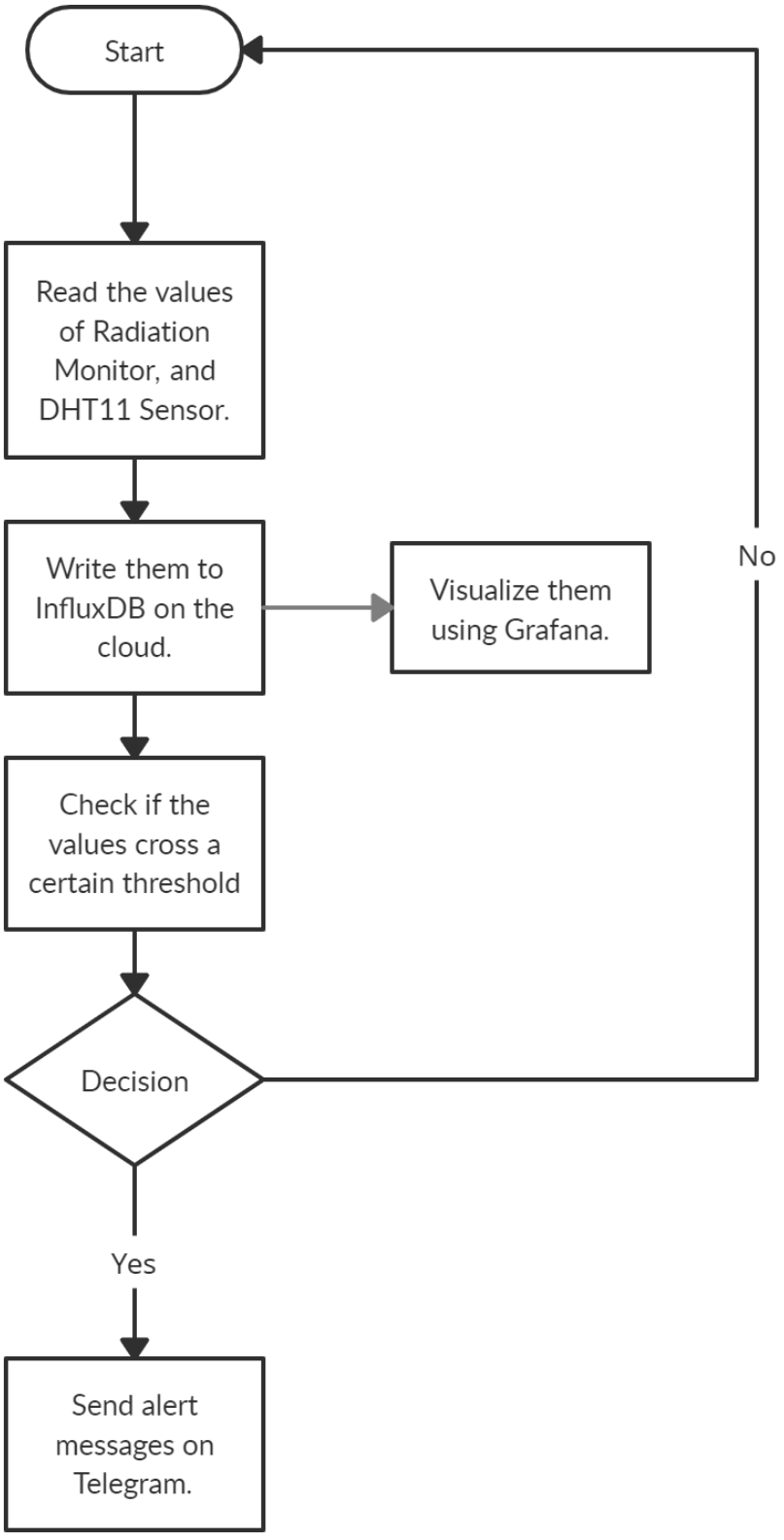

Fig 4.3 Flow Diagram of Radiation and Environmental Conditions Monitor

The above architecture gives us a complete idea of a Radiation and Environmental conditions Monitor with Telegram Responder Bot. This system can be accessed by authorized parties from anywhere in the world.

4.3 CONCLUSION

These are the designs for constructing the project for attaining the required results. These designs can be modified in the future based on our requirements.

CHAPTER 5
RESULTS & DISSCUSSION

5.1 WORKING PRINCIPLE

5.1.1 RADIATION MONITOR USING GEIGER-MÜLLER COUNTER

Radiations are measured using the Geiger-Müller counter. The GM counter measures the radiations based on the principle of ionization. The terminals of a GM Tube are kept at a potential difference of 350V-400V, and this voltage is provided by the DC-DC Boost Converter. Whenever any ionizing radiation falls on the GM Tube, a current pulse is conducted through it. This pulse is of very short duration, and the GM counter counts these pulses in the form of CPM (Counts per minute). This value is converted to micro-Sieverts per hour (μSv/h) using a conversion factor that's specific to each GM Tube. In the case of J305β GM Tube, it is 0.00812037037037.

5.1.2 ENVIRONMENTAL CONDITIONS MONITOR USING DHT11 SENSOR MODULE

Temperature and Humidity are two environmental conditions that provide us more information about our surroundings. Various parameters in the environment affect their values and are therefore of interest to us. In our project, we use the DHT11 Sensor Module to measure these environmental conditions. Adafruit provides us with a library that makes interfacing of the DHT11 Sensor Module to the Raspberry Pi very easy. So, all one has to do is to install the said library by following the instructions on their website and use the starter code to start reading the Temperature and Humidity using the DHT11 Sensor Module. Of course, the code needs to be modified a little so that we can use it the way we want to.

5.1.3 SKYHOOK PRECISION LOCATION SDK

Using a physical GPS module takes a considerable amount of time just to lock on to the three minimum GPS satellites that is required to triangulate our position. And if one lives in a concrete jungle like Hyderabad, this can take a lot of time (even days) starting from a cold boot. Since we cannot presume beforehand the location of where our project will be deployed (despite the recommended places), we would need a better and a faster alternative. Since an internet connection is a basic requirement for our project (for the purposes of relaying sensor values to Influx DB that is hosted on the cloud), we can use Geolocation software modules like Skyhook Precision Location. The only drawback here is that the API for the SDK is available only in C. This can be easily overcome as Python easily integrates with C, and so we can call a C function from Python.

