IOT-UNIT-2

Syllabus :

Prototyping IoT Objects using Microprocessor/Microcontroller :

Working principles of sensors and actuators, setting up the board – Programming for IoT, Reading from Sensors, Communication: communication through Bluetooth, Wi-Fi.

2.1. Working principles of sensors and actuators :

Introduction to Sensors & Actuators:

Although Sensors and actuators are often found in similar areas of applications i.e. in equipment and systems within industries and often interact, they are still two separate components. In one line Sensors are detectors whereas actuators are mover. Both play an important role in process control units and conditional maintenance.

What are sensors?

Sensors are such devices which are used to convert physical quantities, events or characteristics into the electrical signals for the purpose of monitoring and controlling. So sensor takes input from environment and converts into electrical form then fed to the system or controller. Sensor works as an input device. For example- Thermocouple, photo cell, RTD, LVDT, strain gauge, Load cell etc.

A block diagram for sensor is shown below

Figure- Block Diagram of a Sensor

Types of IoT Sensors

There are many different types of sensors, and they come in different shapes and sizes. Here are some of them.

1. Temperature Sensors

Temperature sensors measure the amount of heat generated from an area or an object.

2. Proximity Sensors

Proximity sensors detect the presence or absence of objects near the sensor without physical contact. They often emit a beam of radiation like infrared or an electromagnetic field.

3. Pressure Sensors

These sensors detect changes in a gas or liquid. When the pressure range is beyond a set threshold, pressure sensors alert to the problem. They are used for leak testing, water systems, vehicles, and aircraft.

4. Water Quality Sensors

Water quality sensors monitor the quality of water. They are often used in water distribution systems. There are different kinds of water sensors, including residual chlorine sensors, turbidity sensors, pH sensors, and total organic carbon sensors.

5. Chemical and Gas Sensors

These sensors monitor air quality for the presence of toxic or hazardous gas. They often use semiconductor, electrochemical, or photo-ionization technologies for detection.

6. Infrared Sensors

Some sensors either detect or emit infrared radiation to sense characteristics and changes in the surrounding area. They're useful for measuring heat emissions from an object. Infrared sensors are used in remote controls, healthcare

7. Smoke Sensors

Most people are familiar with smoke detectors, as they have protected our homes and businesses for a long time. However, with improvements based on IoT, smoke detectors are now more user-friendly, convenient, and wire-free.

8. Motion Sensors

Motion sensors detect physical movement in an area. Of course, these sensors play a significant role in the security industry, but they are used in nearly every industry.

Applications include , automatic door controls, energy management systems, and automated parking systems. Standard motion sensors include ultrasonic, microwave, and passive infrared (PIR).

9. Level Sensors

Level sensors detect the level of various substances, including powder, granular material, and liquids. Industries that use them include water treatment, food and beverage manufacturing, oil manufacturing, and waste management.

10. Image Sensors

These sensors convert optical images into signals and are generally used to display or store files electronically. They are found in radar and sonar, biometric devices, night vision equipment, medical imaging, digital cameras, and even some cars.

11. Humidity Sensors

These sensors measure the amount of water vapor in the air. Typical uses include heating and air conditioning systems (HVAC) and weather monitoring and prediction.

12. Accelerometer Sensors

Accelerometer sensors detect the orientation of an object and the rate of change, including tap, shake, tilt, and positioning.

13. Gyroscope Sensors

A gyroscope sensor measures the angular rate or velocity, or the speed of rotation around an axis. They are generally used for navigation in the auto industry for navigation and anti-skid systems as well as in

[redacted] ones.

14. Optical Sensors

Optical sensors measure light and convert it into electrical signals. Many industries make use of optical sensors, including auto, energy, healthcare, and aerospace. Sensors include fiber optics, photodetector, and pyrometer.

