

IOT Social Distancing and Monitoring Robot With Face Mask Detection System

Submitted in Partial fulfilment of the Requirements for the Award of

**Bachelor of Engineering Degree in
Electronics and Communication Engineering**

by

S.Mathew Dinesh (38130132)

S.Abilash (38130701)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

**SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING
SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

May- 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

**DEPARTMENT OF ELECTRONICS AND
COMMUNICATIONENGINEERING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **S.MathewDinesh (38130132)**, **S.Abilash (38130701)** who carried out the Project entitled **“IOT Social Distancing and Monitoring Robot With Face Mask Detection System”**. Under my supervision from SEPTEMBER 2021 to MAY2022.

Internal Guide

Mr.Vinothkumar.C

DEPARTMENT OF EIE

Head of the Department

Dr. T. RAVI, M.E., Ph.D.

Submitted for Viva voce examination held on _____

Internal Examiner

External Examiner

DECLARATION

We **S.Mathew Dinesh (38130132), S.Abilash (38130701)**, hereby declare that the Project Report entitled “**IOT Social Distancing and Monitoring Robot With Face Mask Detection System**” done by us under the guidance of Mr.Vinothkumar.C (Department of EIE) is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

1)

2)

**SIGNATURE OF THE
CANDIDATE**

DATE :

PLACE :

ACKNOWLEDGEMENT

We are pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our sincere thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D. Dean, School of Electrical and Electronics Engineering** and **Dr. T. RAVI, M.E., Ph.D. Head of Department, Department of Electronics and Communication Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Mr.Vinothkumar.C**(Department of EIE) for their valuable guidance, suggestions and constant encouragement paved way for the successful completion of the project work.

We wish to express our sincere thanks to all teaching and Non-teaching staff members of the Department of **Electronics and Communication Engineering** who were helpful in many ways for the completion of the Project.

ABSTRACT

Social distancing is of key importance during the current pandemic. It helps limit the spread of COVID by observing distance between disease spreading individuals. Now it is not possible to station a person 24×7 at each queue to monitor social distancing violations. Banks, Public Offices, Malls, Schools, Theatre, etc., usually see long queues for hours every day. To ensure social distancing in queues we hereby design a social distancing monitoring robot. The robot consists of a 4 wheel design system used to drive the robotic vehicle. It makes use of a line following principle to constantly move along with the queue and monitor for social distancing violations. The robotic uses IR sensing to travel along with the queue to and fro in order to detect violations. The robot is now equipped with obstacle detecting ultrasonic sensor in order to detect obstacles in the vehicle path. The robotic vehicle uses another ultrasonic sensor for detecting distance between 2 individuals in a queue. If any 2 individuals are found having less than 3 feet distance between them, the robot instantly sounds a buzzer and alert to inform about the violation. Also it sends alerts of these violations along with a camera picture using WiFi over IoT to inform the higher authorities/head office to update them about violations with proof so instant disciplinary action can be taken. And also, it is used to measure the temperature of the people in the queue and face mask detection using AI is implemented. Thus this project allows for automatic maintaining social distancing in queues and temperature and face mask monitoring system to help prevent spread of the virus.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1	INTRODUCTION	
	1.1 EMBEDDED SYSTEM	01
	1.2 INTERNET OF THINGS	07
2	LITERATURE REVIEW	
	2.1 LITERATURE SURVEY	14
3	PROJECT DESCRIPTION	
	3.1 PROPOSED SYSTEM	17
	3.2 BLOCK DIAGRAM	18
	3.3 HARDWARE REQUIREMENTS	18
4	HARDWARE DESCRIPTION	
	4.1 RASPBERRY PI	20
	4.2 ULTRASONIC SENSORS	24
	4.3 TEMPERATURE SENSOR	30
	4.4 NODE MCU	35
	4.5 BUZZER	38
	4.6 POWER SUPPLY UNIT	44

5	SOFTWARE DESCRIPTION	
	5.1 RASPBIAN OS	50
	5.2 PYTHON	52
	5.3 PHP	78
6	CONCLUSION	81
7	RESULTS	
	7.1 CIRCUIT CONNECTIONS	82
	7.2 PROGRAM	84
	7.3 OUTPUT RESULTS	90
	REFERENCE	91

CHAPTER-1

INTRODUCTION

Nowadays Social distancing is of key importance during the current pandemic. It helps limit the spread of covid by observing distance between disease spreading individuals. Now it is not possible to station a person 24×7 at each queue to monitor social distancing violations. Banks, Public Offices, Malls, Schools, Theatres etc usually see long queues for hours every day to ensure social distancing in queues we hereby design a social distancing monitoring robot. A key issue is developing guidelines and methods to enforce these social distance constraints in public or private gatherings at indoor or outdoor locations. This gives rise to many challenges including, framing reasonable rules that people can follow when they use public places such as supermarkets, pharmacies, railway and bus stations, spaces for recreation and essential work, and how people can be encouraged to follow the new rules. In addition, it is also crucial to detect when such rules are breached so that appropriate counter-measures can be employed. Detecting social distancing breaches could also help in contact tracing.

1.1 EMBEDDED SYSTEM:

An embedded system is a combination of computer hardware and software, either fixed in capability or programmable, designed for a specific function or functions within a larger system. Industrial machines, agricultural and process industry devices, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines and toys, as well as mobile devices, are possible locations for an embedded system.

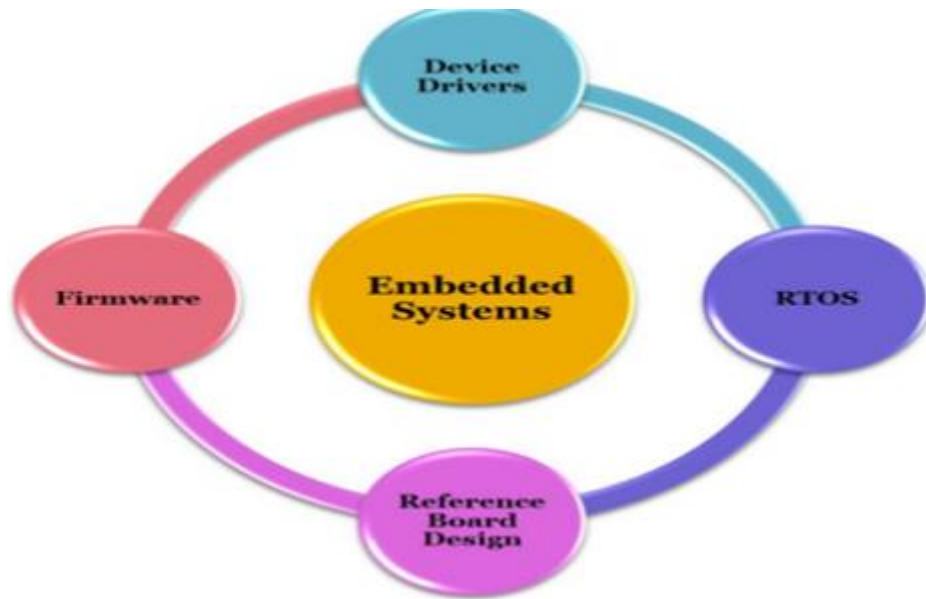
Embedded systems are computing systems, but they can range from having no user interface (UI) -- for example, on devices in which the

system is designed to perform a single task -- to complex graphical user interfaces (GUIs), such as in mobile devices. User interfaces can include buttons, LEDs, touchscreen sensing and more. Some systems use remote user interfaces as well.

Embedded system hardware (microprocessor-based, microcontroller-based)

Embedded system hardware can be microprocessor- or microcontroller-based. In either case, an integrated circuit is at the heart of the product that is generally designed to carry out computation for real-time operations. Microprocessors are visually indistinguishable from microcontrollers, but while the microprocessor only implements a central processing unit (CPU) and, thus, requires the addition of other components such as memory chips, microcontrollers are designed as self-contained systems.

Microcontrollers include not only a CPU, but also memory and peripherals such as flash memory, RAM or serial communication ports. Because microcontrollers tend to implement full (if relatively low computer power) systems, they are frequently put to use on more complex tasks. For example, microcontrollers are used in the operations of vehicles, robots, medical devices and home appliances, among others. At the higher end of microcontroller capability, the term system on a chip (SoC) is often used, although there's no exact delineation in terms of RAM, clock speed and so on.



The embedded market was estimated to be in excess of \$140 billion in 2013, with many analysts projecting a market larger than \$20 billion by 2020. Manufacturers of chips for embedded systems include many mainstays of the computer world, such as Apple, IBM, Intel and Texas Instruments, as well as numerous other companies less familiar to those outside the field. Arm has been a highly influential vendor in this space. The company began as an outgrowth of Acorn, a U.K. maker of early PCs. Arm chips, produced under license by other companies, are based on the reduced instruction set computer (RISC) architecture and are often used in mobile phones; they remain the most widely deployed SoC in the embedded world, with billions of units fielded.



Embedded system software

A typical industrial microcontroller is unsophisticated compared to the typical enterprise desktop computer and generally depends on a simpler, less-memory-intensive program environment. The simplest devices run on bare metal and are programmed directly using the chip CPU's machine code language.

Often, embedded systems use operating systems or language platforms tailored to embedded use, particularly where real-time operating environments must be served. At higher levels of chip capability, such as those found in SoCs, designers have increasingly decided the systems are generally fast enough and the tasks tolerant of slight variations in reaction time that near-real-time approaches are suitable. In these instances,

stripped-down versions of the Linux operating system are commonly deployed, although other operating systems have been pared down to run on embedded systems, including Embedded Java and Windows IoT (formerly Windows Embedded).

Generally, storage of programs and operating systems on embedded devices make use of either flash or rewritable flash memory.

Embedded firmware

The firmware on embedded systems, referred to as embedded firmware, is specific software written into the memory of a device that serves the purpose of ROM, but can be updated more easily. Firmware can be stored in non-volatile memory devices including ROM, programmable ROM, erasable PROM or flash memory. Embedded firmware is used to control various device and system functions, for example, telling the device how to communicate with other devices, perform specific functions and provide input and output functionality.

The delineation between the terms embedded firmware and embedded software are blurring, but embedded software often refers to the only code running on a piece of hardware, while firmware can also refer to the chip that houses a device's basic input/output system (BIOS) or Unified Extensible Firmware Interface (UEFI), which connect software and a system's operating system.

Embedded systems vs. VLSI

Very-large-scale integration, or VLSI, is a term that describes the complexity of an integrated circuit. VLSI is the process of embedding hundreds of thousands of transistors into a chip, whereas LSI (large-scale integration) microchips contain thousands of transistors, MSI (medium-scale integration) contain hundreds of transistors, and SSI (small-scale integration) contain tens of transistors. ULSI, or ultra-large-scale integration, refers to placing millions of transistors on a chip.

VLSI circuits are common features of embedded systems. Many ICs in embedded systems are VLSI, and the use of the VLSI acronym has largely fallen out of favor.

Debugging embedded systems

One area where embedded systems part ways with the operating systems and development environments of other, larger-scale computers is in the area of debugging. While programmers working with desktop computer environments have systems that can run both the code being developed and separate debugger applications that monitor the actions of the development code as it is executed, embedded system programmers generally cannot.

Some programming languages run on microcontrollers with enough efficiency that rudimentary interactive debugging is available directly on the chip. Additionally, processors often have CPU debuggers that can be controlled -- and, thus, control program execution -- via a JTAG or similar debugging port.

In many instances, however, programmers of embedded systems need tools that attach a separate debugging system to the target system via a

serial or other port. In this scenario, the programmer can see the source code on the screen of a conventional personal computer just as would be the case in the debugging of software on a desktop computer. A separate, frequently used approach is to run software on a PC that emulates the physical chip in software, thus making it possible to debug the performance of the software as if it were running on an actual, physical chip.

Broadly speaking, embedded systems have received more attention to testing and debugging because a great number of devices using embedded controls are designed for use in situations where safety and reliability are top priorities.

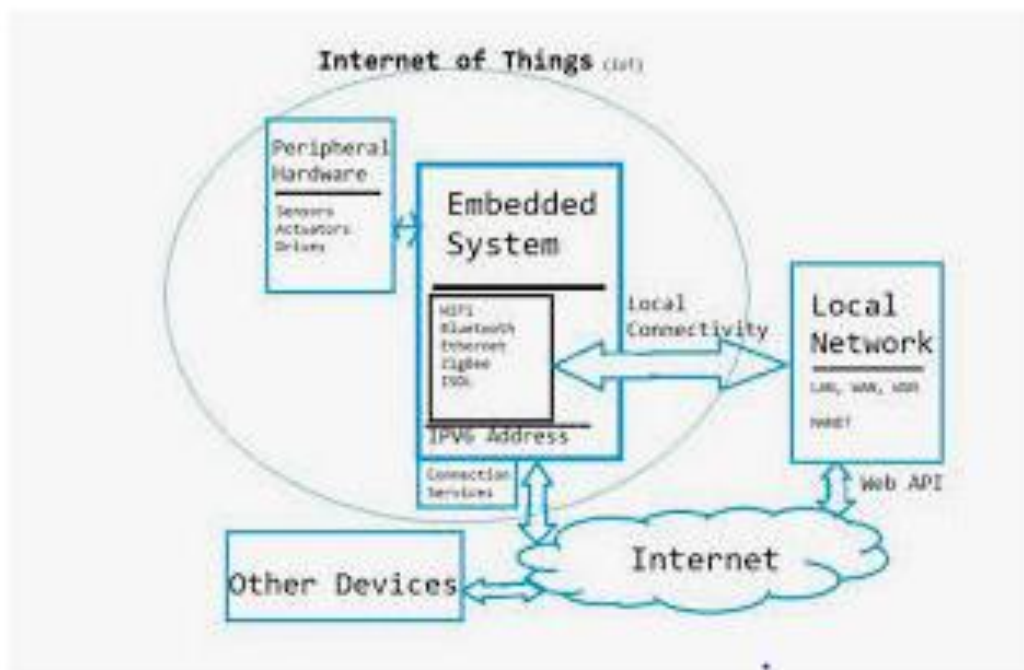
The internet of things builds on an embedded systems base

While some embedded systems can be relatively simple, a growing number either supplant human decision-making or offer capabilities beyond what a human could provide. For instance, some aviation systems, including those used in drones, are able to integrate sensor data and act upon that information faster than a human could, permitting new kinds of operating features.