5.2 RESULT & TESTING

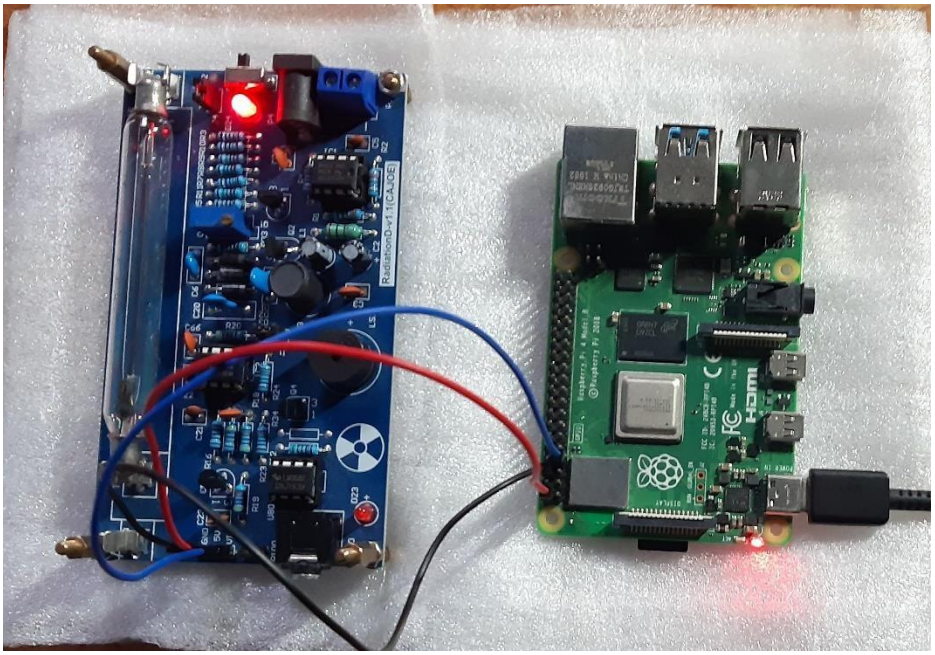

Fig 5.1 : Geiger-Müller Counter interfaced to Raspberry Pi

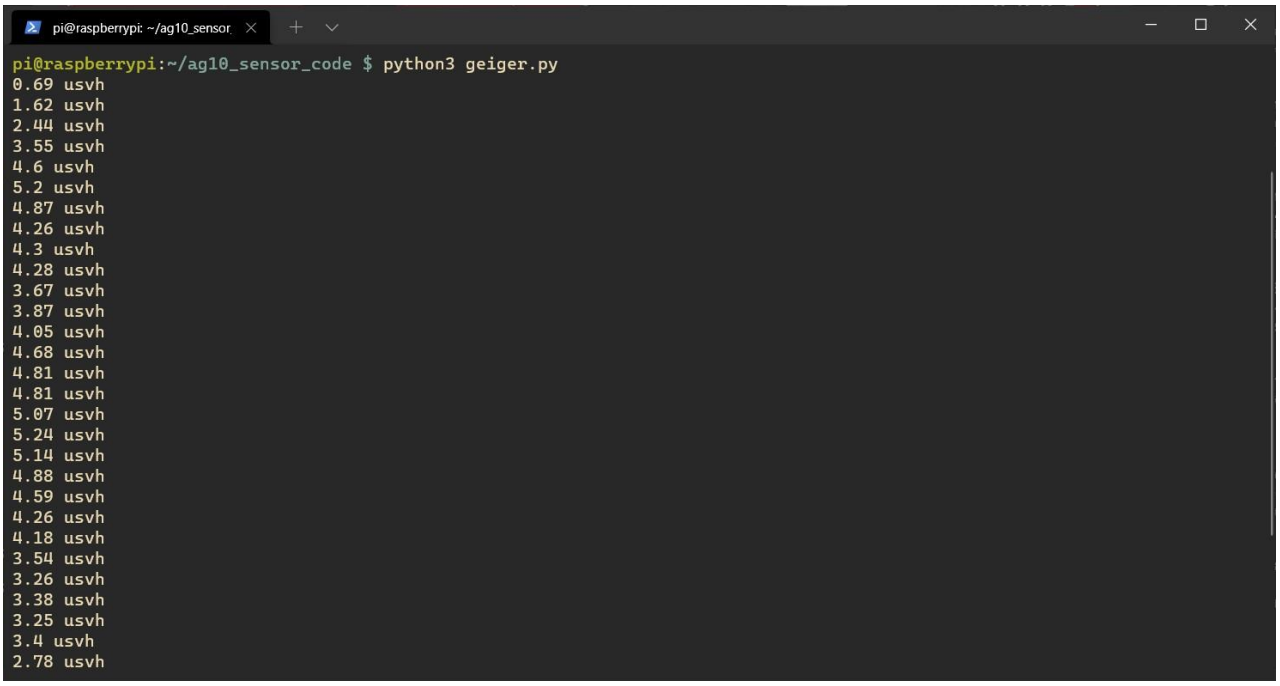

Fig 5.2 : Geiger-Müller Counter Readings printed on the Raspberry Pi's Console