Working Principles of Sensors :

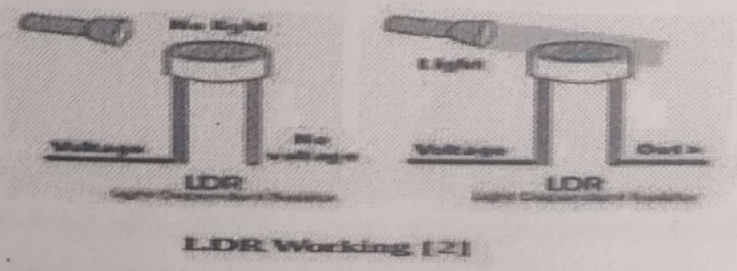
Depending on the application, a system may comprise one or more sensors, sensing a different physical quantity, thereby having a unique sensing mechanism. The two of the most popular sensing mechanisms in MEMS(**Micro-electromechanical Systems**) technology that convert a physical change into an electrical signal are:

1. Resistive based sensing
2. Capacitive based sensing

The sensing mechanism in both the types uses a simple principle – any change in the physical quantity is captured by a change in electrical resistance or capacitance of the material used in the sensor. Thus, a larger change in the physical quantity shows a larger change in the resistance or capacitance of the material and vice-versa.

1. Resistive Based Sensing Mechanism (Using MEMS Technology)

We have been using resistive resistors to measure, analyze, control and observe various physical quantities for over a century. As mentioned earlier, whenever a physical quantity (like pressure) changes, the amount of change in the electrical resistance determines how much the quantity has changed.



Resistive Based Sensors

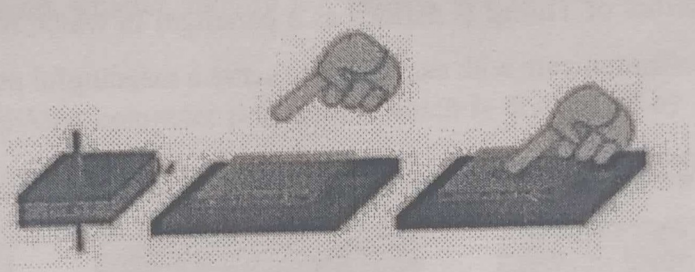
2. *Capacitive Based Sensing Mechanism in IoT Sensors*

A capacitive-based sensing mechanism captures the change in physical quantity by changing the material's capacitance and, like resistance, depends on the material's physical geometry.

A touch sensor is one of the most common capacitive based sensors in an IoT system. A smartphone uses a touch screen consisting of numerous touch sensors. Essentially, it is a pressure sensor that detects the pressure/force from physical touch.

When the screen is stimulated by physical touch, the pressure exerted changes the area or/and distance, which triggers a change in the value of the capacitance underneath the screen.

This change in capacitance acts like an electrical switch that drives an electrical signal to the next stage.



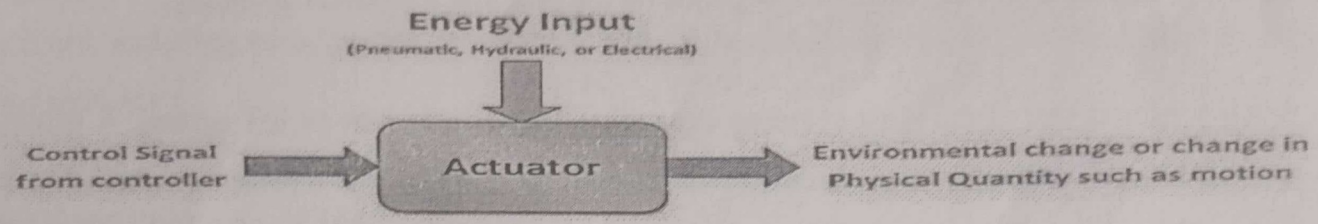
Capacitive Touch Sensor

Working Principles of Actuators?

An actuator is a mechanical or electromechanical device that transforms energy into motion. It accomplishes this by fusing an energy source with an electrical signal. In short, it is the part of any machine that makes movement possible.