1.2 INTERNET OF THINGS:

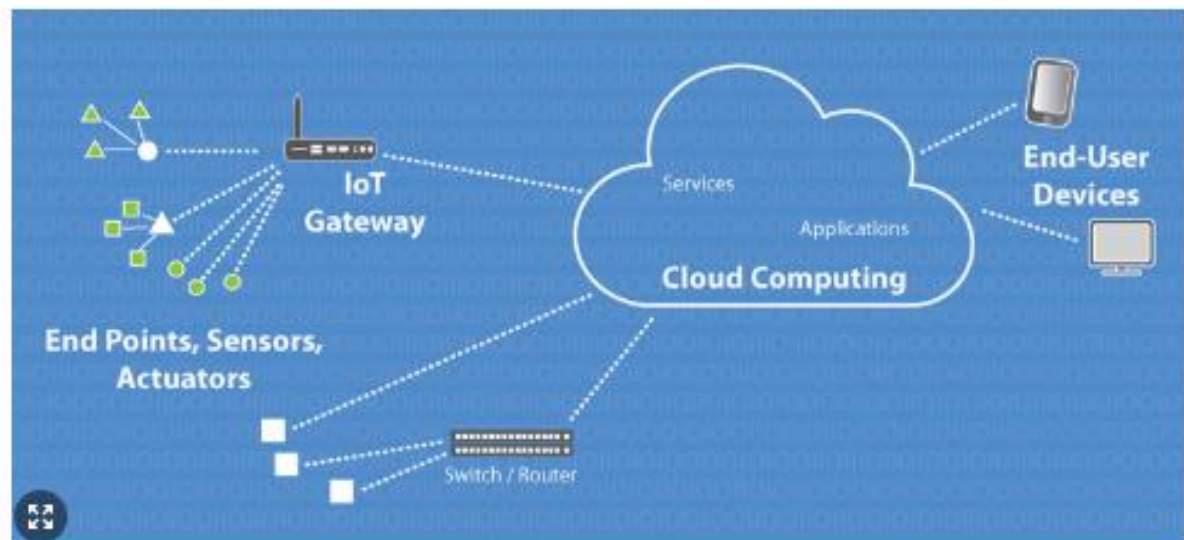
- The term Internet of Things generally refers to scenarios where network connectivity and computing capability extends to objects, sensors and everyday items not normally considered computers, allowing these devices to generate, exchange and consume data with minimal human intervention. There is, however, no single, universal definition.

- **Enabling Technologies:** The concept of combining computers, sensors, and networks to monitor and control devices has existed for decades. The recent confluence of several technology market trends, however, is bringing the Internet of Things closer to widespread reality. These include Ubiquitous Connectivity, Widespread Adoption of IP-based Networking, Computing Economics, Miniaturization, Advances in Data.
- **Connectivity Models:** IoT implementations use different technical communications models, each with its own characteristics. Four common communications models described by the Internet Architecture Board include: Device-to-Device, Device-to-Cloud, Device-to-Gateway, and Back-End Data-Sharing. These models highlight the flexibility in the ways that IoT devices can connect and provide value to the user.



IoT devices are implemented using both hardware and software components. Dedicated hardware components are used to implement the interface with the physical world, and to perform tasks which are more

computationally complex. Microcontrollers are used to execute software that interprets inputs and controls the system. This module discusses the roles of both the hardware and software components in the system. The functions of common hardware components are described and the interface between the software and hardware through the microcontroller is explained. IoT devices often use an operating system to support the interaction between the software and the microcontroller. We will define the role of an operating system in an IoT device and how an IoT operating system differs from a standard one.



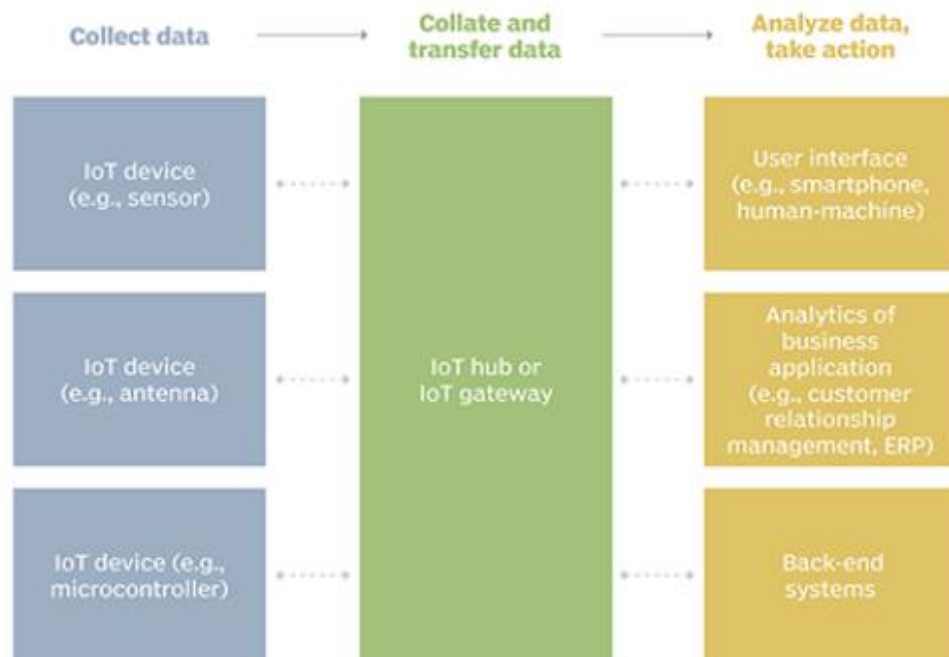
How IoT works

An IoT ecosystem consists of web-enabled smart devices that use embedded processors, sensors and communication hardware to collect, send and act on data they acquire from their environments. IoT devices share the sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be

analyzed or analyzed locally. Sometimes, these devices communicate with other related devices and act on the information they get from one another. The devices do most of the work without human intervention, although people can interact with the devices -- for instance, to set them up, give them instructions or access the data.

The connectivity, networking and communication protocols used with these web-enabled devices largely depend on the specific IoT applications deployed.

Example of an IoT system



Benefits of IoT

The internet of things offers a number of benefits to organizations, enabling them to:

- monitor their overall business processes;

- improve the customer experience;
- save time and money;
- enhance employee productivity;
- integrate and adapt business models;
- make better business decisions; and
- generate more revenue.

IoT encourages companies to rethink the ways they approach their businesses, industries and markets and gives them the tools to improve their business strategies.

Consumer and enterprise IoT applications

There are numerous real-world applications of the internet of things, ranging from consumer IoT and enterprise IoT to manufacturing and industrial IoT (IIoT). IoT applications span numerous verticals, including automotive, telco, energy and more.

In the consumer segment, for example, smart homes that are equipped with smart thermostats, smart appliances and connected heating, lighting and electronic devices can be controlled remotely via computers, smartphones or other mobile devices.

Wearable devices with sensors and software can collect and analyze user data, sending messages to other technologies about the users with the aim of making users' lives easier and more comfortable. Wearable devices are also used for public safety -- for example, improving first responders' response times during emergencies by providing optimized routes to a

location or by tracking construction workers' or firefighters' vital signs at life-threatening sites.

In healthcare, IoT offers many benefits, including the ability to monitor patients more closely to use the data that's generated and analyze it. Hospitals often use IoT systems to complete tasks such as inventory management, for both pharmaceuticals and medical instruments.

CHAPTER-2

LITERATURE SURVEY

COVID-19 FACE MASK DETECTION USING TENSORFLOW, KERAS AND OPENCV

COVID-19 pandemic has become a global challenge faced by people all over the world. Social distancing has been proved to be an effective practice to reduce the spread of COVID-19. Against this backdrop, we propose that the surveillance robots can not only monitor but also promote social distancing. Robots can be flexibly deployed and they can take precautionary actions to remind people of practicing social distancing. In this paper, we introduce a fully autonomous surveillance robot based on a quadruped platform that can promote social distancing in complex urban environments. Specifically, to achieve autonomy, we mount multiple cameras and a 3D LiDAR on the legged robot. The robot then uses an onboard real-time social distancing detection system to track nearby pedestrian groups. Next, the robot uses a crowd-aware navigation algorithm to move freely in highly dynamic scenarios. The robot finally uses a crowd-aware routing algorithm to effectively promote social distancing by using human-friendly verbal cues to send suggestions to over-crowded pedestrians. We demonstrate and validate that our robot can be operated autonomously by conducting several experiments in various urban scenarios.

COVID SURVEILLANCE ROBOT: MONITORING SOCIAL DISTANCING CONSTRAINTS IN INDOOR SCENARIOS

Observing social/physical distancing norms between humans has become an indispensable precaution to slow down the transmission of COVID-19. We present a novel method to automatically detect pairs of humans in a crowded scenario who are not maintaining social distancing, i.e. about 2 meters of space between them using an autonomous mobile

robot and existing CCTV (Closed-Circuit TeleVision) cameras. The robot is equipped with commodity sensors, namely an RGB-D (Red Green Blue—Depth) camera and a 2-D lidar to detect social distancing breaches within their sensing range and navigate towards the location of the breach. Moreover, it discreetly alerts the relevant people to move apart by using a mounted display. In addition, we also equip the robot with a thermal camera that transmits thermal images to security/healthcare personnel who monitors COVID symptoms such as a fever. In indoor scenarios, we integrate the mobile robot setup with a static wall-mounted CCTV camera to further improve the number of social distancing breaches detected, accurately pursuing walking groups of people etc. We highlight the performance benefits of our robot + CCTV approach in different static and dynamic indoor scenarios.

IOT SOCIAL DISTANCING & MONITORING ROBOT FOR QUEUE

Social distancing is of key importance during the current pandemic. It helps limit the spread of covid by observing distance between disease spreading individuals. Now it is not possible to station a person 24×7 at each queue to monitor social distancing violations. Banks, Public Offices, Malls, Schools, Theatres etc usually see long queues for hours every day. To ensure social distancing in queues we hereby design a social distancing monitoring robot. The robot consists of a 4 wheel design system used to drive the robotic vehicle. It makes use of a line following principle to constantly move along with the queue and monitor for social distancing violations. The robotic uses IR sensing to travel along with the queue to and fro in order to detect violations. The robot is now equipped with obstacle detecting ultrasonic sensor in order to detect obstacles in the vehicle path. The robotic vehicle uses another ultrasonic sensor for detecting distance between 2 individuals in a queue. It any 2 individuals

are found having less than 3 feet distance between them, the robot instantly sounds a buzzer and alert to inform about the violation. Also it sends alerts of these violations along with a camera picture using wifi over iot to inform the higher authorities/head office to update them about violations with proof so instant disciplinary action can be taken. Thus this project allows for automatic maintaining social distancing in queues to help prevent spread of the virus.

IOT SOCIAL DISTANCING & MONITORING ROBOT FOR QUEUE

During the current pandemic, social distancing is crucial because it helps limit the development of Covid due to the perceived distance between people who spread the disease. Currently, there is no option to place one person in each row 24/7 to monitor social distancing violations. Banks, authorities, shopping centers, schools, theatres, etc. There are often long queues for several hours a day. To verify the queuing social distance, we want to develop a social distance robot.

AGENT-BASED SIMULATION FOR WEEKEND-EXTENSION STRATEGIES TO MITIGATE INFLUENZA OUTBREAKS

Non-pharmaceutical strategies are vital in curtailing impacts of influenza and have been intensively studied in public health. However, few strategies have explicitly utilized the weekend effect, which has been widely reported to be capable of reducing influenza infections. This study aims to explore six weekend-extension strategies against seasonal and pandemic flu outbreaks. The weekend-extension strategies were designed to extend regular two-day weekend by one, two and three days, respectively, and in combination with either a continuous or discontinuous pattern. Their effectiveness was evaluated using an

established agent-based spatially explicit simulation model in the urbanized area of Buffalo, NY, US.

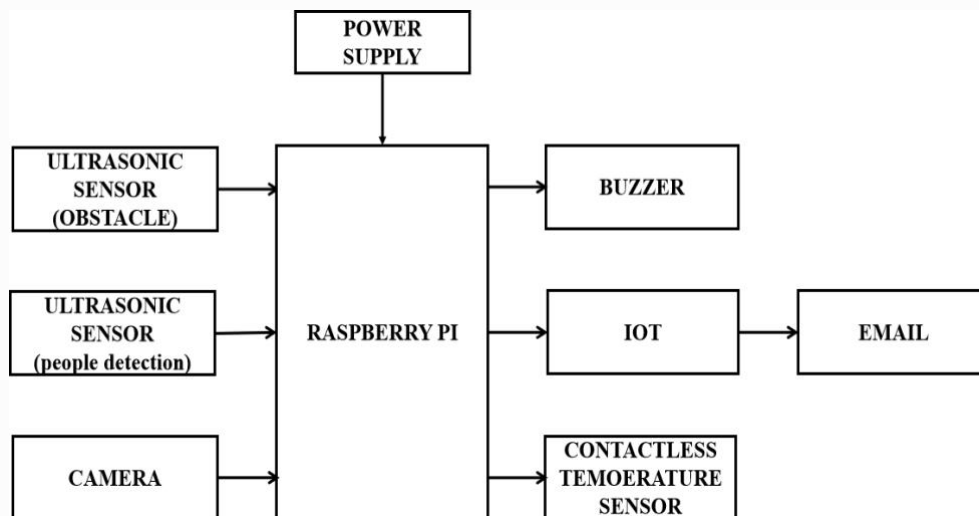
CHAPTER-3

PROJECT DESCRIPTIONS

3.1 PROPOSED SYSTEM:

- The robot will have a 4-wheel design system that will be utilized to operate the robotic vehicle. It employs the line-following concept to keep up with the queue and monitor for social distance infractions. This robot will utilize infrared sensing to travel beside the queue in order to identify human transgressions.
- To identify obstructions in the vehicle's route, the robot now has an obstacle-detecting ultrasonic sensor. Another ultrasonic sensor is used by the robotic vehicle to detect the distance between two people in a queue.
- If any two individuals are found to have less than 3 feet between them, the robot instantly starts to sound and alerts them about the violation.
- Also, it will send an SMS alerts of these violations along that it send a alert in IoT using Wi-Fi to inform the higher authorities to update them about violations with proof so instant disciplinary action can be taken against those humans and send an images in mail.
- And the face mask has been detected for the every individuals. Thus, this project allows for automatic maintenance of social distance in queues to help prevent the spread of the SARS COVID-19 virus.

3.2 BLOCK DIAGRAM:



3.3 HARDWARE REQUIREMENTS

- RASPBERRY PI
- ULTRASONIC SENSOR
- CAMERA
- BUZZER
- IOT
- CONTACTLESS TEMPERATURE SENSOR

3.4 SOFTWARE REQUIREMENTS

- RASPBIAN OS
- PYTHON
- PHP

- OPEN CV
- KERAS
- TENSORFLOW

CHPATER-4

HARDWARE DESCRIPTIONS

4.1 RASPBERRY PI:

The **Raspberry Pi** is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside its target market for uses such as robotics. It does not include peripherals (such as keyboards and mice) and cases. However, some accessories have been included in several official and unofficial bundles.