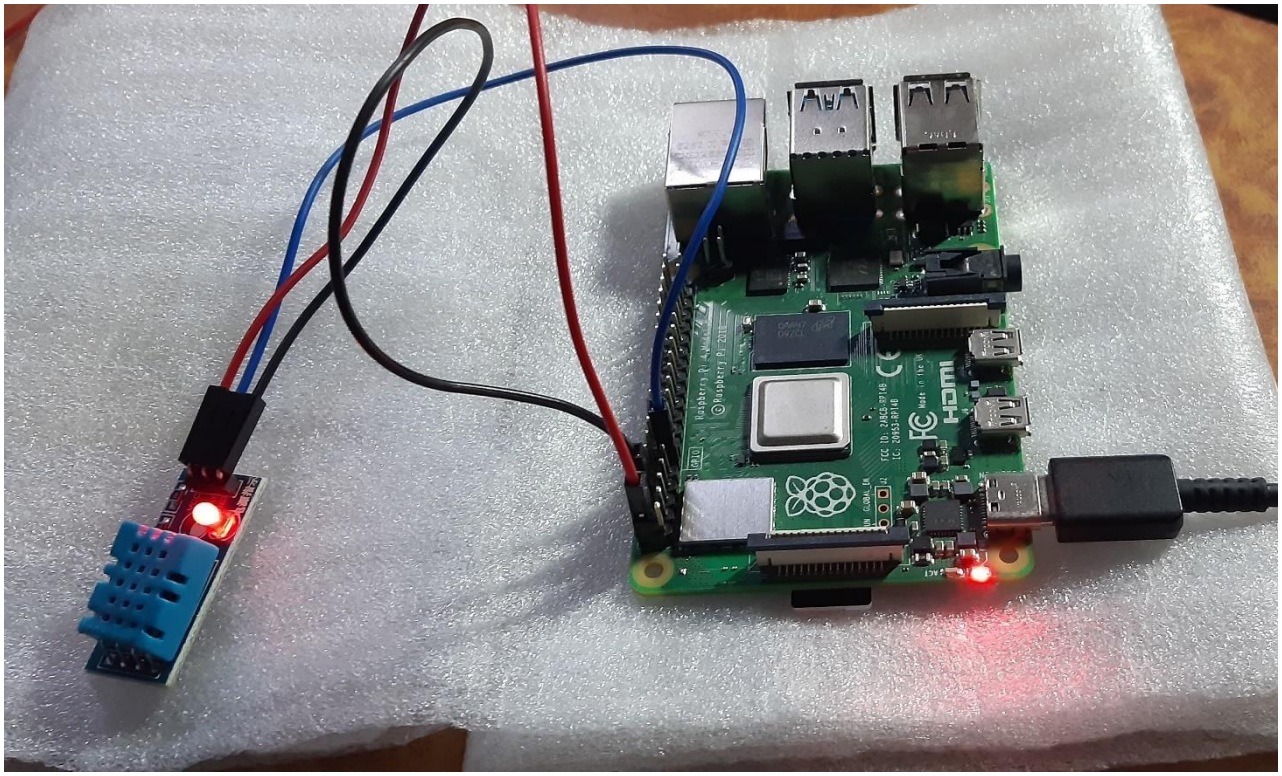

Fig 5.3 : DHT11 Sensor Module interfaced to Raspberry Pi

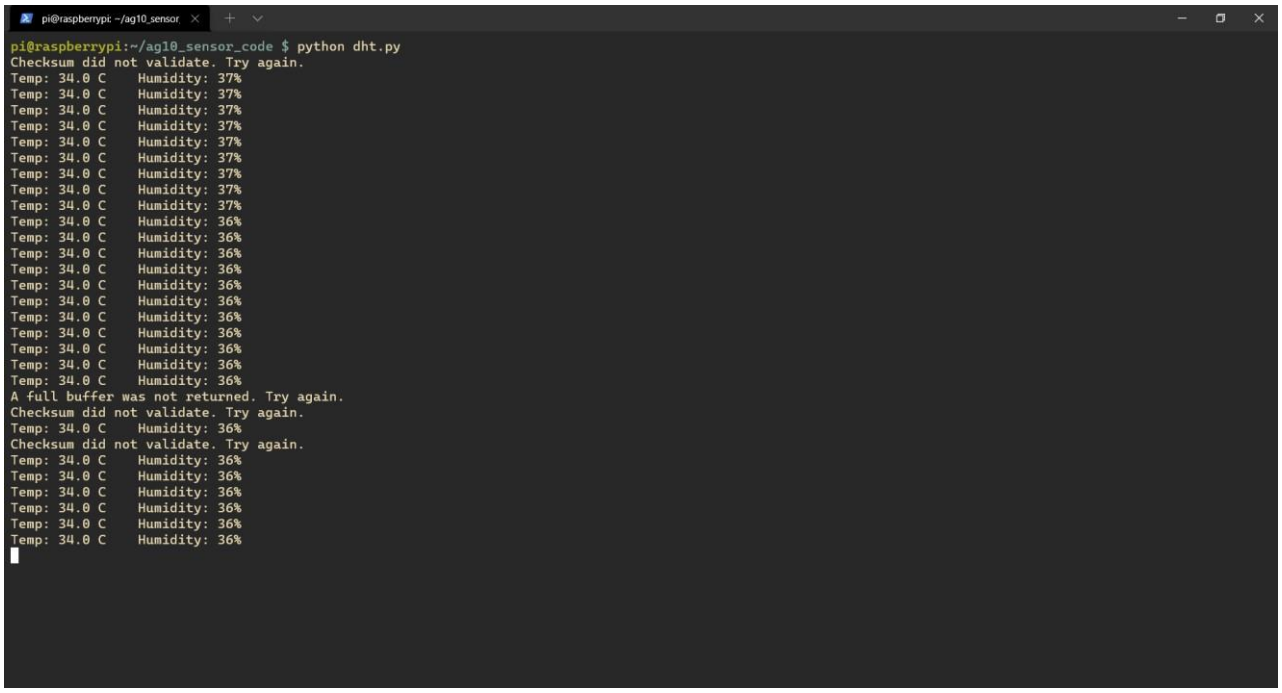

Fig 5.4 : DHT11 Sensor Module readings printed on the Raspberry Pi's Console