In IoT, an actuator is responsible for the physical movement of an object. It can be a device that can move things and is powered using different sources such as the battery, electric, or manually-generated energy. Actuator acts as output device.

A block diagram of actuator is shown below-



Actuator examples- motor actuator, servo motor, stepper motor, heaters, electro pneumatic actuator, electro-hydraulic actuator, magnetic actuator etc.

Sensors and actuators in Iot

The Internet of Things is defined as a paradigm in which objects equipped with sensor, actuator and processor communicate with each other to serve a meaningful purpose.

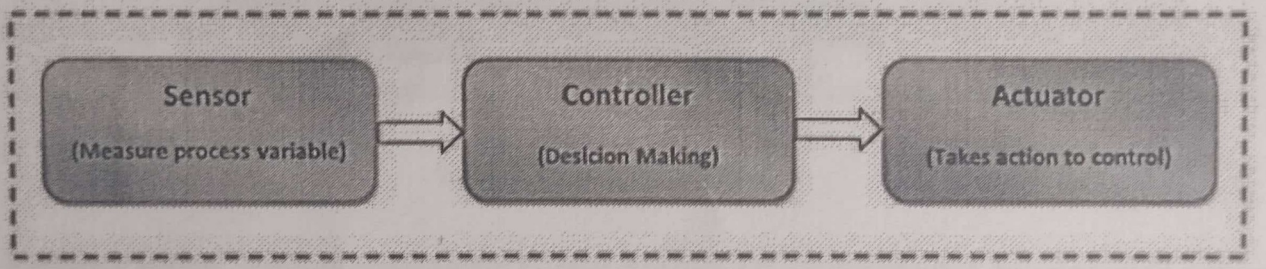


Figure- Sensor and actuator in a system

2.2. Setting up the Board :

MICRO CONTROLLERS :

- Internet of Things devices take advantage of more tightly integrated and miniaturised solutions—from the most basic level of microcontrollers to more powerful system-on-chip (SoC) modules.
- These systems combine the processor, RAM, and storage onto a single chip, which means they are much more specialised, smaller than their PC equivalents, and also easier to build into a custom design.
- These microcontrollers are the engines of countless sensors and automated factory machinery.
- Microcontrollers are very limited in their capabilities. Usually, they offer RAM capabilities measured in kilobytes and storage in the tens of kilobytes.
- The modern chips are much smaller, require less power, and run about five times faster than their 1980s counterparts.
- Unlike the market for desktop computer processors, which is dominated by two manufacturers (Intel and AMD), the microcontroller market consists of many manufacturers.
  (Atmel, Microchip, NXP, Texas Instruments, etc.,)

SYSTEM-ON-CHIPS (MICRO PROCESSORS)

- In between the low-end microcontroller and a full-blown PC, sits the SoC.
- Like the microcontroller, these SoCs combine a processor and a number of peripherals onto a single chip but usually have more capabilities.
- The processors usually range from a few hundred megahertz to the gigahertz. RAM measured in megabytes rather than kilobytes.
- Storage for SoC modules tends not to be included on the chip, with SD cards being a popular solution.
- The greater capabilities of SoC mean that they need some sort of operating system to marshal their resources.
- A wide selection of embedded operating systems, both closed and open source, is available and from both specialised embedded providers and the big OS players, such as Microsoft and Linux.

1. ARDUINO :

- These days the Arduino project covers a number of microcontroller boards, but its birth was in Ivrea in Northern Italy in 2005.
- A group from the Interaction Design Institute Ivrea (IDII) wanted a board for its design students to use to build interactive projects.