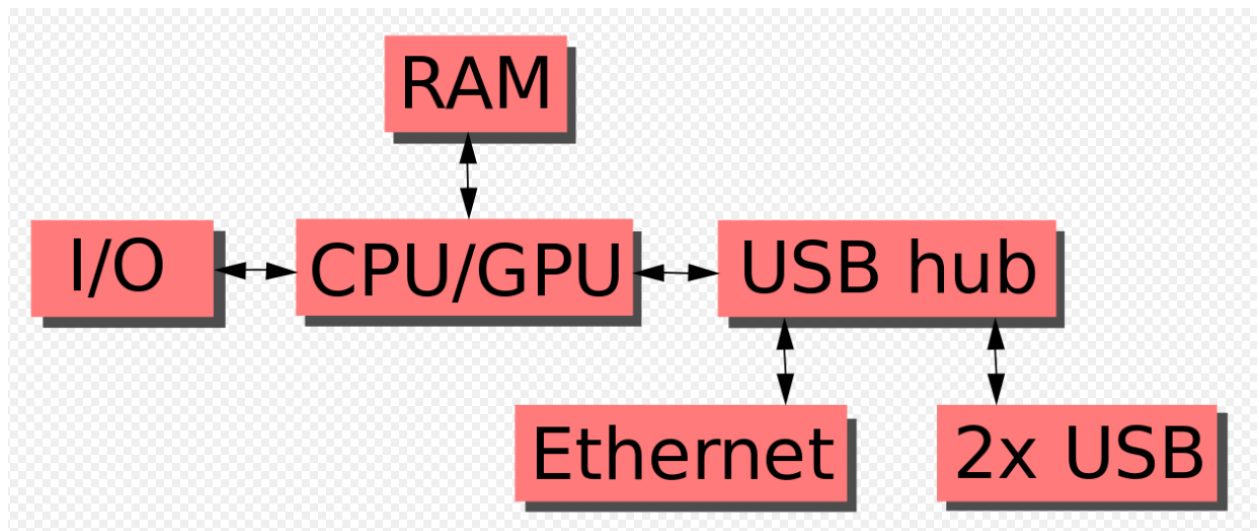
The organization behind the Raspberry Pi consists of two arms. The first two models were developed by the Raspberry Pi Foundation. After the Pi Model B was released, the Foundation set up Raspberry Pi Trading, with Eben Upton as CEO, to develop the third model, the B+. Raspberry Pi Trading is responsible for developing the technology while the Foundation is an educational charity to promote the teaching of basic computer science in schools and in developing countries.

HARDWARE:

The Raspberry Pi hardware has evolved through several versions that feature variations in memory capacity and peripheral-device support.

This block diagram describes Model B and B+; Model A, A+, and the Pi Zero 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.



Processor

- The Broadcom BCM2835 SoC used in the first generation Raspberry Pi includes a 700 MHz ARM11 76JZF-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 1176JZ(F)-S is the same CPU used in the original iPhone,^[22] 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+ uses a Broadcom BCM2837B0 SoC with a 1.4 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache.

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.^[23] In parallelised 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 ARM Cortex-A53 processor, is described as having ten times the performance of a Raspberry Pi 1. This was suggested^l to be highly dependent upon task threading and instruction set use.^l Benchmarks showed the Raspberry Pi 3 to be approximately 80% faster than the Raspberry Pi 2 in parallelised tasks.

Overclocking

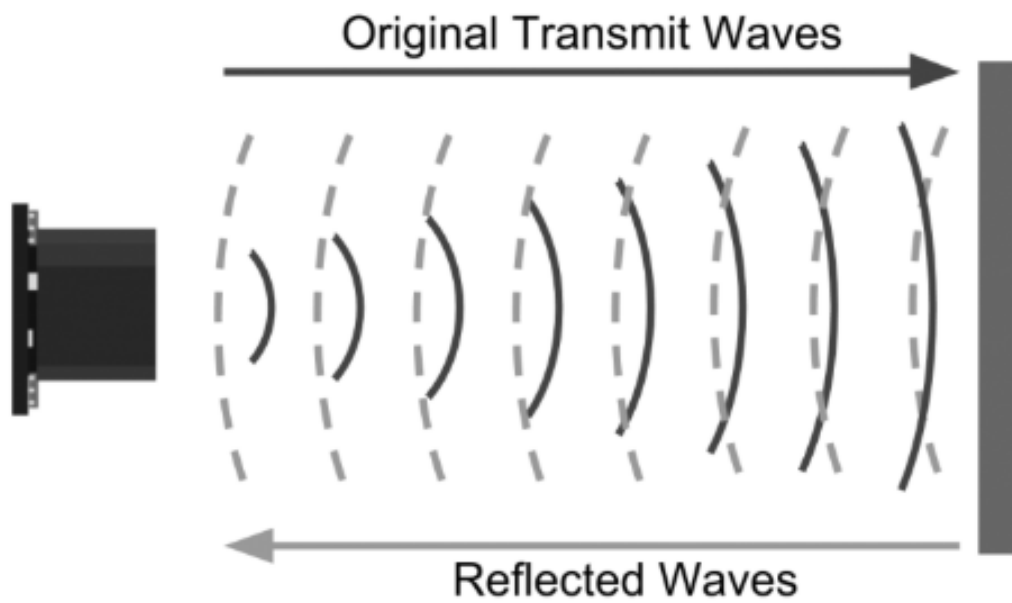
- Most Raspberry Pi systems-on-chip could be overclocked to 800 MHz, and some to 1000 MHz. There are reports the Raspberry Pi 2 can be similarly overclocked, in extreme cases, even to 1500 MHz (discarding all safety features and over-voltage limitations). In the Raspbian Linux distro the overclocking 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 maximise 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 individual board and on which of the turbo settings is used.

4.2 ULTRASONIC SENSORS:

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves. An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity. High-frequency sound waves reflect from boundaries to produce distinct echo patterns.

Ultrasonic sound vibrates at a frequency above the range of human hearing. Transducers are the microphones used to receive and send the ultrasonic sound. Our ultrasonic sensors, like many others, use a single transducer to send a pulse and to receive the echo. The sensor determines the distance to a tsonic pulse.



Ultrasound is reliable in any lighting environment and can be used inside or outside. Ultrasonic sensors can handle collision avoidance for a robot, and being moved often, as long as it isn't too fast.

ULTRASONIC SENSOR MODULE



Ultrasonics are so widely used, they can be reliably implemented in grain bin sensing applications, water level sensing, drone applications and sensing cars at your local drive-thru restaurant or bank.

Ultrasonic rangefinders are commonly used as devices to detect a collision.

Ultrasonic Sensors are best used in the non-contact detection of:

- Presence
- Level

Defining sound wave

Sound is a mechanical wave travelling through the mediums, which may be a solid, or liquid or gas. Sound waves can travel through the mediums with specific velocity depends on the medium of propagation. The sound waves which are having high frequency reflect from boundaries and produces distinctive echo patterns.

Laws of physics for sound waves

Sound waves are having specific frequencies or number of oscillations per second. Humans can detect sounds in a frequency range from about 20Hz to 20 KHz. However, the frequency range normally employed in ultrasonic detection is 100 KHz to 50MHz. The velocity of ultrasound at a particular time and temperature is constant in a medium.

$$W = C/F \text{ (or) } W = CT$$

Where W = Wave length

C = Velocity of sound in a medium

F = Frequency of wave

T=Time Period

The most common methods of ultrasonic examination utilize either longitudinal waves or shear waves. The longitudinal wave is a compression wave in which the particle motion is in the same direction of the propagation wave. The shear wave is a wave motion in which the particle motion is perpendicular to the direction of propagation. Ultrasonic detection introduces high frequency sound waves into a test object to obtain information about the object without altering or damaging it in any way. Two values are measured in ultrasonic detection.

The amount of time, taking for the sound to travel through the medium and amplitude of the received signal. Based on velocity and time thickness can be calculated.

Thickness of material = Material sound velocity X Time of Flight

Transducers for Wave Propagation and particle detection

For sending sound waves and receiving echo, ultrasonic sensors, normally called transceivers or transducers will be used. They work on a principle similar to radar that will convert electrical energy into mechanical energy in the form of sound, and vice versa.

The commonly used transducers are contact transducers, angle beam transducers, delay line transducers, immersion transducers, and dual element transducers. Contact transducers are typically used for locating voids and cracks to the outside surface of a part as well as measuring thickness. Angle beam transducers use the principle of reflection and

mode conversion to produce refracted shear or longitudinal waves in the test material.

Delay line transducers are single element longitudinal wave transducers used in conjunction with a replaceable delay line. One of the reasons for choosing delay line transducer is that near surface resolution can be improved. The delay allows the element to stop vibrating before a return signal from the reflector can be received.

The major advantages offered by immersion transducers over contact transducers are Uniform coupling reduces sensitivity variations, Reduction in scan time, and increases sensitivity to small reflectors.

Operation of ultrasonic sensors:

When an electrical pulse of high voltage is applied to the ultrasonic transducer it vibrates across a specific spectrum of frequencies and generates a burst of sound waves. Whenever any obstacle comes ahead of the ultrasonic sensor the sound waves will reflect back in the form of echo and generates an electric pulse. It calculates the time taken between sending sound waves and receiving echo. The echo patterns will be compared with the patterns of sound waves to determine detected signal's condition.

Applications involving Ultrasonic detection:

The distance of obstacle or discontinuities in metals is related to velocity of sound waves in a medium through which waves are passed and the time taken for echo reception. Hence the ultrasonic detection can be used for finding the distances between particles, for detecting the discontinuities in metals and for indicating the liquid level.

Ultrasonic Distance Measurement

Ultrasonic sensors are used for distance measuring applications. These gadgets regularly transmit a short burst of ultrasonic sound to a target, which reflects the sound back to the sensor. The system then measures the time for the echo to return to the sensor and computes the distance to the target using the speed of sound within the medium.

Different sorts of transducers are utilized within industrially accessible ultrasonic cleaning devices. An ultrasonic transducer is affixed to a stainless-steel pan which is filled with a solvent and a square wave is applied to it, conferring vibration energy on the liquid.



The ultrasonic distance sensors measure distance using sonar; an ultrasonic (well above human hearing) beat is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo return. Output from the ultrasonic sensor is a variable-width beat that compares to the distance to the target.

8 Features of Ultrasonic Distance Sensor:

1. Supply voltage: 5V (DC).
2. Supply current: 15mA.
3. Modulation frequency: 40Hz.
4. Output: 0 – 5V (Output high when obstacle detected in range).
5. Beam Angle: Max 15 degree.
6. Distance: 2cm – 400cm.
7. Accuracy: 0.3cm.
8. Communication: Positive TTL pulse.

4.3 TEMPERATURE SENSOR:

‘The world is getting warmer day by day.’ This sentence has started to crop up every now and then in our day to day lives. But seldom do we wonder what it means. What is warmer? How warm is warm? Is it hotter than hot, or just hotter than cold? To our human senses, temperature is only a subjective evaluation. For an objective and reproducible measurement, we need to quantify the temperature values, and to do that, a suitable measurement device is required

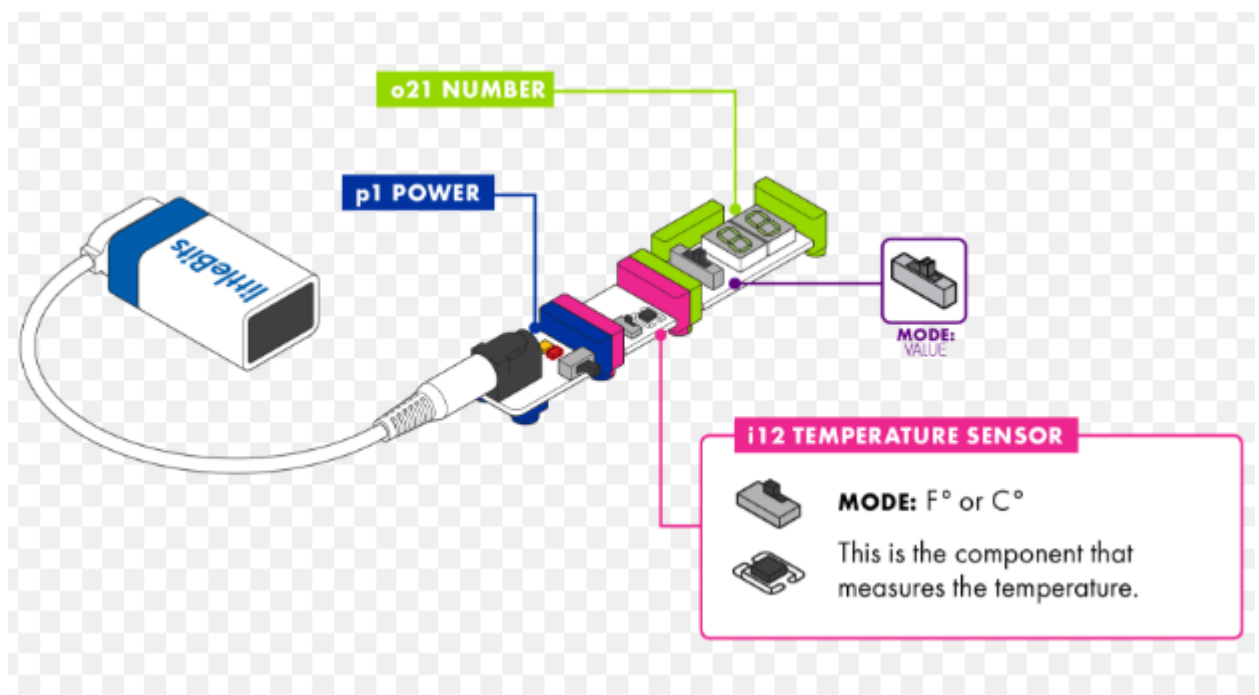
SPECIFICATIONS:

Range	0°...100°C
Accuracy	< 0,1°C + NTC-spread over 0°...70°C
Resolution	2 m°C@30°C and 25 m°C@100°C
Linearity	see table
No. Sensors	1....30
Type	5 kOhm NTC-resistors Max. wireresistance 3,3 Ohm
Conditioner	Wheatstone-bridge with +15 Bits ADC
Calibration	at 0°C and 100°C _ 0,1°C
Drift	1 m°C/°C with periodic calibration and 5 m°C/°C without calibration
Scanfreq.	default max. 10 meas./s
Amb.temp	0°...60°
Control	Micro-Wire™ 5 (Di, Do, Clk en CS)
Levels	0 and 5 Volt
Power	+12 Volt 3,5 mAmp (+5,5 +20 Volt) -5 Volt 1 mAmp (-4,5-8 Volt)
Dimensions	224 x 120 x 60 + 40 mm

Temperature and its Measurement

Simply speaking, temperature is the degree of hotness of the body which is a measure of the heat content in the body. The problem to quantify the heat content of the body on a scale did not arise until the invention of the Steam Engine. The curiosity of scientists to understand the behavior of water at different levels of heat contents gave rise to a formal and better laid out study. One of the first references for ‘temperature’ dates back to 1760, when Joseph Black declared that applying the same heat to different materials resulted in different temperatures. Years of rigorous scientific study led to many theories ranging from the simple ‘Caloric’ concept, which treated heat as a material substance which is exchanged among materials, to Carnot’s description of heat as a form of energy

(which laid the foundation of the first law of thermodynamics). However, none of them satisfactorily explained the concept of temperature. It was Maxwell's theory which offered good reasoning into it. He defined temperature of a body as is its thermal property which provides information about the energy content of the system. It is the measure of the average kinetic energy (energy by virtue of motion) of the molecules of the substance and signifies a heat potential due to which heat flows from higher temperature to lower temperature.



There are many different types of **Temperature Sensor** available and all have different characteristics depending upon their actual application. A temperature sensor consists of two basic physical types:

- **Contact Temperature Sensor Types** – These types of temperature sensor are required to be in physical contact with the object being sensed and use conduction to monitor changes in temperature. They can be used to detect solids, liquids or gases over a wide range of temperatures.
- **Non-contact Temperature Sensor Types** – These types of temperature sensor use convection and radiation to monitor changes

in temperature. They can be used to detect liquids and gases that emit radiant energy as heat rises and cold settles to the bottom in convection currents or detect the radiant energy being transmitted from an object in the form of infra-red radiation (the sun).