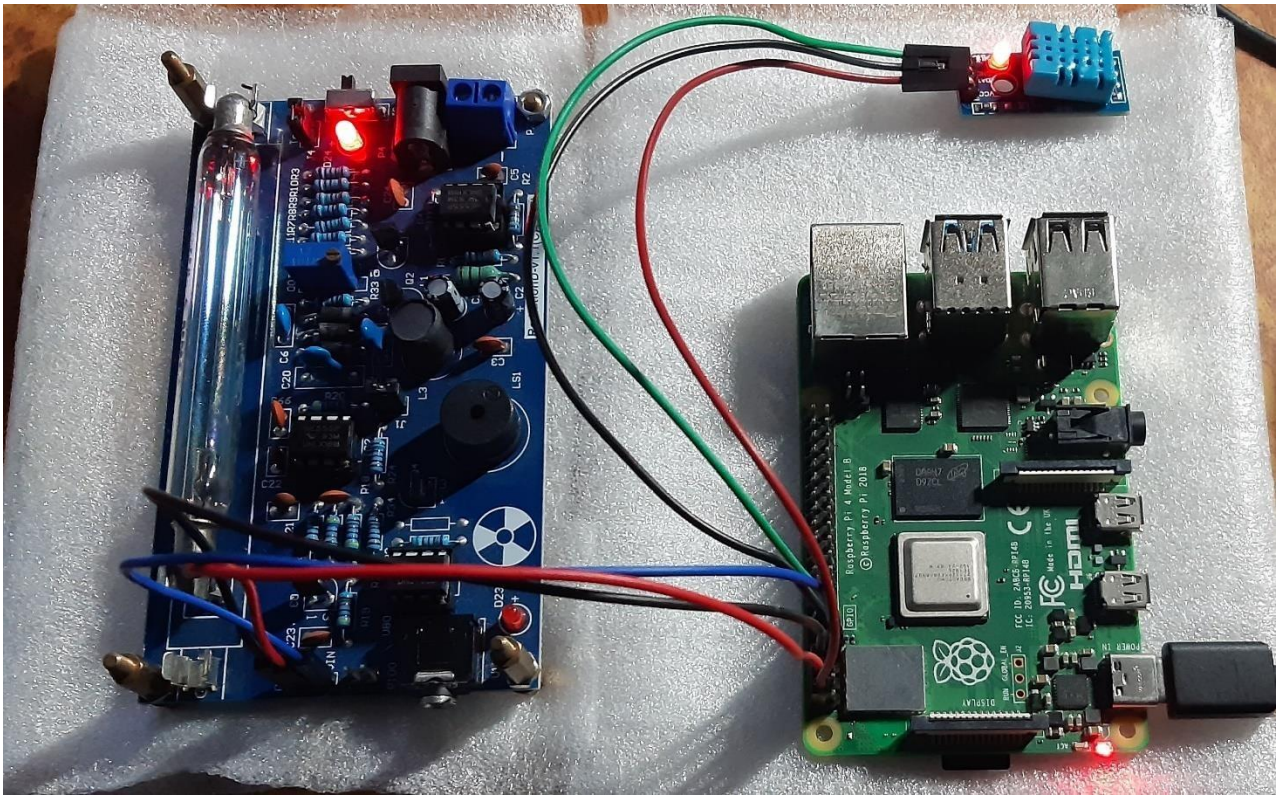

Fig 5.5 : Geiger-Müller Counter and DHT11 Sensor Module interfaced to Raspberry Pi

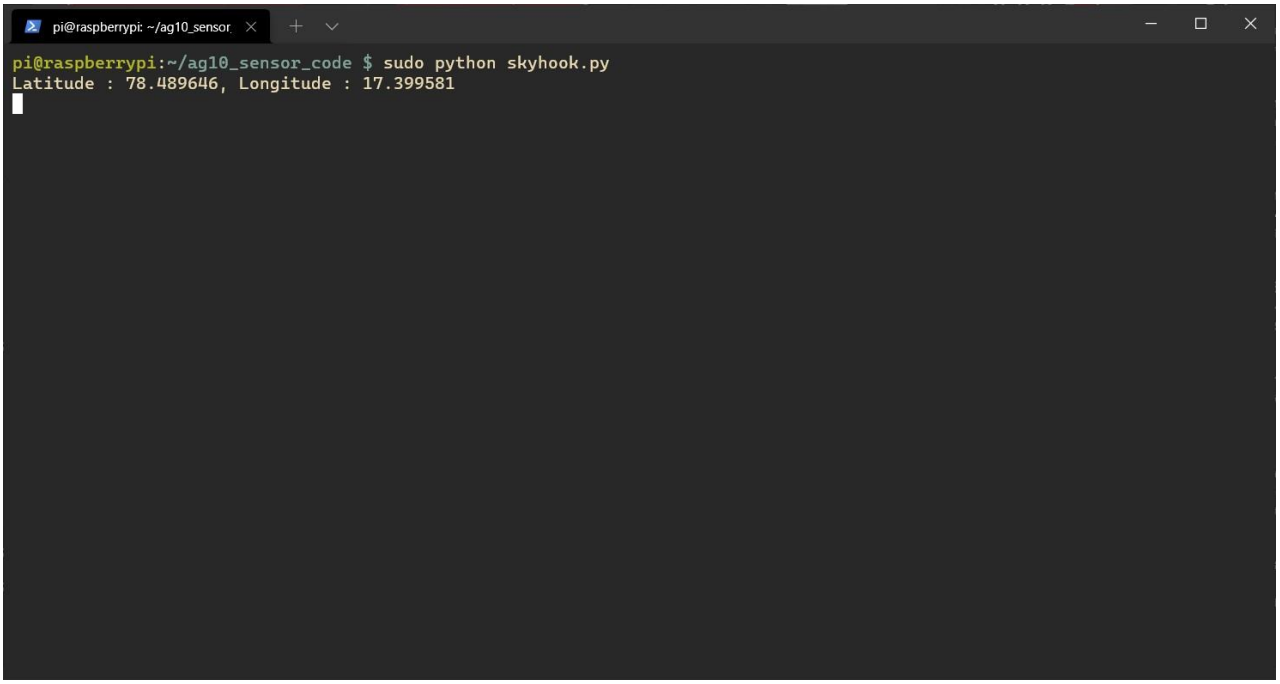

Fig 5.6 : Skyhook Precision Location API output printed on the Raspberry Pi's Console