DEVELOPING ON THE ARDUINO :

i) Integrated Development Environment (IDE) :

- Usually develop against the Arduino using the integrated development environment (IDE) that the team supply at http://arduino.cc.
- Although this is a fully functional IDE, based on the one used for the Processing language , it is very simple to use.
- Most Arduino projects consist of a single file of code, so you can think of the IDE mostly as a simple file editor. The controls that you use the most are those to check the code (by compiling it) or to push code to the board.

ii) Pushing Code :

- Connecting to the board should be relatively straightforward via a USB cable. Sometimes you might have issues with the drivers or with permissions on the USB port. After this, choose the correct serial port and the board type.

iii) Operating System :

- The Arduino doesn't, by default, run an OS as such, only the bootloader, which simplifies the code-pushing process described previously.
- When you switch on the board, it simply runs the code that you have compiled until the board is switched off again (or the code crashes).
- It is, however, possible to upload an OS to the Arduino, usually a lightweight real-time operating system (RTOS) such as FreeRTOS/DuinOS.
- The main advantage of one of these operating systems is their built-in support for multitasking.

iv) Language :

- The language usually used for Arduino is a slightly modified dialect of C++ derived from the Wiring platform. It includes some libraries used to read and write data from the I/O pins provided and to do some basic handling for "interrupts".

- The code needs to provide only **two routines:**
  - **setup()**: This routine is run once when the board first boots.Use it to set the modes of I/O pins to input or output or to prepare a **data structure** which will be used throughout the program.
  - **Loop()**: This routine is run repeatedly in a tight loop while the Arduino is switched on. Typically, you might check some input, do some calculation on it, and perhaps do some output in response.

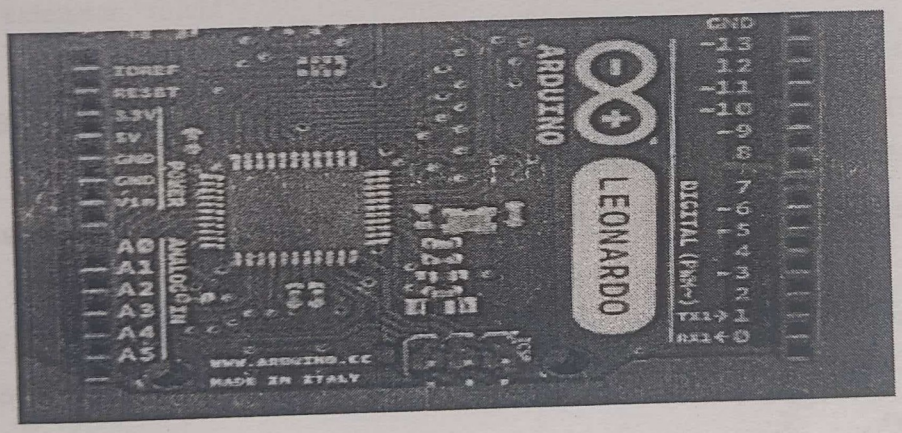
v) Debugging :

The first commercially available version of the WhereDial has a bank of half a dozen LEDs specifically for consumer-level debugging. In the case of an error, the pattern of lights showing may help customers fix their problem or help flesh out details for a support request.

- The Arduino IDE provides a **serial monitor** which echoes the data that the Arduino has sent over the USB cable. This could include any textual information, such as logging information, comments, and details about the data that the Arduino is receiving and processing.

SOME NOTES ON THE HARDWARE :

- The Arduino exposes a number of GPIO pins and is usually supplied with "headers". The headers are optimised for prototyping and for being able to change the purpose of the Arduino easily.



- Each pin is clearly labelled on the controller board. The details of pins vary from the smaller boards such as the Nano, the classic form factor of the Uno, and the larger boards such as the Mega or the Due. In general, you have power outputs such as 5 volts or 3.3 volts (usually labelled 5V and 3V3, or perhaps just 3V), one or more electric ground connections (GND), numbered digital pins, and numbered analogue pins prefixed with an *A*.