The two basic types of contact or even non-contact temperature sensors can also be sub-divided into the following three groups of

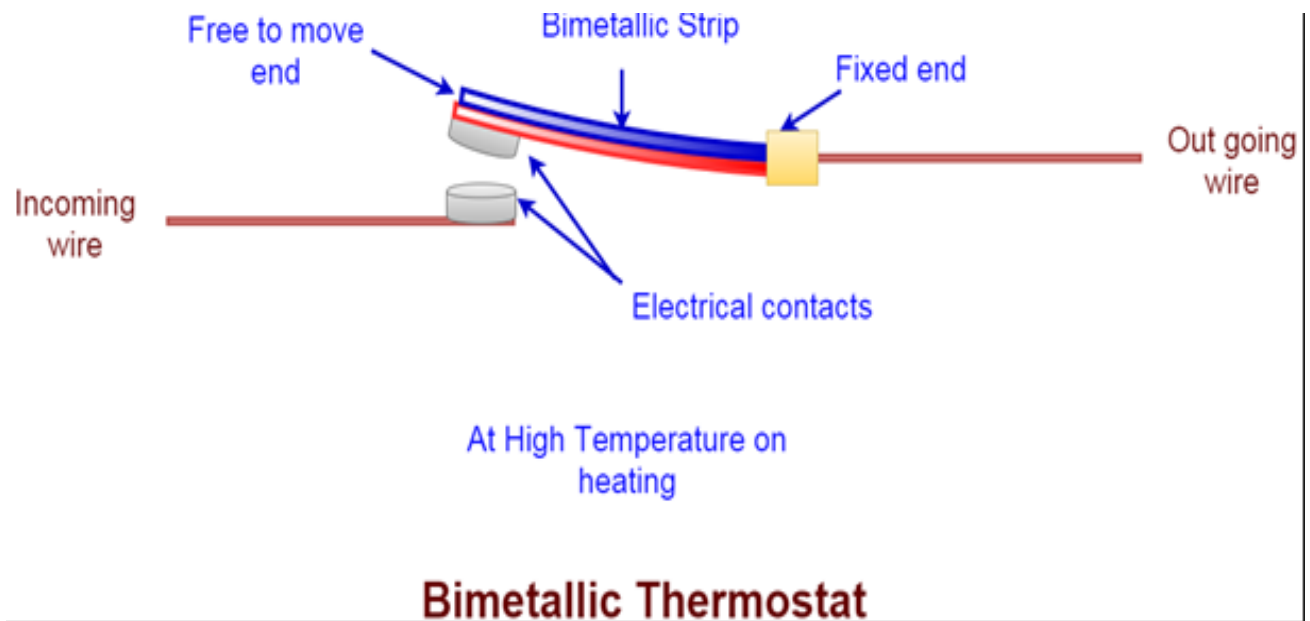
- **sensors,**
- ***Electro-mechanical,***
- ***Resistive and Electronic.***

The Thermostat

The **Thermostat** is a contact type electro-mechanical temperature sensor or switch, that basically consists of two different metals such as nickel, copper, tungsten or aluminum etc., that are bonded together to form a **Bi-metallic strip**. The different linear expansion rates of the two dissimilar metals produces a mechanical bending movement when the strip is subjected to heat.

The bi-metallic strip can be used itself as an electrical switch or as a mechanical way of operating an electrical switch in thermostatic controls and are used extensively to control hot water heating elements in boilers, furnaces, hot water storage tanks as well as in vehicle radiator cooling systems.

The Bi-metallic Thermostat



The thermostat consists of two thermally different metals stuck together back-to-back. When it is cold the contacts are closed and current passes through the thermostat. When it gets hot, one metal expands more than the other and the bonded bi-metallic strip bends up (or down) opening the contacts preventing the current from flowing.

On/Off Thermostat

There are two main types of bi-metallic strips based mainly upon their movement when subjected to temperature changes. There are the “snap-action” types that produce an instantaneous “ON/OFF” or “OFF/ON” type action on the electrical contacts at a set temperature point, and the slower “creep-action” types that gradually change their position as the temperature changes.

Snap-action type thermostats are commonly used in our homes for controlling the temperature set point of ovens, irons, immersion hot water tanks and they can also be found on walls to control the domestic heating system.

Creeper types generally consist of a bi-metallic coil or spiral that slowly unwinds or coils-up as the temperature changes. Generally, creeper type

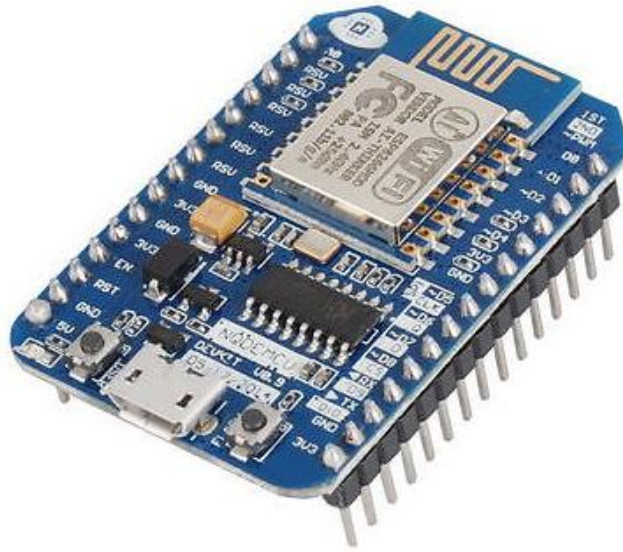
bi-metallic strips are more sensitive to temperature changes than the standard snap ON/OFF types as the strip is longer and thinner making them ideal for use in temperature gauges and dials etc.

Although very cheap and are available over a wide operating range, one main disadvantage of the standard snap-action type thermostats when used as a temperature sensor, is that they have a large hysteresis range from when the electrical contacts open until when they close again. For example, it may be set to 20°C but may not open until 22°C or close again until 18°C.

So the range of temperature swing can be quite high. Commercially available bi-metallic thermostats for home use do have temperature adjustment screws that allow for a more precise desired temperature set-point and hysteresis level to be pre-set.

4.4 Node MCU

NodeMCU is an open source IoT platform. It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module. The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the Lua scripting language. It is based on the e Lua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open-source projects, such as lua-cjso and SPIFFS.

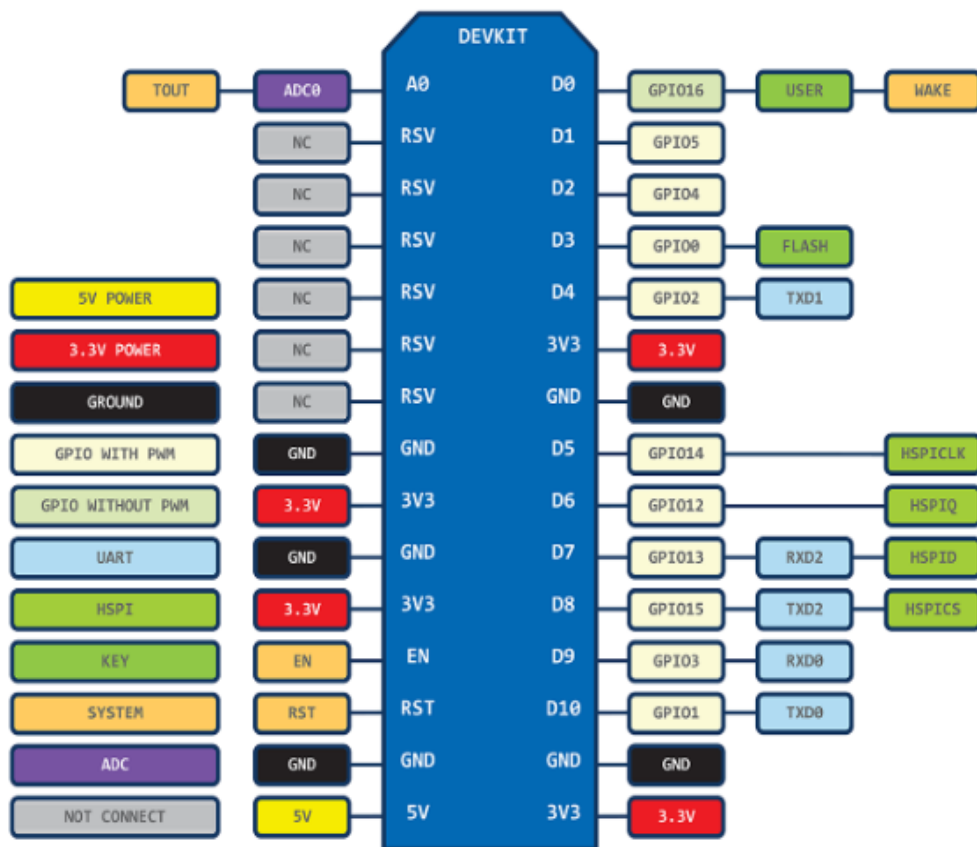


The NodeMCU (Node MicroController Unit) is an open source software and hardware development environment that is built around a very inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains all crucial elements of the modern computer: CPU, RAM, networking (wifi), and even a modern operating system and SDK. When purchased at bulk, the ESP8266 chip costs only \$2 USD a piece. That makes it an excellent choice for IoT projects of all kinds.

However, as a chip, the ESP8266 is also hard to access and use. You have to solder wires, with the appropriate analog voltage, to its PINs for the simplest tasks such as powering it on or sending a keystroke to the “computer” on the chip. And, you have to program it in low-level machine instructions that can be interpreted by the chip hardware. While this level of integration is not a problem when the ESP8266 is used as an embedded controller chip in mass-produced electronics, it is a huge burden for hobbyists, hackers, or students who want to experiment with it in their own IoT projects.

Borrowing a page from the successful playbooks of Arduino or a Raspberry Pi, the NodeMCU project aims to simplify ESP8266 development. It has two key components.

1. An open source ESP8266 firmware that is built on top of the chip manufacturer's proprietary SDK. The firmware provides a simple programming environment based on eLua (embedded Lua), which is a very simple and fast scripting language with an established developer community. For new comers, the Lua scripting language is easy to learn.
2. A DEVKIT board that incorporates the ESP8266 chip on a standard circuit board. The board has a built-in USB port that is already wired up with the chip, a hardware reset button, wifi antenna, LED lights, and standard-sized GPIO (General Purpose Input Output) pins that can plug into a bread board. Figure 1 shows the DEVKIT board, and Figure 2 shows the schema of its pins.
3. The Arduino project creates an open-source hardware design and software SDK for a versatile IoT controller. Similar to NodeMCU, the Arduino hardware is a microcontroller board with a ready USB connector, LED lights, and standard data pins. It also defines standard interfaces to interact with sensors or other boards. But unlike NodeMCU, the Arduino board can have different types of CPU chips (typically an ARM or Intel x86 chip) with memory chips, and a variety of programming environments. In fact, there is an Arduino reference design for the ESP8266 chip as well. However, the flexibility of Arduino also means significant variations across different vendors. For example, most Arduino boards do not have wifi capabilities and some even have a serial data port instead of a USB port. I feel that NodeMCU provides a more consistent and accessible experience for IoT developers.



4.5 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, and confirmation of user input such as a mouse click or keystroke.



TYPES

Electromechanical

Early devices were based on an electromechanical system identical to an electric bell without the metal gong. Similarly, a relay may be connected to interrupt its own actuating current, causing the contacts to buzz. Often these units were anchored to a wall or ceiling to use it as a sounding board. The word "buzzer" comes from the rasping noise that electromechanical buzzers made.

Mechanical

A joy buzzer is an example of a purely mechanical buzzer and they require drivers. Other examples of them are doorbells.



Piezoelectric

A piezoelectric element may be driven by an oscillating electronic circuit or other audio signal source, 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.

A piezoelectric buzzer/beeper also depends on acoustic cavity resonance or Helmholtz resonance to produce an audible beep

APPLICATIONS

While technological advancements have caused buzzers to be impractical and undesirable, there are still instances in which buzzers and similar circuits may be used. Present day applications include:

- Novelty uses
- Judging panels
- Educational purposes
- Annunciator panels
- Electronic metronomes
- Game show lock-out device
- Microwave ovens and other household appliances
- Sporting events such as basketball games

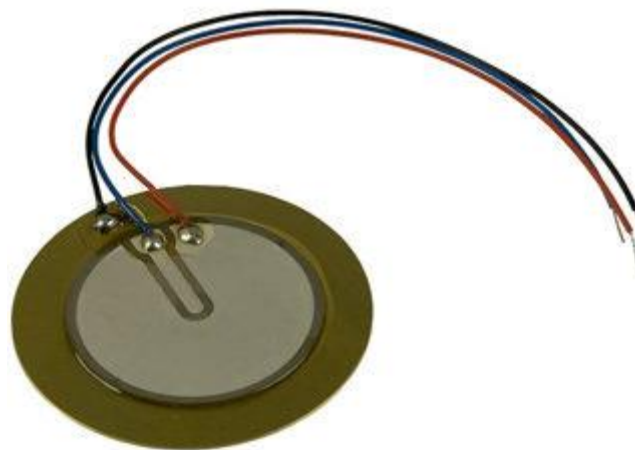
- Electrical alarms
- Joy buzzer (mechanical buzzer used for pranks)

Simple Piezo Buzzer circuit diagram

Piezo buzzers are one of the most common buzzers available around, they got their name from the piezoelectric material used as the active element.

These buzzers are usually driven at a relatively higher voltage but low current, consumes a little power, but still capable of producing very high sound.

The Piezo element must be a three terminal one, like in the picture.

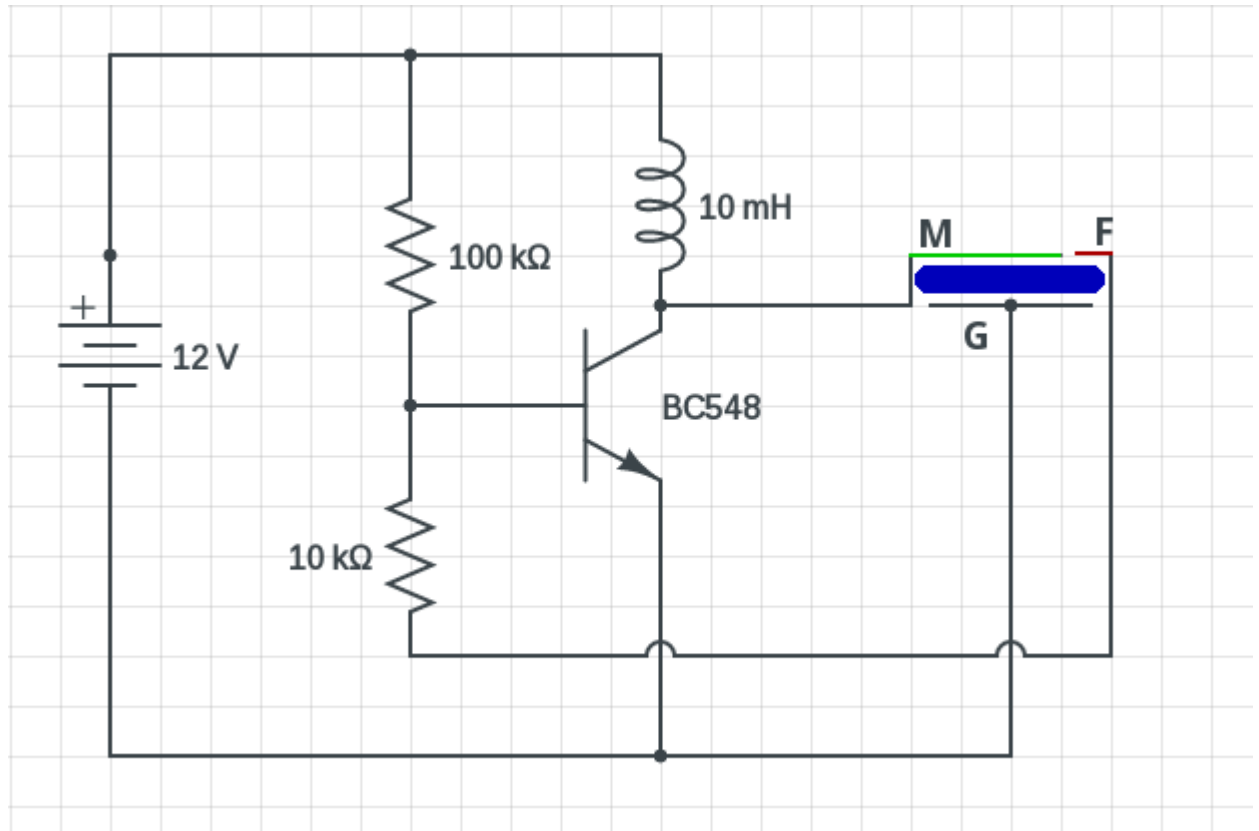


The blue wire is connected to feedback(**F**) terminal, red wire to the main (**M**) terminal and the black wire to the piezo element's ground(**G**) plate.