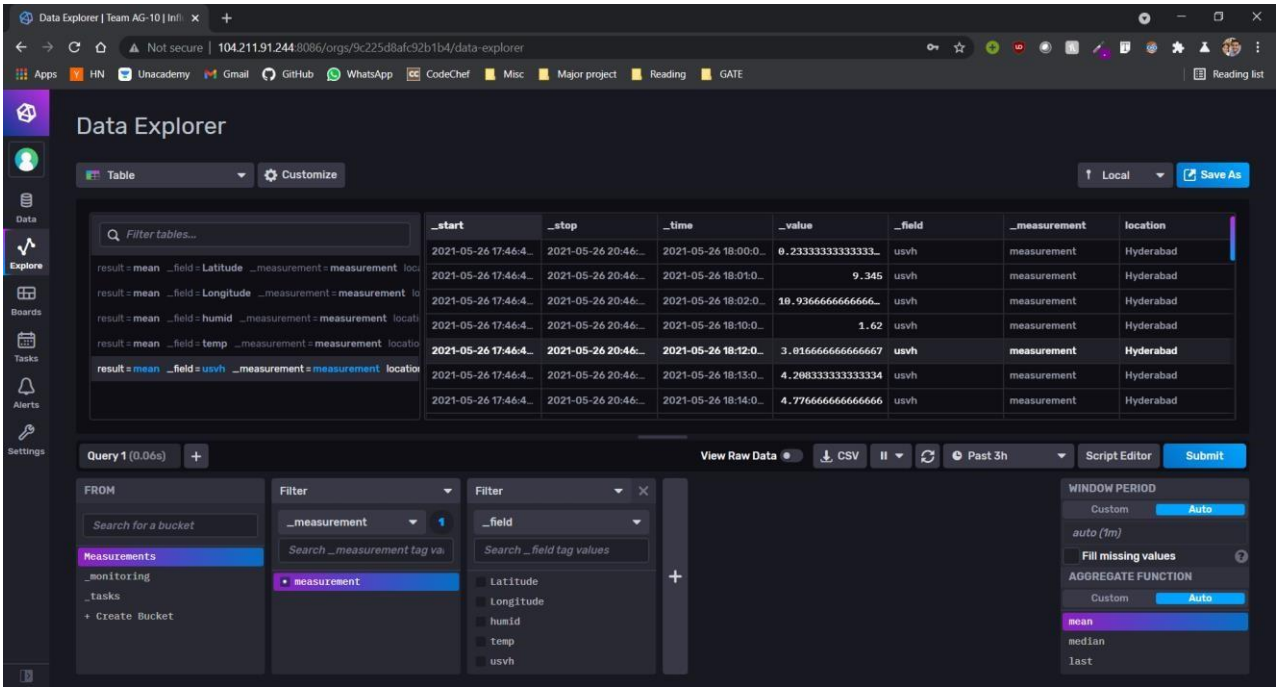

Fig 5.7 : Sensor Readings that were fed into InfluxDB

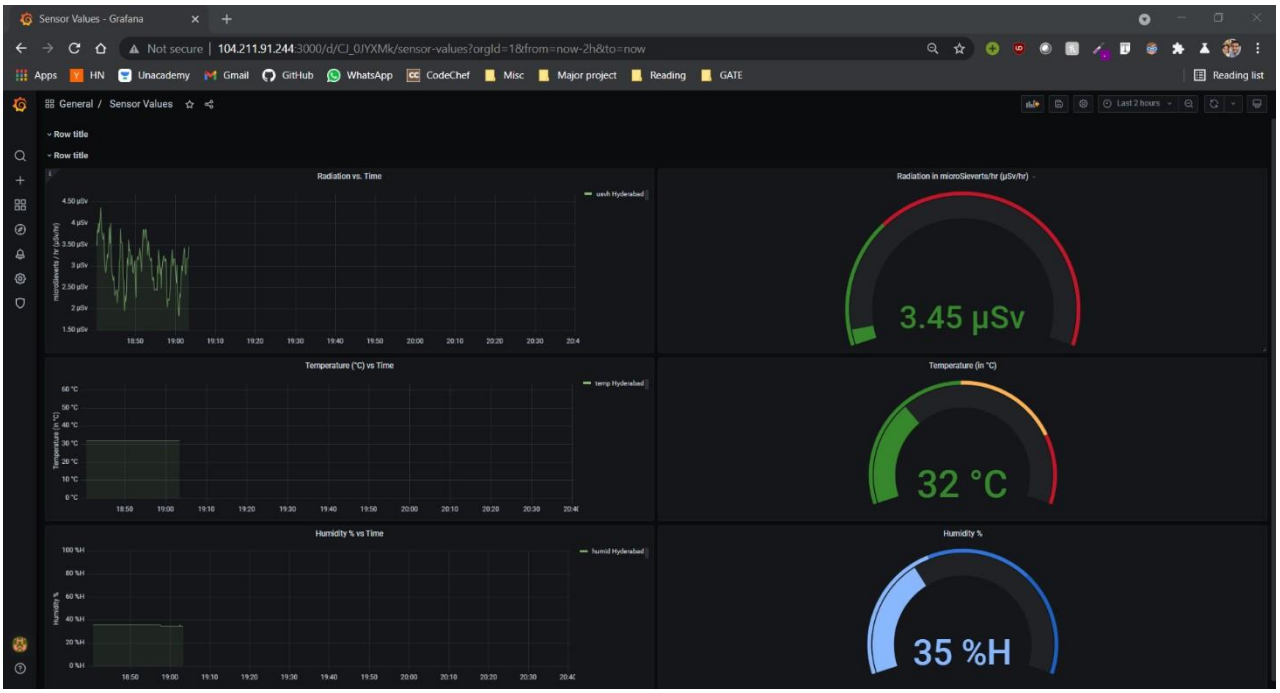

Fig 5.8 : Grafana Visualizations for the three parameters (Radiation, Temperature and Humidity)