- The LilyPad has an entirely different specialism, as it has a flattened form (shaped, as the name suggests, like a flower with the I/O capabilities exposed on its "petals") and is designed to make it easy to wire up with conductive thread, and so a boon for wearable technology projects.
- You can power the Arduino using a USB connection from your computer. In the Arduino world, these add-on boards are called *shields, perhaps* because they cover the actual board as if protecting it.
- Some shields provide networking capabilities—Ethernet, WiFi, or Zigbee wireless, for example. Motor shields make it simple to connect motors and servos; there are shields to hook up mobile phone LCD screens; others to provide capacitive sensing; others to play MP3 files or WAV files from an SD card; and all manner of other possibilities.

OPENNESS :

- The Arduino project is completely open hardware and an open hardware success story.
- The only part of the project protected is the Arduino trademark, so they can control the quality of any boards calling themselves an Arduino.
- In addition to the code being available to download freely, the circuit board schematics and even the EAGLE PCB design files are easily found on the Arduino website.
- In some cases, such as with the wireless-focused Arduino Fio board, what starts as a third-party board (it was originally the Funnel IO) is later adopted as an official Arduino-approved board.

2. RASPBERRY PI :

- The Raspberry Pi, unlike the Arduino, wasn't designed for physical computing at all, but rather, for education. The vision of Eben Upton, trustee and cofounder of the Raspberry Pi Foundation, was to build a computer that was small and inexpensive and designed to be programmed and experimented.
- The model names of the Raspberry Pi, "Model A" and "Model B", hark back to the different versions of the BBC Micro. Many of the other trustees of the Raspberry Pi Foundation, officially founded in 2009.
- The following table compares the specs of the latest, most powerful Arduino model, the Due, with the top-end Raspberry Pi Model B:

	Arduino Due	**Raspberry Pi Model B**
CPU Speed	84 MHz	700 MHz ARM11
GPU	None	Broadcom Dual-Core VideoCore IV Media Co-Processor

	Arduino Due	Raspberry Pi Model B
RAM	96KB	512MB
Storage	512KB	SD card (4GB +)
OS	Bootloader	Various Linux distributions, other operating systems available
Connections	54 GPIO pins 12 PWM outputs 4 UARTs SPI bus I²C bus USB 16U2 + native host 12 analogue inputs (ADC) 2 analogue outputs (DAC)	8 GPIO pins 1 PWM output 1 UART SPI bus with two chip selects I²C bus 2 USB host sockets Ethernet HDMI out Component video and audio out

- So, the Raspberry Pi is effectively a computer that can run a real, modern operating system, communicate with a keyboard and mouse, talk to the Internet, and drive a TV/monitor with high-resolution graphics.
- Importantly, the Pi Model B has built-in Ethernet (as does the Arduino Ethernet, although not the Due) and can also use cheap and convenient USB WiFi dongles, rather than having to use an extension "shield".

CASES AND EXTENSION BOARDS :

- Beyond these largely aesthetic projects, extension boards and other accessories are already available for the Raspberry Pi. However, many interesting kits are in development, such as the Gertboard, designed for conveniently playing with the GPIO pins.

DEVELOPING ON THE RASPBERRY PI :

- **Operating System :**

  Although many operating systems can run on the Pi, we recommend using a popular Linux distribution, such as :

  Raspbian:

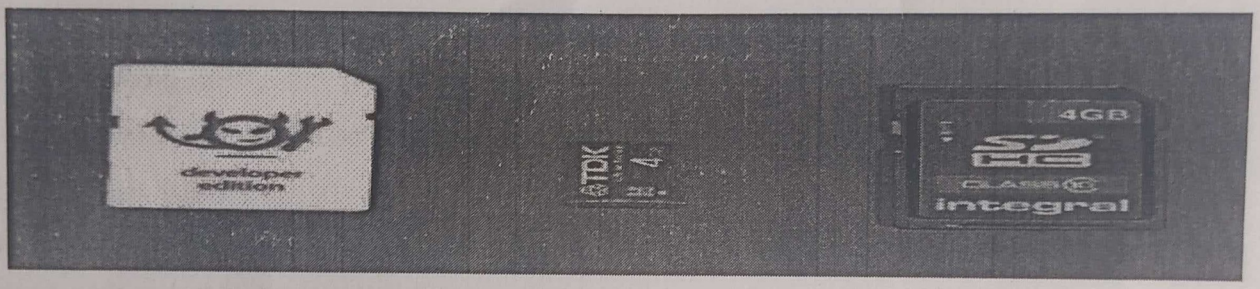
  Released by the Raspbian Pi Foundation, Raspbian is a distro based on Debian. This is the default "official" distribution and is certainly a good choice for general work with a Pi.