The inductor coil's value and shape is not crucial. You can use any coil from **1mH** to **10mH** or more, or even no measured value at all. I used a 40 turn coil on a small ferrite toroid in the final design.

Circuit diagram and construction

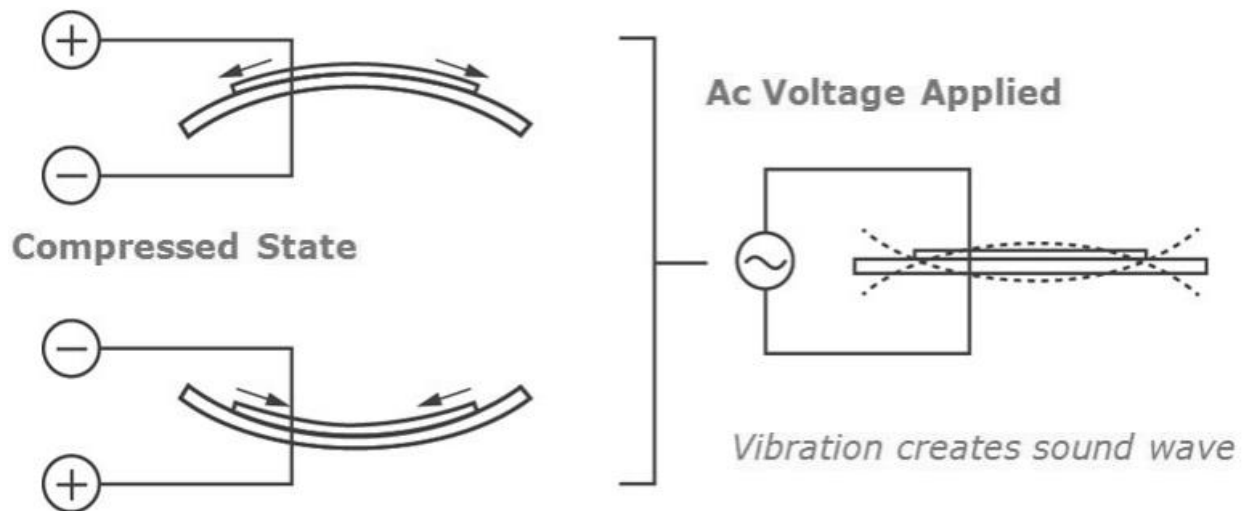
Lets have a look at the circuit diagram,



Note the piezoelectric element's pinout, **M** is the main terminal, **F** is the feedback terminal and **G** is the ground plate.

The circuit is fairly simple, you can use a little piece of strip board to make it. As this piezo buzzer circuit uses very few components, so it also could be constructed by soldering the components to each other.

When A voltage is applied to the electrodes of the piezo element, they produces flex in either way. This flex force bends the ground plate up and down.



The exact opposite thing happens too, when a piezoelectric element is subjected to varying pressure, it produces voltage.

As you've seen before, self drive piezo buzzers are constructed with an extra electrically isolated feedback electrode. The voltage created by the flex force is available in the feedback terminal.

The piezo buzzer is placed in a resonant cavity, there is a hole in the opposite side of the resonant cavity from where the buzzing sound comes out.

The driver circuit and piezo buzzer co-operates soon between themselves and they start oscillating on the resonant frequency of the piezo buzzer.

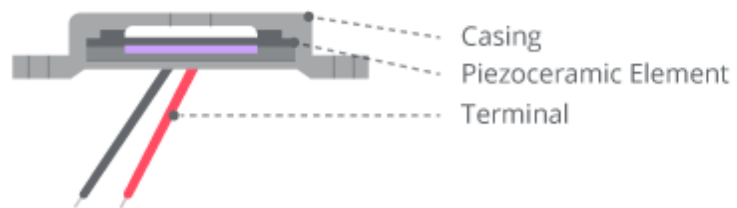
Piezo Buzzer Characteristics

- Wide operating voltage: 3~250 V
- Lower current consumption: less than 30 mA higher rated frequency
- Larger footprint

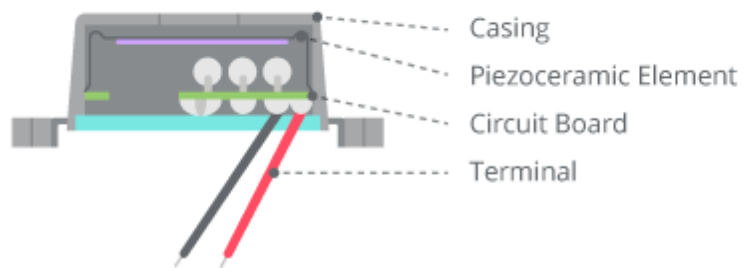
- Higher sound pressure level

STRUCTURE OF PIEZO BUZZER

Transducer *(without driving circuitry)*



Indicator *(with driving circuitry)*



There are two types of piezo buzzers - transducers and indicators. Transducers consist of a casing, a piezoceramic element and a terminal. In order to operate a transducer, the user must send a square wave signal to the buzzer. Indicators consist of a casing, a piezoceramic element, a circuit board and a terminal. In order to operate an indicator, the user must send the buzzer a specified dc voltage.

4.6 POWER SUPPLY UNIT:

A power supply (sometimes known as a power supply unit or PSU) is a device or system that supplies electrical or other types of energy to an output load or group of loads. The term is most commonly applied to electrical energy supplies, less often to mechanical ones, and rarely to others.

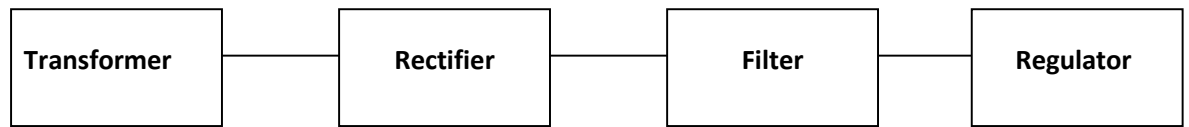


Fig4.1 Block diagram of power supply

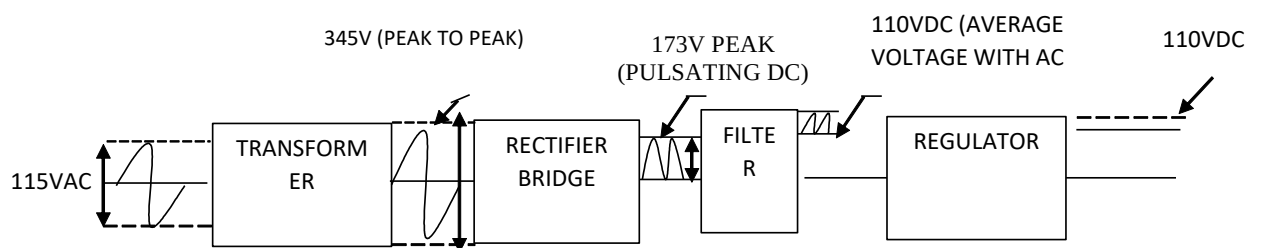


Fig 4.2 Processing of Power supply

The transformer steps up or steps down the input line voltage and isolates the power supply from the power line. The RECTIFIER section converts the alternating current input signal to a pulsating direct current. However, as you proceed in this chapter you will learn that pulsating dc is not desirable. For this reason, a FILTER section is used to convert pulsating dc to a purer, more desirable form of dc voltage.

The final section, the REGULATOR, does just what the name implies. It maintains the output of the power supply at a constant level in spite of large changes in load current or input line voltages. Now that you know what each section does, let's trace an ac signal through the power supply. At this point you need to see how this signal is altered within each section of the power supply. Later on in the chapter you will see how these changes take place. An input signal of 115 volts ac is applied to the primary of the transformer.

The transformer is a step-up transformer with a turn's ratio of 1:3. You can calculate the output for this transformer by multiplying the input voltage by the ratio of turns in the primary to the ratio of turns in the secondary; therefore, $115 \text{ volts ac} \times 3 = 345 \text{ volts ac}$ (peak-to-peak) at the output. Because each diode in the rectifier section conducts for 180 degrees of the 360-degree input, the output of the rectifier will be one-half, or approximately 173 volts of pulsating dc. The filter section, a network of resistors, capacitors, or inductors, controls the rise and fall time of the varying signal; consequently, the signal remains at a more constant dc level. You will see the filter process more clearly in the discussion of the actual filter circuits. The output of the filter is a signal of 110 volts dc, with ac ripple riding on the dc. The reason for the lower voltage (average voltage) will be explained later in this chapter. The regulator maintains its output at a constant 110-volt dc level, which is used by the electronic equipment (more commonly called the load).

Simple 5V power supply for digital circuits

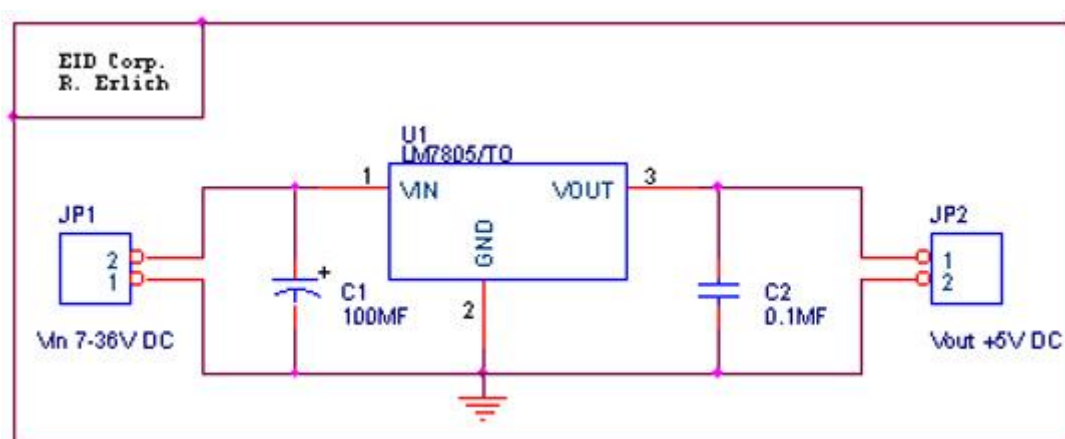
- Brief description of operation: Gives out well-regulated +5V output, output current capability of 100 mA
- Circuit protection: Built-in overheating protection shuts down output when regulator IC gets too hot
- Circuit complexity: Very simple and easy to build
- Circuit performance: Very stable +5V output voltage, reliable operation
- Availability of components: Easy to get, uses only very common basic components
- Design testing: Based on datasheet example circuit, I have used this circuit successfully as part of many electronics projects

- Applications: Part of electronics devices, small laboratory power supply
- Power supply voltage: Unregulated DC 8-18V power supply
- Power supply current: Needed output current + 5 mA
- Component costs: Few dollars for the electronics components + the input transformer cost

Circuit description

This circuit is a small +5V power supply, which is useful when experimenting with digital electronics. Those transformers are easily available, but usually their voltage regulation is very poor, which makes them not very usable for digital circuit experimenter unless a better regulation can be achieved in some way.

This circuit can give +5V output at about 150 mA current, but it can be increased to 1 A when good cooling is added to 7805 regulator chip. The circuit has over overload and thermal protection.



Circuit diagram

Circuit diagram of the power supply:

The capacitors must have enough high voltage rating to safely handle the input voltage feed to circuit. The circuit is very easy to build into a piece of Vero board.

Pinout of the 7805 regulator IC.

1. Unregulated voltage in
2. Ground
3. Regulated voltage out

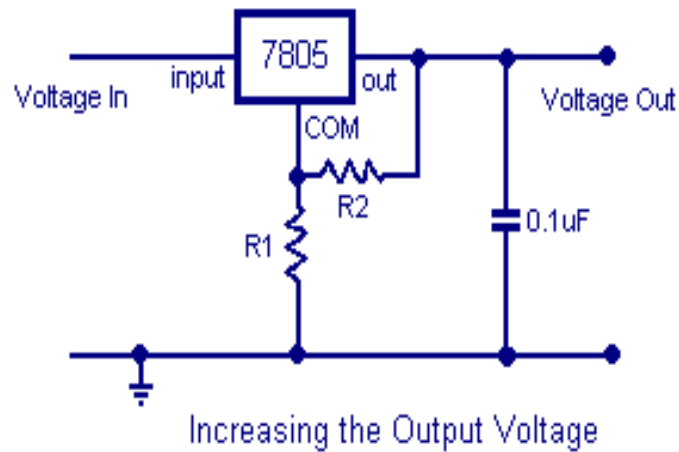
COMPONENT LIST

- 7805 regulator IC.
- 100 uF electrolytic capacitor, at least 25V voltage rating.
- 10 uF electrolytic capacitor, at least 6V voltage rating.
- 100 nF ceramic or polyester capacitor.

MORE OUTPUT CURRENT

If we need more than 150 mA of output current, we can update the output current up to 1A doing the following modifications:

- Change the transformer from where we take the power to the circuit to a model which can give as much current as we need from output
- Put a heat sink to the 7805 regulators (so big that it does not overheat because of the extra losses in the regulator)



MORE OUTPUT CURRENT

OTHER OUTPUT VOLTAGES

If we need other voltages than +5V, we can modify the circuit by replacing the 7805 chips with another regulator with different output voltage from regulator 78xx chip family. The last numbers in the chip code tells the output voltage. The input voltage must be at least 3V greater than regulator output voltage to otherwise the regulator does not work well.