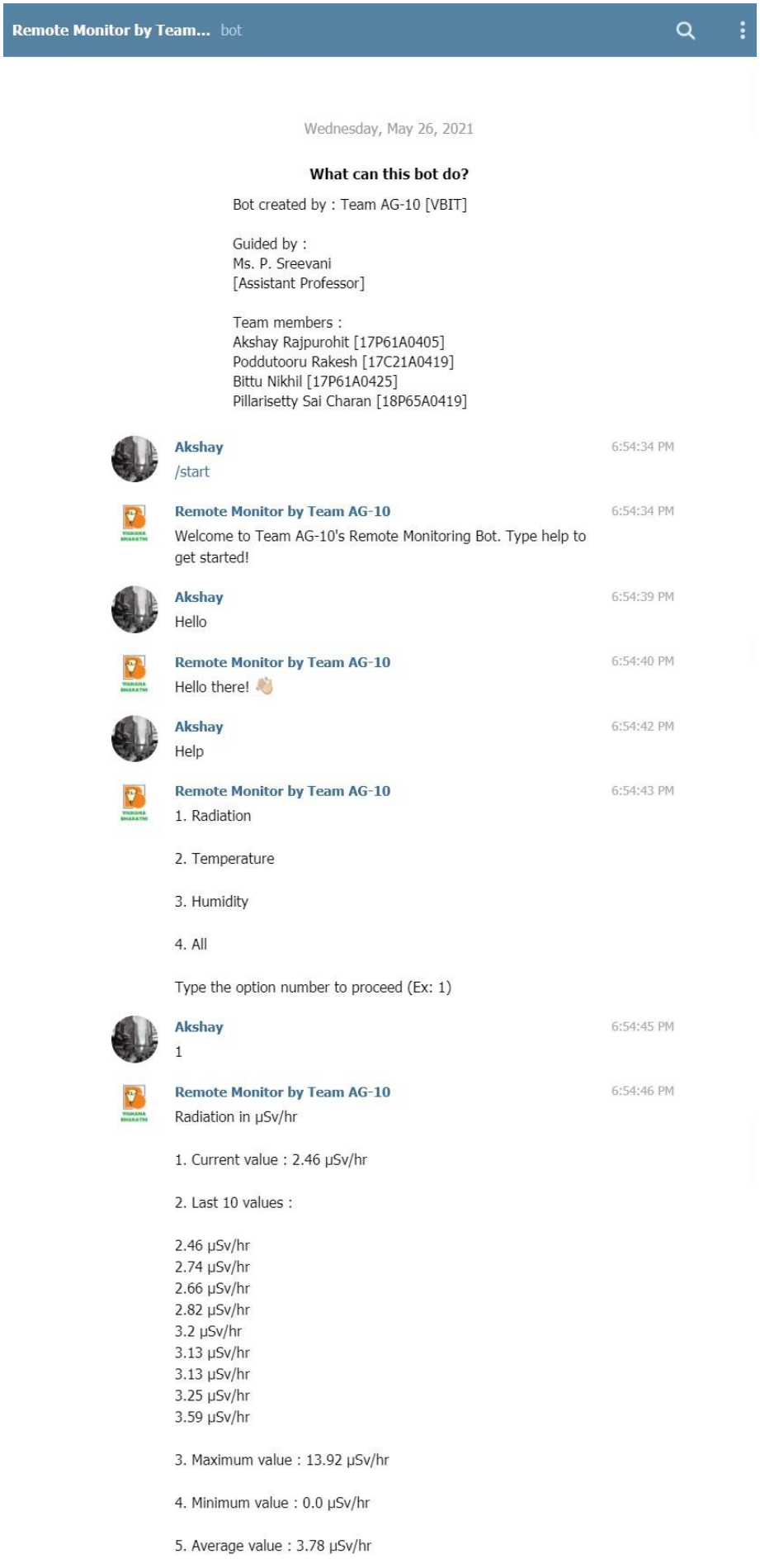

Fig 5.9 Telegram Responder Bot Output (Responses based on User Input)

CHAPTER 6

CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

Thus, our project **"Remote Monitoring of Radiation and Environmental Conditions with Telegram Responder Bot"** has been implemented and deployed according to our requirements. Using the widespread Raspberry Pi platform, the background radiation monitoring system has been developed. The system measures the power of the equivalent radiation and the accumulated dose during a period and outputs the received information to Influx DB deployed on the cloud. When the equivalent dose rate is exceeded, it sends warning signals. The high reliability of the background radiation monitoring system is provided by the rapid response of the system in emergency situations: the exceeding of the permissible power of the equivalent radiation dose, controlling of the battery charge. Similarly, the DHT11 Sensor Module provides us with the Temperature and Humidity of the environment in which the project is deployed. The timely Telegram messages can protect people from getting exposed to excessive radiation and detect its sources and eliminate invisible dangers in proper time. The developed background radiation monitoring system can be used for both stationary and mobile measurements.

6.2 FUTURE SCOPE

Sensors with higher general accuracy (DHT22 instead of DHT11, and SBM2020 instead of J305β) can be employed to get readings that are closer to the real values. The Telegram Responder Bot can be replaced with a WhatsApp Bot since more people use WhatsApp as compared to Telegram. An admin panel along with a user login can be built to control the users who are allowed to view the data (Influx DB or Grafana). Finally, a 4G/5G-enabled GSM HAT can be used with the Raspberry Pi to provide continuous internet connection everywhere (even in the absence of Wi-Fi). And the Raspberry Pi 4 can be replaced with a smaller SBLC (Single Board Linux Computer) to further enhance the portability of the system.

CHAPTER 7

REFERENCES

[1]. Holovatyy, Andriy; Teslyuk, Vasyl; Kryvinska, Natalia; Kazarian, Artem. 2020. "Development of Microcontroller-Based System for Background Radiation Monitoring" Sensors 20, no. 24: 7322. https://doi.org/10.3390/s20247322

[2]. Geiger Counter - Radiation Sensor Board for Arduino and Raspberry Pi https://www.cooking-hacks.com/documentation/tutorials/geiger-counter-radiation-sensorboard-arduino-raspberry-pi-tutorial/index.html

[3]. DHT11 Temperature and Humidity Sensing using Raspberry Pi https://learn.adafruit.com/dht-humidity-sensing-on-raspberry-pi-with-gdocslogging/python-setup

[4]. Skyhook Precision Location SDK API Documentation https://resources.skyhook.com/wiki/type/documentation/precision-location/precisionlocation-sdk-for-mac-windows-linux/218006393

[5]. Creating a Telegram Bot using Python https://www.youtube.com/watch?v=PTAkiukJK7E