Occidentalis:

This is Adafruit's customised Raspbian. Unlike Raspbian,the distribution assumes that you will use it "headless"—not connected to keyboard and monitor—so you can connect to it remotely by default. (Raspbian requires a brief configuration stage first.)

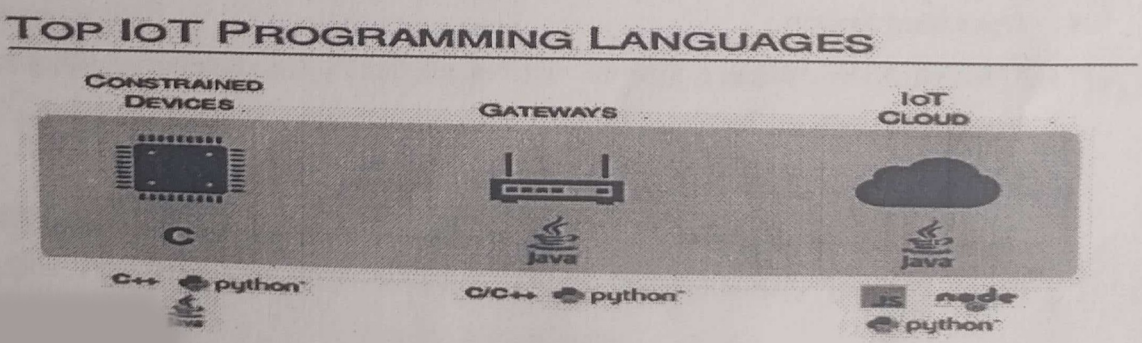
- The sshd (SSH protocol daemon) is enabled by default, so you can connect to the console remotely.
- The device registers itself using zero-configuration networking (zeroconf) with the name raspberrypi.local, so you don't need to know or guess which IP address it picks up from the network in order to make a connection.
- Some SD cards don't work well with the Pi; apparently, "Class 10" cards work best.

- An Electric Imp (left), next to a micro SD card (centre), and an SD card (right).



2.3 Programming for IoT

Programming for IoT is usually a polyglot (multiple languages) effort since the Internet-of-Things (IoT) is a system of inter-related computing devices that are provided with unique identifiers and the ability to transfer data over a network. The choice of programming-language depends on the capability and purpose of the device. IoT encompasses a variety of devices including edge devices, gateways, and cloud servers.



The most popular languages in IoT are **Java, C, C++, Python, Javascript, Node.js, Assembler, PHP, C#, Lua, R, Go, Ruby, Swift and Rust** in descending order of popularity. Other languages include **Parasail, Microsoft P, Eclipse Mita, Kotlin, Dart, MicroPython, and B#.**

Programming paradigms for IoT

A programming paradigm is a way to classify the programming language based on its execution model. It should be kept in mind that a particular language can below to more than one paradigm. The following programming paradigms are suited for IoT:

- **Functional** - Functional programming helps solve the challenges of scalability and concurrency. Its preference for immutability, function composition, avoiding side-effects, less code, etc. avoid several IoT pitfalls.
- **Dataflow** - Dataflow programming emphasizes the movement of data. It models programs as a series of connections. Explicitly defined inputs and outputs connect operations. An operation runs as soon as all of its inputs become valid. Dataflow languages are inherently parallel and can work well in large decentralized systems.
- **Event-Driven** - Event-driven programming (events triggered by sensors, connectivity, or time) is well suited to IoT where devices spend most of their time in power-saving modes, wake up due to some event, process data, send it out, and go back to sleep.