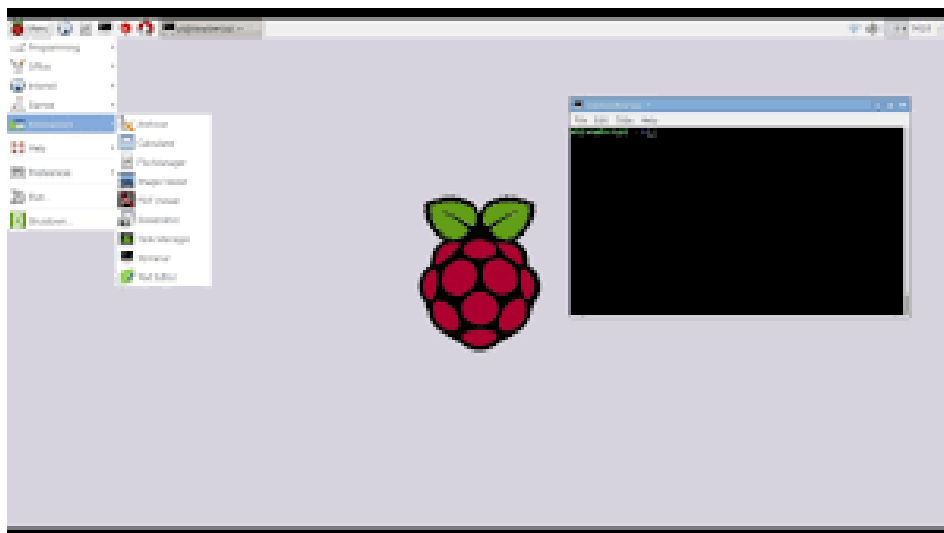
CHAPTER-5

SOFTWARE DESCRIPTIONS

5.1 RASPBIAN OS

Raspberry Pi OS (formerly **Raspbian**) is a Debian-based operating system for Raspberry Pi. Since 2015, it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the Raspberry Pi family of compact single-board computers. The first version of Raspbian was created by Mike Thompson and Peter Green as an independent project.^[5] The initial build was completed in June 2012.

Previous, Raspberry Pi OS versions were 32-bit only and based on Debian, taking the name Raspbian. Since the more recent 64-bit versions no longer use Debian, the name was changed to Raspberry Pi OS for both the 64-bit and 32-bit versions. As of 1 February 2021, the 64-bit version is in beta and is not suitable for general use.



Raspberry Pi OS is highly optimized for the Raspberry Pi line of compact single-board computers with ARM CPUs. It runs on every Raspberry Pi except the Pico microcontroller. Raspberry Pi OS uses a modified LXDE as its desktop environment with the OpenBox stacking

window manager, along with a unique theme. The distribution is shipped with a copy of the algebra program Wolfram Mathematica^[4] and a version of Minecraft called Minecraft: Pi Edition, as well as a lightweight version of the Chromium web browser.

FEATURES:

User Interface

Raspberry Pi OS looks similar to many common desktops, such as macOS and Microsoft Windows. The menu bar is positioned at the top and contains an application menu and shortcuts to Terminal, Chromium, and File Manager. On the right is a Bluetooth menu, a Wi-Fi menu, volume control, and a digital clock

Package Management

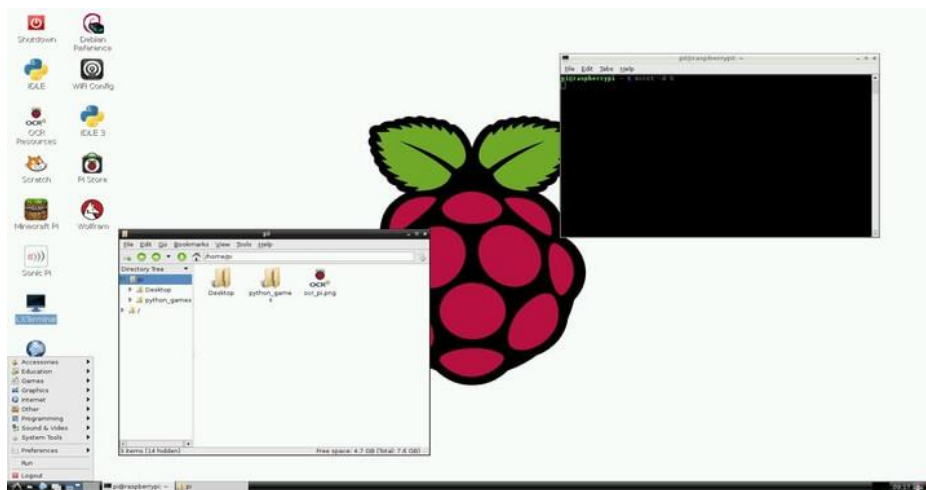
Packages can be installed via APT, the Recommended Software app, and by using the Add/Remove Software tool, a GUI wrapper for APT

Microsoft Repository Controversy

In late January 2021, Raspberry Pi OS' raspberrypi-sys-mods package added a trusted GPG key and sources.list.d entry to APT without user consent. This addition granted Microsoft the ability to install and run any software during the daily critical update process on all Pi that had done a manual apt upgrade to receive the change. The change was not pushed as a critical update and, as of yet, the excessive permission has not been abused by Microsoft and would seem unlikely to ever be abused. The author of the change acknowledged on GitHub that too many rights were granted to Microsoft and also acknowledged delaying the public release of the source code for the change.

In addition to the permissions, the change also causes Pi running an updated Raspberry Pi OS to contact packages.microsoft.com daily and

thereby reveal their IP address as a Raspberry Pi OS user for potential use in tracking or marketing efforts. On 8 February 2021, the original author made another change that restricted Microsoft's ability to install software to packages beginning with the string "code" but Microsoft can still run code as root so this restriction is trivial to bypass. As of 8 February 2021, the issue is not resolved and the Raspberry Pi Foundation has locked or deleted many of the related threads on their public forum and their GitHub pages but has acknowledged there is a problem to be resolved and that they are working on it.



Components

PCManFM is a file browser allowing quick access to all areas of the computer, and was redesigned in the first Raspberry Pi OS Buster release (2019-06-20).

Raspberry Pi OS originally used Epiphany as the web browser, but switched to Chromium with the launch of its redesigned desktop.^[13]

Raspberry Pi OS comes with many beginner IDEs, such as Thonny Python IDE, Mu Editor, and Greenfoot. It also ships with educational software like Scratch and Bookshelf.

5.2 PYTHON

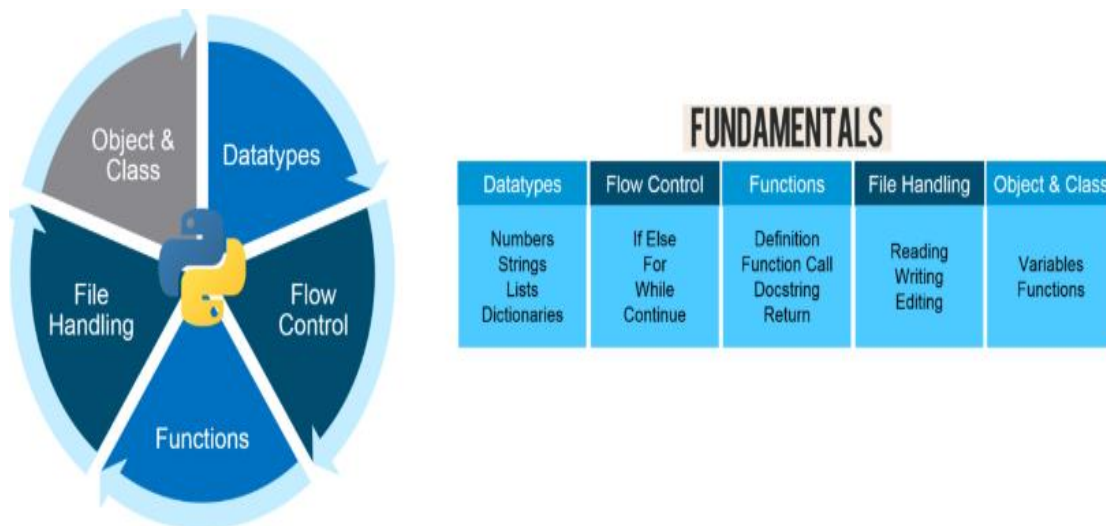
Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter()`, `map()`, and `reduce()` functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard M.



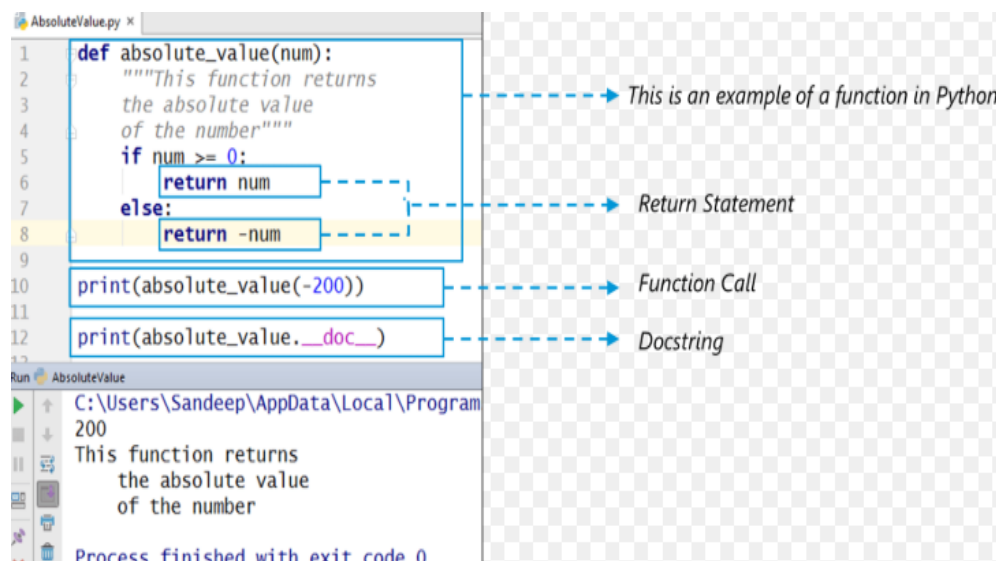
The language's core philosophy is summarized in the document *The Zen of Python* (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is *not* considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach

to language design in favor of "there should be one—and preferably only one—obvious way to do it".



Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.^[51] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or

shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonists*, *Pythonistas*, and *Pythoneers*

Syntax and semantics

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. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Main article: Python syntax and semantics § Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure.^[1] This feature is also sometimes termed the off-side rule.

Statements and control flow

Python's statements include (among others):

- The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and

this fundamental mechanism (including the nature of Python's version of *variables*) illuminates many other features of the language. Assignment in C, e.g., `x = 2`, translates to "typed variable name `x` receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, `x = 2`, translates to "(generic) name `x` receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed *binding* the name to the object. Since the name's storage location doesn't *contain* the indicated value, it is improper to call it a *variable*. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., `x = 2`; `y = 2`; `z = 2` result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to *some* object, which **will** have a type; thus there is dynamic typing.

- The `if` statement, which conditionally executes a block of code, along with `else` and `elif` (a contraction of else-if).
- The `for` statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The `while` statement, which executes a block of code as long as its condition is true.

- The `try` statement, which allows exceptions raised in its attached code block to be caught and handled by `except` clauses; it also ensures that clean-up code in a `finally` block will always be run regardless of how the block exits.
- The `raise` statement, used to raise a specified exception or re-raise a caught exception.
- The `class` statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The `def` statement, which defines a function or method.
- The `with` statement, from Python 2.5 released on September 2006, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.
- The `pass` statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The `assert` statement, used during debugging to check for conditions that ought to apply.
- The `yield` statement, which returns a value from a generator function. From Python 2.5, `yield` is also an operator. This form is used to implement coroutines.
- The `import` statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using `import`: `import <module name> [as <alias>]` or `from <module name> import *` or `from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ...`

- The `print` statement was changed to the `print()` function in Python 3.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

Expressions

Some Python expressions are similar to languages such as C and Java, while some are not:

- Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division and integer division. Python also added the `**` operator for exponentiation.
- From Python 3.5, the new `@` infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication
- In Python, `==` compares by value, versus Java, which compares numerics by value^[69] and objects by reference.^[70] (Value comparisons in Java on objects can be performed with the `equals()` method.) Python's `is` operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example `a <= b <= c`.

- Python uses the words `and`, `or`, `not` for its boolean operators rather than the symbolic `&&`, `||`, `!` used in Java and C.
- Python has a type of expression termed a *list comprehension*. Python 2.4 extended list comprehensions into a more general expression termed a *generator expression*.
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
- Conditional expressions in Python are written as `x if c else y`^[71] (different in order of operands from the `c ? x : y` operator common to many other languages).
- Python makes a distinction between lists and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The `+` operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable `t` initially equal to `(1, 2, 3)`, executing `t = t + (4, 5)` first evaluates `t + (4, 5)`, which yields `(1, 2, 3, 4, 5)`, which is then assigned back to `t`, thereby effectively "modifying the contents" of `t`, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.
- Python features *sequence unpacking* where multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left-hand side of

the equal sign in an assignment statement. The statement expects an *iterable* object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.

- Python has a "string format" operator `%`. This functions analogous to `printf` format strings in C, e.g. `"spam=%s eggs=%d" % ("blah", 2)` evaluates to `"spam=blah eggs=2"`. In Python 3 and 2.6+, this was supplemented by the `format()` method of the `str` class, e.g. `"spam={0} eggs={1}".format("blah", 2)`. Python 3.6 added "f-strings": `blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'`.
- Python has various kinds of string literals:
 - Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (`\`) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".
 - Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
 - Raw string varieties, denoted by prefixing the string literal with an `r`. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "`@-quoting`" in C#.

- Python has array index and array slicing expressions on lists, denoted as `a[key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the *start* index up to, but not including, the *stop* index. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

- List comprehensions vs. `for`-loops
- Conditional expressions vs. `if` blocks
- The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements.

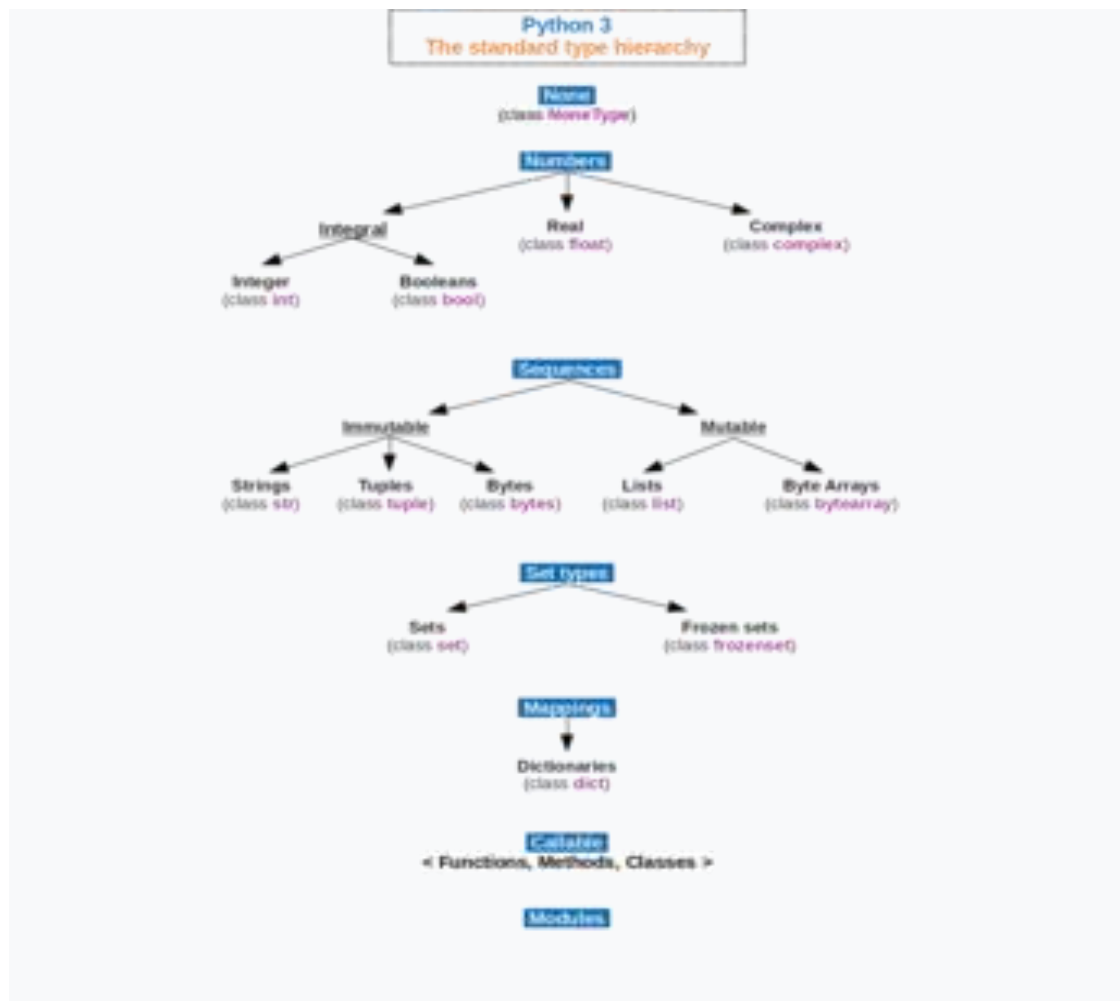
Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is syntactically valid (but probably unintended) C code but `if c = 1: ...` causes a syntax error in Python.