CHAPTER 8
APPENDIX

8.1 GEIGER COUNTER CODE

```
# Script that reads values from the CAJOE Geiger Counter

import time
import datetime
import RPi.GPIO as GPIO
from collections import deque
from writeToDB import write
from alert import telegram_bot_sendtext

# use GPIO.setmode(GPIO.BOARD) to use pin numbers
GPIO.setmode(GPIO.BOARD)

# use GPIO.setwarnings(False) to disable warnings
GPIO.setwarnings(False)

counts = deque()
hundredcount = 0
usvh_ratio = 0.00812037037037  # This is for the J305 tube
threshold = 2.00 # Set threshold for alert texts

# This method fires on edge detection (the pulse from the counter board)


def countme(channel):
    global counts, hundredcount
    timestamp = datetime.datetime.now()
    counts.append(timestamp)

    # Every time we hit 100 counts, run count100 and reset
    hundredcount = hundredcount + 1
    if hundredcount >= 100:
        hundredcount = 0
        count100()

# This method runs the servo to increment the mechanical counter


def count100():
    GPIO.setup(12, GPIO.OUT)
    pwm = GPIO.PWM(12, 50)

    pwm.start(4)
    time.sleep(1)
    pwm.start(9.5)
    time.sleep(1)
    pwm.stop()


# Set the input with falling edge detection for geiger counter pulses
GPIO.setup(7, GPIO.IN)
GPIO.add_event_detect(7, GPIO.FALLING, callback=countme)

loop_count = 0

# In order to calculate CPM we need to store a rolling count of events in the
last 60 seconds
# We then calculate the radiation in micro Sieverts per hour by multiplying it
with a factor.
```

```
while True:
    loop_count = loop_count + 1

    try:
        while counts[0] < datetime.datetime.now() -
datetime.timedelta(seconds=60):
            counts.popleft()
    except IndexError:
        pass  # there are no records in the queue.

    if loop_count == 10:

        # Calculate the radiation in micro Sieverts per hour,
        # write it to InfluxDB and reset the count

        usvh = float("{:.2f}".format(len(counts)*usvh_ratio))
        write("usvh", usvh)

        # Check if the usvh value exceeds threshold and if it does,
        # send a telegram text

        if(usvh >= threshold):
            message = f"ALERT! RADIOACTIVITY HAS CROSSED THRESHOLD LIMITS! LAST
VALUE : {usvh} µSv/hr"
            telegram_bot_sendtext(message)

        # print the measurements (if you need it)
        print(f"{usvh} usvh")

        loop_count = 0

    time.sleep(1)
```

8.2 DHT11 SENSOR CODE

```
import time
import board
import adafruit_dht
from writeToDB import write
from alert import telegram_bot_sendtext

# initialize the dht11 device, with data pin connected to:
dhtDevice = adafruit_dht.DHT11(board.D17)
temperature_threshold = 25
humidity_threshold = 25

# you can pass DHT22 use_pulseio=False if you wouldn't like to use pulseio.
# This may be necessary on a Linux single board computer like the Raspberry Pi,
# but it will not work in CircuitPython.
# dhtDevice = adafruit_dht.DHT22(board.D18, use_pulseio=False)

while True:
    try:
        # Write the values to InfluxDB

        temperature = dhtDevice.temperature
        humidity = dhtDevice.humidity

        write("humid", humidity)
        write("temp", temperature)

        # Check if the temperature or humidity value exceeds threshold and if it
does,
        # send a telegram text

        if(temperature >= temperature_threshold):
            message = f"ALERT! TEMPERATURE HAS CROSSED THRESHOLD LIMITS! LAST
VALUE : {temperature} °C"
            telegram_bot_sendtext(message)

        if(humidity >= humidity_threshold):
            message = f"ALERT! HUMIDITY HAS CROSSED THRESHOLD LIMITS! LAST VALUE
: {humidity} %"
            telegram_bot_sendtext(message)

        #print the measurements (if you need it)
        print("Temp: {:.1f} C    Humidity: {}% ".format(temperature, humidity))
    except RuntimeError as error:
        # Errors happen fairly often, DHT's are hard to read, just keep going
        print(error.args[0])
        time.sleep(2.0)
        continue
    except Exception as error:
        dhtDevice.exit()
        raise error

    time.sleep(2.0)
```

8.3 SKYHOOK PRECISION LOCATION CODE

8.3.1. C FUNCTION THAT INTERACTS WITH THE API

```
#include "./wpsapi.h"
#include <stdio.h>
#include <stdlib.h>
double *getLocation()
{
    // initialize the WPS API
    WPS_load();

    // set the API key
    //(found in my.skyhook.com under projects -> the project you're developing)
    const char *key = "YOUR_KEY_HERE";
    WPS_set_key(key);

    // get the location
    WPS_Location *location;
    WPS_ReturnCode rc;

    rc = WPS_location(NULL, WPS_NO_STREET_ADDRESS_LOOKUP, &location);
    if (rc != WPS_OK)
    {
        fprintf(stderr, "*** WPS_location failed (%d)!\n", rc);
    }
    else
    {
        double *coordinates = malloc(sizeof(double) * 2);
        coordinates[0] = location->longitude;
        coordinates[1] = location->latitude;

        // free resources being used by the WPS_location object
        WPS_free_location(location);

        // free all resources being used by the WPS API
        WPS_unload();

        /*
           * Units of coordinates are either in DD (Decimal Degrees) (or) in DMS
(Degrees, Minutes and Seconds)
           * Latitude - North or South of the Equator. If North, then the DD
value is positive, otherwise negative.
           * Longitude - East or West of the Prime Meridian. If East, then the DD
value is positive, otherwise negative.
        */

        // return the coordinates to the calling function
        return coordinates;
    }
}
```

8.3.2. PYTHON DRIVER CODE:

```
import ctypes
import time
from numpy.ctypeslib import ndpointer
from writeToDB import write

while True:

    # use the shared library generated by skyhookpl/getlocation.c
    getloc = ctypes.CDLL("skyhookpl/libgetloc.so")

    # use the ndpointer function provided by numpy, and pass the expected array
datatype and the expected size of the same array
    # use this to set the value of restype, which will help identify the datatype
of the return value of the C function
    getloc.getLocation.restype = ndpointer(dtype=ctypes.c_double, shape=(2,))

    # store the coordinates array of type double returned by the function defined
by getlocation.c as a list in result
    # and return it to the calling function
    result = getloc.getLocation()
    write("Latitude", result[0])
    write("Longitude", result[1])

    # print the values if you need to
    print(f"Latitude : {result[0]}, Longitude : {result[1]}")
    time.sleep(300)
```

8.4 WRITE TO DATABASE CODE

```
import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS
import constants as C

def write(climate_variable, value):

    bucket = C.influxdb_bucket
    org = C.influxdb_org
    token = C.influxdb_token
    url = C.influxdb_url

    client = influxdb_client.InfluxDBClient(
        url=url,
        token=token,
        org=org
    )

    write_api = client.write_api(write_options=SYNCHRONOUS)

    p = influxdb_client.Point("measurement").tag("location", "Hyderabad").field(
        climate_variable, value)
    write_api.write(bucket=bucket, org=org, record=p)
```