Programming Languages used in different Devices :

- **Edge Devices** - These are constrained-resource embedded systems. For very small devices (higher constraints), **Assembly and C** are the languages of choice. A better processor and more computing power on the device enables one to use **C, Python, Node.js, Java**. The focus is to minimize instruction count, and maximize execution speed and resource management.
- **Gateways** - Gateways manage communication and do analysis of data from many devices through several different buses. More languages can be run on these devices because of their increased computing power, including **C, C++, Java, Python, and Node.js**.

- **Cloud** - With the nearly unlimited computing capability that's available, frameworks like Apache **Hadoop** and **HiveQL** can compute and process large IOT datasets. Statistical computing and visualization can be done using languages like **R** or **Julia.**

2.4 Reading from Sensors :

Reading from sensors is a process of collecting data from physical sensors such as temperature sensors, pressure sensors, accelerometers, and many others. The collected data is then used for various purposes such as monitoring, analysis, and control.

The process of reading from sensors typically involves the following steps:

1. Sensor selection: Choose the appropriate sensor for the specific application and environment. Consider factors such as measurement range, accuracy, and resolution.
2. Connection: Connect the sensor to a microcontroller, data logger, or other device that can read the sensor's output. The connection may involve soldering, plugging in a connector, or wire splicing.
3. Configuration: Configure the device to read the sensor's output. This may involve setting parameters such as measurement range, sampling rate, and filtering.
4. Read data: Read the sensor's output. This may involve sending a command to the sensor to start a measurement and then waiting for the result. The output may be in the form of an analog voltage, digital signal, or serial communication protocol.
5. Processing: Process the sensor data to extract useful information. This may involve filtering, averaging, or converting the data to a different format.
6. Storage: Store the data in a database or file for later analysis or retrieval. This may involve using a microcontroller's built-in memory or an external memory device.
7. Analysis: Analyze the data to gain insights into the system being monitored. This may involve using statistical analysis, machine learning, or other techniques.

Reading from sensors is a fundamental part of many applications, from environmental monitoring to
he right tools and techniques, it is possible to collect and analyze data from
iciency, safety, and performance of many systems.

Reading from sensors involves retrieving data from physical devices that detect and measure various environmental parameters, such as temperature, pressure, humidity, light, sound, and motion.

Depending on the type of sensor and the application, the data can be analog or digital, continuous or discrete, and noisy or precise.

- There are many ways to read from sensors, but some common approaches include:

1. **Analog-to-digital conversion (ADC):**

This involves converting the analog signal from the sensor into a digital signal that can be processed by a computer or microcontroller. The ADC can be integrated into the sensor or external to it, and the resolution and sampling rate can affect the accuracy and latency of the readings.

2. **Serial communication:**

This involves using a protocol such as I2C, SPI, or UART to transmit and receive data between the sensor and the host device. The protocol can define the data format, speed, and addressing, and may require specific hardware or software configurations.

3. **Wireless communication:**

This involves using a wireless protocol such as Bluetooth, Wi-Fi, or LoRa to transmit and receive data between the sensor and the host device. The protocol can define the range, power consumption, and security of the communication, and may require specific antennas, modules, or protocols.

4. **Signal processing:**

This involves applying filters, algorithms, or models to the raw sensor data to extract meaningful information or detect anomalies. The processing can be done in real-time or offline, and can involve machine learning, statistics, or domain-specific knowledge.

- Overall, reading from sensors requires careful selection, configuration, and validation of the hardware, software, and communication protocols, as well as calibration, maintenance, and quality assurance of the sensors themselves.
- It is also important to consider the privacy, ethics, and legal implications of collecting and analyzing sensor data, especially in sensitive or critical applications.