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.



EXAMPLE OF PYTHON PROGRAMMING

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass `type` (itself an instance of itself), allowing metaprogramming and reflection.

```

menu=["Bread","Doughnut","Muffine","COOKIE"]
prices=[5,2,3,1]

def setup():
    size(500,500)

def draw():
    line(250,0,250,500)

    text("Menu",100,50)

    text("Bread: $5", 30, 100)
    text("Doughnut: $2", 30, 120)
    text("Muffin: $2", 30, 140)
    text("COOKIE: $2", 30, 160)

    text("Prices",350,50)

```

Before version 3.0, Python had two kinds of classes: *old-style* and *new-style*. The syntax of both styles is the same, the difference being whether the class `object` is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named *mypy* supports compile-time type checking.

PYTHON IS USED BECAUSE:

1)Readable and Maintainable Code

While writing a software application, you must focus on the quality of its source code to simplify maintenance and updates. The syntax rules of Python allow you to express concepts without writing additional code. At the same time, Python, unlike other programming languages, emphasizes on code readability, and allows you to use English keywords instead of punctuations. Hence, you can use Python to build custom applications without writing additional code. The readable and clean code base will help you to maintain and update the software without putting extra time and effort.

2) Multiple Programming Paradigms

Like other modern programming languages, Python also supports several programming paradigm. It supports object oriented and structured programming fully. Also, its language features support various concepts in functional and aspect-oriented programming. At the same time, Python also features a dynamic type system and automatic memory management. The programming paradigms and language features help you to use Python for developing large and complex software applications.

3) Compatible with Major Platforms and Systems

At present, Python is supporting many operating systems. You can even use Python interpreters to run the code on specific platforms and tools. Also, Python is an interpreted programming language. It allows you to you to run the same code on multiple platforms without recompilation. Hence, you are not required to recompile the code after making any

alteration. You can run the modified application code without recompiling and check the impact of changes made to the code immediately. The feature makes it easier for you to make changes to the code without increasing development time.

4) Robust Standard Library

Its large and robust standard library makes Python score over other programming languages. The standard library allows you to choose from a wide range of modules according to your precise needs. Each module further enables you to add functionality to the Python application without writing additional code. For instance, while writing a web application in Python, you can use specific modules to implement web services, perform string operations, manage operating system interface or work with internet protocols. You can even gather information about various modules by browsing through the Python Standard Library documentation.

5) Many Open Source Frameworks and Tools

As an open source programming language, Python helps you to curtail software development cost significantly. You can even use several open source Python frameworks, libraries and development tools to curtail development time without increasing development cost. You even have option to choose from a wide range of open source Python frameworks and development tools according to your precise needs. For instance, you can simplify and speedup web application development by using robust Python web frameworks like Django, Flask, Pyramid, Bottle and Cherrypy. Likewise, you can accelerate desktop GUI application

development using **Python GUI frameworks** and toolkits like PyQt, PyJs, PyGUI, Kivy, PyGTK and WxPython.

6) Simplify Complex Software Development

Python is a general-purpose programming language. Hence, you can use the programming language for developing both desktop and web applications. Also, you can use Python for developing complex scientific and numeric applications. Python is designed with features to facilitate data analysis and visualization. You can take advantage of the data analysis features of Python to create custom big data solutions without putting extra time and effort. At the same time, the data visualization libraries and APIs provided by Python help you to visualize and present data in a more appealing and effective way. Many **Python developers** even use Python to accomplish artificial intelligence (AI) and natural language processing tasks.

7) Adopt Test Driven Development

You can use Python to create prototype of the software application rapidly. Also, you can build the software application directly from the prototype simply by refactoring the Python code. Python even makes it easier for you to perform coding and testing simultaneously by adopting test driven development (TDD) approach. You can easily write the required tests before writing code and use the tests to assess the application code continuously. The tests can also be used for checking if the application meets predefined requirements based on its source code.

However, Python, like other programming languages, has its own shortcomings. It lacks some of the built-in features provided by other modern programming language. Hence, you have to use Python libraries, modules, and frameworks to accelerate custom software development. Also, several studies have shown that Python is slower than several widely used programming languages including Java and C++. You have to speed up the Python application by making changes to the application code or using custom runtime. But you can always use Python to speed up software development and simplify software maintenance.

Advantages or Benefits of Python

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are-

- **Extensive Support Libraries**

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

- **Integration Feature**

Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or

Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.

- **Improved Programmer's Productivity**

The language has extensive support libraries and clean object-oriented designs that increase two to tenfold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

- **Productivity**

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

Limitations or Disadvantages of Python

Python has varied advantageous features, and programmers prefer this language to other programming languages because it is easy to learn and code too. However, this language has still not made its place in some computing arenas that includes Enterprise Development Shops. Therefore, this language may not solve some of the enterprise solutions, and limitations include-

- **Difficulty in Using Other Languages**

The Python lovers become so accustomed to its features and its extensive libraries, so they face problem in learning or working on other programming languages. Python experts may see the declaring of cast

“values” or variable “types”, syntactic requirements of adding curly braces or semi colons as an onerous task.

- **Weak in Mobile Computing**

Python has made its presence on many desktop and server platforms, but it is seen as a weak language for mobile computing. This is the reason very few mobile applications are built in it like Carbonnelle.

- **Gets Slow in Speed**

Python executes with the help of an interpreter instead of the compiler, which causes it to slow down because compilation and execution help it to work normally. On the other hand, it can be seen that it is fast for many web applications too.

- **Run-time Errors**

The Python language is dynamically typed so it has many design restrictions that are reported by some Python developers. It is even seen that it requires more testing time, and the errors show up when the applications are finally run.

- **Underdeveloped Database Access Layers**

As compared to the popular technologies like JDBC and ODBC, the Python’s database access layer is found to be bit underdeveloped and primitive. However, it cannot be applied in the enterprises that need smooth interaction of complex legacy data.

Python Features

Python's features include –

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows a student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintaining.
- **A broad standard library** – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features. A few are listed below –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Languages Influenced by Python

Python's design and philosophy have influenced many other programming languages:

- Boo uses indentation, a similar syntax, and a similar object model.^[153]
- Cobra uses indentation and a similar syntax, and its "Acknowledgements" document lists Python first among languages that influenced it. However, Cobra directly supports design-by-contract, unit tests, and optional static typing.
- CoffeeScript, a programming language that cross-compiles to JavaScript, has Python-inspired syntax.
- ECMAScript borrowed iterators and generators from Python.^[156]

- Go is designed for the "speed of working in a dynamic language like Python" and shares the same syntax for slicing arrays.
- Groovy was motivated by the desire to bring the Python design philosophy to Java.
- Julia was designed "with true macros [.. and to be] as usable for general programming as Python [and] should be as fast as C". Calling to or from Julia is possible; to with PyCall.jl and a Python package pyjulia allows calling, in the other direction, from Python.
- Kotlin is a functional programming language with an interactive shell similar to Python. However, Kotlin is strongly typed with access to standard Java libraries.
- Ruby's creator, Yukihiro Matsumoto, has said: "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language".
- Swift, a programming language developed by Apple, has some Python-inspired syntax.
- GDScript, dynamically typed programming language used to create video-games. It is extremely similar to Python with a few minor differences.

Python's development practices have also been emulated by other languages. For example, the practice of requiring a document describing the rationale for, and issues surrounding, a change to the language (in Python, a PEP) is also used in Tcl and Erlang.

Python received TIOBE's Programming Language of the Year awards in 2007, 2010 and 2018. The award is given to the language with the greatest growth in popularity over the year, as measured by the TIOBE index

There are two attributes that make development time in Python faster than in other programming languages:

1. Python is an interpreted language, which precludes the need to compile code before executing a program because Python does the compilation in the background. Because Python is a high-level programming language, it abstracts many sophisticated details from the programming code. Python focuses so much on this abstraction that its code can be understood by most novice programmers.
2. Python code tends to be shorter than comparable codes. Although Python offers fast development times, it lags slightly in terms of execution time. Compared to fully compiling languages like C and C++, Python programs execute slower. Of course, with the processing speeds of computers these days, the speed differences are usually only observed in benchmarking tests, not in real-world operations. In most cases, Python is already included in Linux distributions and Mac OS X machines.

APPLICATIONS:

Web Applications

You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Some of the popular platforms for creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS.

Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

Scientific and Numeric Computing

There are numerous libraries available in Python for scientific and numeric computing. There are libraries like: SciPy and NumPy that are used in general purpose computing. And, there are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on.

Also, the language is heavily used in machine learning, data mining and deep learning.

Creating software Prototypes

Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited and efficiency is a must.

However, Python is a great language for creating prototypes. For example: You can use Pygame (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.

Good Language to Teach Programming

Python is used by many companies to teach programming to kids and newbies.

It is a good language with a lot of features and capabilities. Yet, it's one of the easiest language to learn because of its simple easy-to-use syntax.

FEATURES OF PYTHON:

- 1. A simple language which is easier to learn**

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

2. Free and open-source

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute softwares written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

3. Portability

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

4. Extensible and Embeddable

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

5. A high-level, interpreted language

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

6. Large standard libraries to solve common tasks

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server? You can use MySQLdb library using `import MySQLdb`.

Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it won't break your application.

7. Object-oriented

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

With OOP, you are able to divide these complex problems into smaller sets by creating objects.

5.3 PHP

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994. PHP is a recursive acronym for "PHP: Hypertext Preprocessor". PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites. It is integrated with a number of popular databases,

including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time. PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time. PHP is forgiving: PHP language tries to be as forgiving as possible. PHP Syntax is C-Like.

1. PHP is an acronym for "PHP: Hypertext Preprocessor"
2. PHP is a widely-used, open source scripting language
3. PHP scripts are executed on the server
4. PHP is free to download and use
5. PHP files can contain text, HTML, CSS, JavaScript, and PHP code
6. PHP code is executed on the server, and the result is returned to the browser as plain HTML
7. PHP files have extension ".php"

CHARACTERISTICS OF PHP

Five important characteristics make PHP's practical nature possible –

Simplicity

Efficiency

Security

Flexibility

Familiarity

STRUCTURE

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php  
echo "My first PHP script!";  
?>
```

```
</body>  
</html>
```

CHAPTER-6

CONCLUSION

We conclude that the hybrid configuration outperforms configurations in which only one of the two components is used for tracking and pursuing non-compliant pedestrians. Therefore, all individuals in an indoor environment are encouraged to maintain a 6-foot distance from each other. The current approach for issuing a warning to violating pedestrians using a monitor has limitations, and we need to develop better human-robot approaches. As more such monitoring robots are used to check for social distances or collecting related data, this could also affect the behavior of pedestrians in different settings. we need to perform more studies on the social impact of such robots. Due to COVID restrictions, have only been able to evaluate the performance of COVID-robot in low to medium density laboratory settings. Eventually, we want to evaluate the robot's performance in crowded public settings and outdoor scenarios. we will also need to design better techniques to improve the enforcement of social distancing by using better human-robot interaction methods.

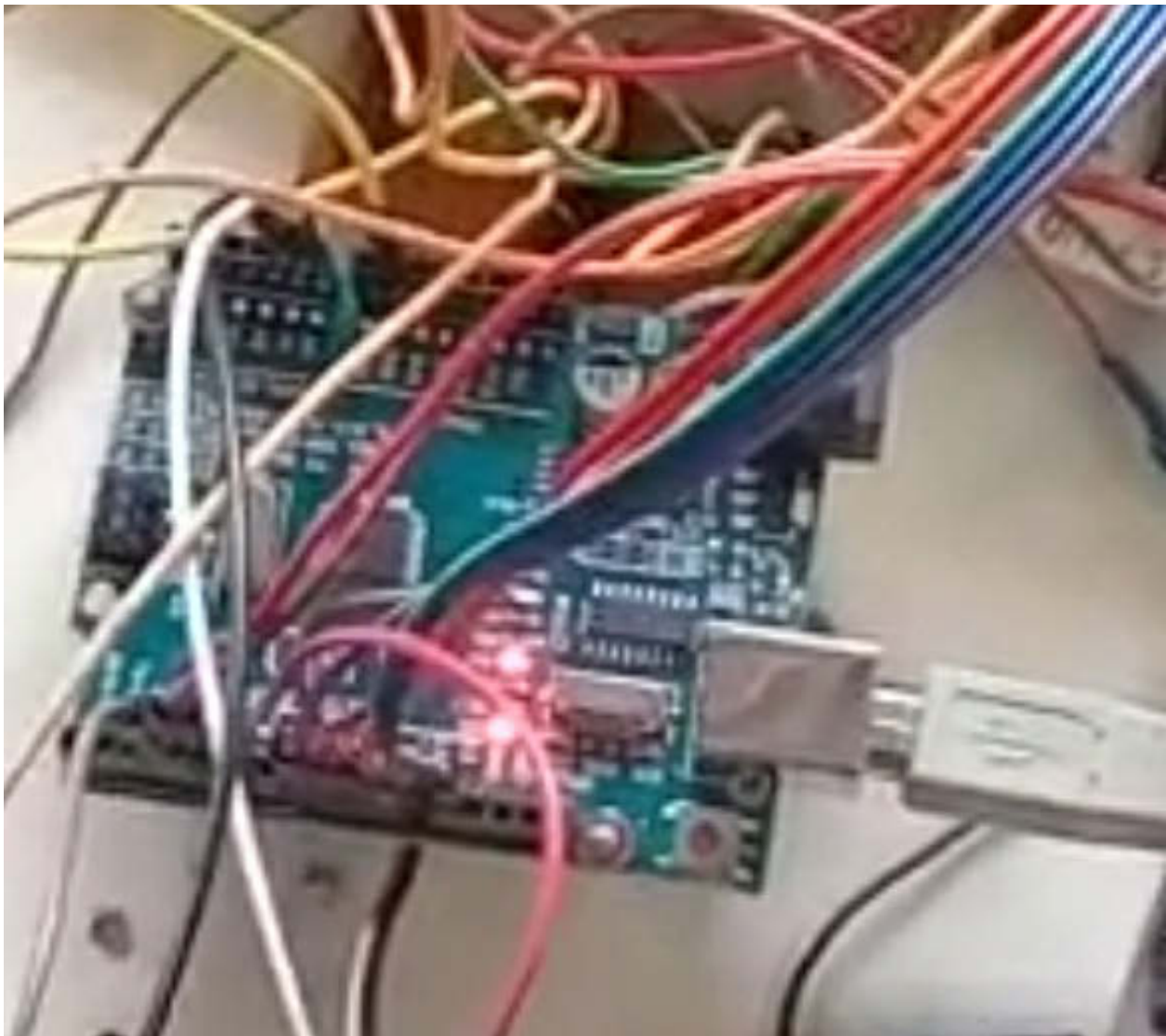
CHAPTER-7

RESULTS

The circuit is connected as per the circuit diagram shown in fig. Then we need to upload the code in Rpi microcontroller. After uploading the source code in Raspberry pi microcontroller we can see the output.

7.1. CIRCUIT CONNECTION:

The circuit connection is shown in the below figures.





Circuit Connection - View 2

- This is a whole experimental setup for the project IoT Social Distancing Robot With Face Mask Detection system by using various sensors, Rpi microcontroller along with the robotic vehicle body.

7.2. PROGRAM