8.5 ALERT NOTIFICATION CODE

```
import constants as C
import requests

def telegram_bot_sendtext(bot_message):

   bot_token = C.telegram_token
   bot_chatID = C.telegram_chatid
   for chatid in bot_chatID:
       send_text = 'https://api.telegram.org/bot' + bot_token +
'/sendMessage?chat_id=' + chatid + '&parse_mode=Markdown&text=' + bot_message
   response = requests.get(send_text)
```

8.6 TELEGRAM BOT CODE

8.6.1 MAIN FUNCTION CODE

```
import constants as C
import responses as R
from telegram.ext import *


# print a message to the console indicating that the bot has started running
print("Bot is currently running")


# handler definitions - start, user input (message) and error handlers.


def start_command(update, context):
    # send a nice little welcome message
    update.message.reply_text('Welcome to Team AG-10\'s Remote Monitoring Bot.
Type help to get started!')


def handle_message(update, context):
    # convert text to lower case for easier handling
    text = str(update.message.text).lower()
    # send the message to responses(), which will in turn produce appropriate
output based on the input
    response = R.responses(text)
    # send the output as a reply to the user
    update.message.reply_text(response)

def handle_error(update, context):
    # print error message to the console
    print(f"Update {update} caused error {context.error}")


def main():
    # create updaters and dispatcher using the token provided by BotFather
    updater = Updater(C.telegram_token, use_context=True)
    dispatcher = updater.dispatcher

    # create a /start handler
    dispatcher.add_handler(CommandHandler("start", start_command))

    # handle user messages
    dispatcher.add_handler(MessageHandler(Filters.text, handle_message))

    # handle errors and print them out to console
    dispatcher.add_error_handler(handle_error)

    # constantly check for messages,
    # and remain idle if there are no messages being receieved
    updater.start_polling()
    updater.idle()


# start the process
main()
```

8.6.2 INTERACTION RESPONSES CODE

```
import sys
from query import Query as query
sys.path.append(".")


class Response:

    def help_text(self):
        return """
1. Radiation
2. Temperature
3. Humidity
4. All
Type the option number to proceed (Ex: 1)
"""

    def interact(self, user_messsage):

        # define emojis as variables using unicode strings
        wave = "\U0001F44B"

        # check input and produce appropriate response
        if(user_messsage == "hi" or user_messsage == "hello"):
            return "Hello there! " + wave
        elif(user_messsage == "bye" or user_messsage == "goodbye"):
            return "See you later! " + wave
        elif(user_messsage == "what's up" or user_messsage == "sup"):
            return "I\'m monitoring those sensors! How about you?"

    def undefined(self):
        # produce an appropriate response for undefined messages/behaviour
        response_text = "I don't understand. Please refer to the following help
message:\n"
        response_text += self.help_text()
        return response_text

    def query_handler(self, input):

        # instantiate an object from class Query as 'Q'
        Q = query()

        # convert string to integer for ease of handling
        input = int(input)

        # check input and call appropriate methods
        climate_variables = ["usvh", "temp", "humid", "all"]
        return Q.preprocessor(climate_variables[input - 1])


def responses(input_text):

    # create object of class Response
    response = Response()

    # convert user input to lower-case for ease of processing
    user_messsage = str(input_text).lower()

    # check for casual interaction
    if user_messsage in (
        "hi", "hello",
        "bye", "goodbye",
            "what's up", "sup"):
        return response.interact(user_messsage)
```

```
    # check for help
    if user_messsage in ("help", "help?", "help!", "help."):
        return response.help_text()

    # perform operations based on numeric inputs (mostly running queries on
InfluxDB)
    if int(user_messsage) in range(1, 10):
        return response.query_handler(user_messsage)

    # undefined user input response
    return response.undefined()
```

8.6.3 DATABASE QUERYING CODE

```
import constants as C
import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS


# define the bucket, org, token and url of the InfluxDB instance
bucket = C.influxdb_bucket
org = C.influxdb_org
token = C.influxdb_token
url = C.influxdb_url

# create a client instance
client = influxdb_client.InfluxDBClient(
    url=url,
    token=token,
    org=org
)

# alias the query_api method of client to query_api
query_api = client.query_api()


class Query:

    # function to query the InfluxDB with the given parameters and return results
    def querier(self, days, field):

        # write the query
        query = f'from(bucket: "{bucket}")\
        |> range(start: -{days}d)\
        |> filter(fn: (r) => r._field == "{field}")'

        # send the query to InfluxDB and store the response in result
        result = client.query_api().query(org=org, query=query)

        # process the response and store the data we need in results and return
it to the calling function
        results = []
        for table in result:
            for record in table.records:
                if type(record.get_value()) != str:
                    results.append(record.get_value())
        return results

    def computer(self, input, climate_variable):
        # begin the response text by checking the input
        # and set units accordingly
        units = ""
        if(climate_variable == "usvh"):
            response_text = "Radiation in µSv/hr\n\n"
            units = "µSv/hr"
        elif(climate_variable == "temp"):
            response_text = "Temperature in °C\n\n"
            units = "°C"
        else:
            response_text = "Humidity %\n\n"
            units = "%"

        # Add the current value
        response_text += f"1. Current value : {input[-1]} {units}\n\n"
        # Add the last 10 values
        response_text += "2. Last 10 values : \n\n"
        for i in range(1, 10):
```

```
            response_text += str(input[-1 * i]) + " " + units + "\n"
        # Add the maximum value
        response_text += f"\n3. Maximum value : {max(input)} {units}\n\n"
        # Add the minimum value
        response_text += f"4. Minimum value : {min(input)} {units}\n\n"
        # Add the average value
        response_text += f"5. Average value : {round(sum(input)/len(input), 2)}
{units}\n\n"

        return response_text

    def preprocessor(self, input):
        if(input != "all"):
            return self.computer(self.querier(60, input), input)
        else:
            climate_variables = ["usvh", "temp", "humid"]
            response_text = ""
            for i in climate_variables:
                response_text += self.computer(self.querier(60, i), i)
                response_text += "\n\n\n"
            return response_text
```