```
1  # -*- coding: utf-8 -*-
2  import keras
3  import cv2
4  import numpy as np
5  #import dlib
6  from datetime import datetime
7  import os
8  import math
9  from objectDetection.detectionModel import
10
11
12  #from attention_blocks import cbam_block
13  now_ = datetime.utcnow().strftime('%Y%m%d')
14  record_dir = "./log_{}".format(now_)
15  os.makedirs(record_dir)
16
17
18  def draw_label(image, point, label, font:
19                  font_scale=0.8, thickness:
20                  size = cv2.getTextSize(label, font,
21                  x, y = point
22                  cv2.rectangle(image, (x, y - size[1]
23                  cv2.putText(image, label, point, fon
24
25
26
27
28  def image_normalization(image):
29      min_v = image.min()
30      max_v = image.max()
31
32      normalized=(image-min_v)/(max_v-min_v)
33
34      return normalized
35
36
37  #def main():
38  maskModel = keras.models.load_model(r"C:"
```

Program in the software(cont)

```

36
37 #def main():
38 maskModel = keras.models.load_model(r"C:\Users\SriAnnai\Desktop\maskModel.h5")
39 #trained_model = keras.models.load_model(r"C:\Users\SriAnnai\Desktop\trained_model.h5")
40 #detector = dlib.get_frontal_face_detector()
41
42 objectAPI = ObjectDetection(r"C:\Users\SriAnnai\Desktop\objectAPI.h5")
43
44 prototxtPath = r"C:\Users\SriAnnai\Desktop\prototxtPath.txt"
45 weightsPath = r"C:\Users\SriAnnai\Desktop\weightsPath.txt"
46 faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
47
48 cap = cv2.VideoCapture(r"C:\Users\SriAnnai\Desktop\video.mp4")
49 #cap = cv2.VideoCapture(0)
50
51 #cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
52 #cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
53 while True:
54     ret, image=cap.read()
55
56     image=cv2.resize(image,(640,360))
57     input_img = cv2.cvtColor(image, cv2.COLOR_BGR2_RGB)
58     # img_size = np.asarray(input_img.shape).astype('float32')
59     # bounding_boxes = detector(input_img, img_size)
60
61     boxes, scores, classes, num = objectAPI.detect(image)
62     humBoxes,humScore = humanBoxesAdScore(image, boxes, scores, classes, num)
63
64     (h, w) = input_img.shape[:2]
65
66     blob = cv2.dnn.blobFromImage(input_img, 1.0, (104.0, 177.0, 123.0))
67     faceNet.setInput(blob)
68     detections = faceNet.forward()
69
70
71     a = []
72     b = []
73
74     # if 1 == (humBoxes[0][0] > 0):

```

Program in the software(cont- 2)

```

115
116 # pass the blob through the network and
117 # print(detections.shape)
118
119     for i in range(0, detections.shape[2]:
120 # extract the confidence (i.e., probability)
121 # the detection
122         confidence = detections[0, 0, i, 3]
123 # filter out weak detections by ensuring
124 # greater than the minimum confidence
125         if confidence > 0.5:
126 # compute the (x, y)-coordinates of the
127 # the object
128             box = detections[0, 0, i, 3:]
129             (startX, startY, endX, endY)
130
131 # ensure the bounding boxes fall within
132 # the frame
133             (startX, startY) = (max(0, startX), max(0, startY))
134             (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
135
136 # extract the face ROI, convert it from
137 # ordering, resize it to 224x224, and
138         cropped = input_img[startY:endY, startX:endX]
139         image_resized = cv2.resize(cropped, (224, 224))
140         # preprocessing for face2exp
141         image_resized=image_normalize(image_resized)
142         image_ex_f = np.expand_dims(image_resized, axis=0)
143         pred_f = maskModel.predict(image_ex_f)
144         prob_f = np.squeeze(pred_f)
145
146         if prob_f>0.4:
147             #prob_max = prob_f
148             label = "Mask"
149         else:
150             label = "No_Mask"
151         print(prob_f)
152         print(label)

```

Program in the software(cont-3)

```

13 def draw_label(image, point, label, font:
14             font_scale=0.8, thickness:
15             size = cv2.getTextSize(label, font,
16             x, y = point
17             cv2.rectangle(image, (x, y - size[1]
18             cv2.putText(image, label, point, font
19
20
21 def image_normalization(image):
22     min_v = image.min()
23     max_v = image.max()
24
25     normalized=(image-min_v)/(max_v-min_v
26
27     return normalized
28
29
30 #def main():
31
32
33 #maskModel = keras.models.load_model("D:
34 #trained_model = keras.models.load_model(
35 #detector = dlib.get_frontal_face_detector
36
37 objectAPI = ObjectDetection(r"C:\Users\Sri
38
39 #objectAPI = ObjectDetection(r"D:\Aishu\
40 #
41 #
42 #prototxtPath = r"D:\Aishu\FaceMask-Dete
43 #weightsPath = r"D:\Aishu\FaceMask-Dete
44 #faceNet = cv2.dnn.readNet(prototxtPath,
45
46 #cap = cv2.VideoCapture(0)
47
48 cap = cv2.VideoCapture(r"C:\Users\SriAnna
49
50 #cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)

```

Program in the software(cont-4)

```

59         # bounding_boxes = detector(input_
60
61         boxes, scores, classes, num = ob
62         humBoxes, humScore = humanBoxesAd
63
64         margin = 44
65
66         # (h, w) = input_img.shape[:2]
67         #
68         # blob = cv2.dnn.blobFromImage(inp
69         # (104.0, 177.0, 123.0))
70         # faceNet.setInput(blob)
71         # detections = faceNet.forward()
72         #
73         a = []
74         b = []
75
76         if len(humBoxes) > 0:
77             for i in range(len(humBoxes)):
78                 (x, y) = (boxes[i][0]+(b
79                 # (w, h) = (boxes[i][2], h
80                 a.append(x)
81                 b.append(y)
82
83                 distance=[]
84                 nsd = []
85                 for i in range(0, len(a)-1):
86                     for k in range(1, len(a)):
87                         if(k==i):
88                             break
89                         else:
90                             x_dist = (a[k] - a[i]
91                             y_dist = (b[k] - b[i]
92                             d = math.sqrt(x_dist
93                             distance.append(d)
94                             if(d <=165):
95                                 nsd.append(i)
96                                 nsd.append(k)

```

■

Program in the software (cont-5)

```

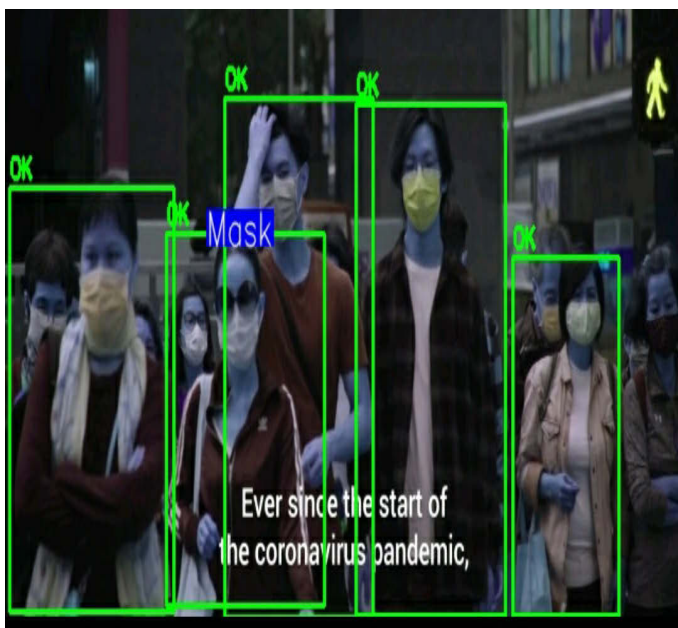
89         else:
90             x_dist = (a[k] - a[i])
91             y_dist = (b[k] - b[i])
92             d = math.sqrt(x_dist**2 + y_dist**2)
93             distance.append(d)
94             if(d <=165):
95                 nsd.append(i)
96                 nsd.append(k)
97             nsd = list(dict.fromkeys(nsd))
98             # print(nsd)
99             color = (0, 0, 255)
100             for i in nsd:
101                 (x, y) = (boxes[i][0], boxes[i][1])
102                 (w0, h0) = (boxes[i][2], boxes[i][3])
103                 cv2.rectangle(input_img, (x, y), (x+w0, y+h0), color, 2)
104                 text = "Alert"
105                 cv2.putText(input_img, text, (x+w0, y+h0), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 1)
106
107             color = (0, 255, 0)
108             if len(humBoxes) > 0:
109                 for i in range(len(humBoxes)):
110                     if (i in nsd):
111                         break
112                     else:
113                         (x, y) = (humBoxes[i][0], humBoxes[i][1])
114                         (w1, h1) = (humBoxes[i][2], humBoxes[i][3])
115                         cv2.rectangle(input_img, (x, y), (x+w1, y+h1), color, 2)
116                         text = 'OK'
117                         cv2.putText(input_img, text, (x+w1, y+h1), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 1)
118
119
120             cv2.imshow("result", input_img)
121             key = cv2.waitKey(1)
122             if key & 0xFF == ord('q'):
123                 break
124
125     if __name__ == '__main__':
126         main()

```

7.3 Results



Face Mask Detection System



Social Distancing System

■

REFERENCE

- [1] Fan, Tingxiang, Zhiming Chen, Xuan Zhao, Jing Liang, Cong Shen, Dinesh Manocha, Jia Pan, and Wei Zhang. "Autonomous social distancing in urban environments using a quadruped robot." arXiv preprint arXiv:2008.08889 (2020).
- [2] Sathyamoorthy, Adarsh Jagan, Utsav Patel, Moumita Paul, Yash Savle, and Dinesh Manocha. "COVID surveillance robot: Monitoring social distancing constraints in indoor scenarios." Plos one 16, no. 12 (2021): e0259713.
- [3] Das, Arjya, Mohammad Wasif Ansari, and Rohini Basak. "Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV." In 2020 IEEE 17th India Council International Conference (INDICON), pp. 1-5. IEEE, 2020.
- [4] J. Nalewicki, Singapore is Using a Robotic Dog to Enforce Proper Social Distancing During COVID-19, 2020, [online] Available: <https://www.smithsonianmag.com/smart-news/singapore-using-robotic-dog-enforce-proper-social-distancing-during-covid-19-180974912/>.
- [5] A. J. Sathyamoorthy, U. Patel, Y. A. Savle, M. Paul and D. Manocha, "COVID-robot: Monitoring social distancing constraints in crowded scenarios", arXiv:2008.06585, 2020, [online] Available: <http://arxiv.org/abs/2008.06585>.
- [6] Ravi Khandelwal, Manish Kumar Mukhija, Satish Kumar Alaria, "Numerical Simulation and Performance Assessment of Improved Particle Swarm Optimization Based Request Scheduling in Edge Computing for IOT Applications", New Arch-International Journal Of Contemporary Architecture, vol-8, issue-2, pp. 155-169, 2021.

- [7] G. K. Soni, S. Gour, Mr. K. Agarwal, A. Sharma, C. S. Shekhawat, B. Sharma, “ IOT Based Smart Agriculture Monitoring System”, Design Engineering, Issue-6, pp. 2243- 2253, 2021.
- [8] Manish Mukhija, “A Resourceful Technique for virtual Machine Migration in Fog Computing”, International Journal of Innovative Science and Research Technology Vol.6, Issue 6, pp 167-170, June 2016
- [9] Kumar, Adarsh, Kriti Sharma, Harvinder Singh, Sagar Gupta Naugriya, Sukhpal Singh Gill, and Rajkumar Buyya. "A drone-based networked system and methods for combating coronavirus disease (COVID-19) pandemic." Future Generation Computer Systems, 2021.
- [10] Fu, Shuangquan, and Pritesh Chandrashekhar Bhavsar. "Robotic arm control based on internet of things." In 2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT), pp. 1-6. IEEE, 2019.
- [11] Priyanka, P. "Iot Based Covid-19 Patient Health Monitoring System" Turkish Journal of Computer and Mathematics Education (TURCOMAT) 12, no. 12, pp. 4599-4604, 2021.
- [12] Pooja Saini, Manish Mukhija, Prakash Dangi, “Design simulation of Efficient Character Recognition System Using Improved PreProcessing and Feature Extraction Process”, Journal of The Gujrat Research Society, Vol.-21, Issue 16, pp. 1883-1889, ISSN: 0374-8588, December 2019.
- [13] Ennafiri, M., and T. Mazri. "Internet of Things for Smart Healthcare: A Review on a Potential IOT Based System and Technologies to Control COVID-19 Pandemic." Innovations in Smart Cities Applications, Volume 4, 2021.

- [14] Monika Mehra, Manish Kumar, Anjali Maurya, Charu Sharma, Shanu, “MERN Stack Web Development”, *Annals of the Romanian Society for Cell Biology*, 25(6), 11756–11761, 2021.
- [15] Zhao, Zhuo, Yangmyung Ma, Adeel Mushtaq, Abdul M. Azam Rajper, Mahmoud Shehab, Annabel Heybourne, Wenzhan Song, Hongliang Ren, and Zion Tsz Ho Tse. "Applications of Robotics, AI, and Digital Technologies During COVID-19: A Review." *Disaster Medicine and Public Health Preparedness*, pp. 1-23, 2021.
- [16] Siddiqui, Shama, Muhammad Zeeshan Shakir, Anwar Ahmed Khan, and Indrakshi Dey. "Internet of Things (IoT) Enabled Architecture for Social Distancing During Pandemic." *Frontiers in Communications and Networks*, 2021.
- [17] Manish Mukhija, “A Resourceful Technique for virtual Machine Migration in Fog Computing”, *International Journal of Innovative Science and Research Technology*, vol-6, issue-6, pp. 167- 170, 2016.