Example for Reading from Sensor :

1. Temperature Sensor:

- One example of a sensor is a temperature sensor. These sensors measure the temperature of the surrounding environment and output a signal that can be read by a microcontroller or computer.
- To read from a temperature sensor, you would typically connect it to an analog input on a microcontroller or an ADC (analog-to-digital converter) if you're using a computer.
- The analog voltage output from the sensor is then converted to a digital value that represents the temperature. Here's some example code in Python for reading from a DS18B20 temperature sensor using a Raspberry Pi:

```
import os
import glob
import time
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'
def read_temp_raw():
    f = open(device_file, 'r')
```

```
    f.close()
    return lines
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c
while True:
    print(read_temp())
    time.sleep(1)
```

2. Light Sensor:

- Another example of a sensor is a light sensor. These sensors measure the amount of light in the environment and output a signal that can be read by a microcontroller or computer. To read from a light sensor, you would typically connect it to an analog input on a microcontroller or an ADC if you're using a computer.
- The analog voltage output from the sensor is then converted to a digital value that represents the light level. Here's some example code in Arduino for reading from a photoresistor:

```
int photoresistorPin = 0;
void setup() {
```

```
  Serial.begin(9600);
}
void loop() {
  int photoresistorValue = analogRead(photoresistorPin);
  Serial.println(photoresistorValue);
  delay(1000);
}
```

3. Motion Sensor:

- A third example of a sensor is a motion sensor. These sensors detect motion in the environment and output a signal that can be read by a microcontroller or computer. To read from a motion sensor, you would typically connect it to a digital input on a microcontroller.
- When the sensor detects motion, it outputs a high signal on the digital input, which can be read by the microcontroller. Here's some example code in Python for reading from a PIR (passive infrared) motion sensor using a Raspberry Pi:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
pir_pin = 11
GPIO.setup(pir_pin, GPIO.IN)
while True:
  if GPIO.input(pir_pin):
    print("Motion detected!")
  else:
    print("No motion detected.")
  time.sleep(1)
```

2.5 Communication through Bluetooth in IoT :

Bluetooth is a wireless communication technology that is commonly used in Internet of Things (IoT) applications. It allows devices to communicate with each other over short distances (typically up to 10 meters) without the need for cables.

- There are two main types of Bluetooth communication in IoT:

  → Bluetooth Classic

  → Bluetooth Low Energy (BLE)

i) Bluetooth Classic :

Bluetooth Classic is the older and more common type of Bluetooth communication.

ii) Bluetooth Low Energy (BLE) :

BLE is a newer and more energy-efficient version that was introduced with Bluetooth 4.0.

- To establish a Bluetooth connection between two devices, they need to go through a process called pairing. During pairing, the devices exchange security keys to ensure that they are authorized to communicate with each other. Once the pairing process is complete, the devices can communicate over Bluetooth.

- In an IoT system, Bluetooth can be used in a **variety of ways**, such as:

1. **Sensor Data Transfer:**

Bluetooth can be used to transfer sensor data from IoT devices to a central hub or server. For example, a wearable fitness tracker might use Bluetooth to transfer data about a user's heart rate, steps taken, and calories burned to a smart phone app.

2. Control and Configuration:

Bluetooth can also be used to remotely control and configure IoT devices. For example, a smart thermostat might allow users to adjust the temperature settings using a smartphone app that communicates with the thermostat over Bluetooth.

3. Asset Tracking:

Bluetooth can also be used to track the location of IoT devices. This is particularly useful for tracking assets in indoor environments where GPS signals may not be available. For example, Bluetooth beacons can be placed throughout a warehouse to track the location of inventory.

- Here's some example code in Python for establishing a Bluetooth connection between two devices using the PyBluez library:

```
import bluetooth

server_address = "AA:BB:CC:DD:EE:FF" # MAC address of the Bluetooth device you want to connect to

port = 1 # Port number used for the Bluetooth connection

client_socket = bluetooth.BluetoothSocket(bluetooth.RFCOMM)

client_socket.connect((server_address, port))

while True:

    data = client_socket.recv(1024)

    print("Received data:", data)

client_socket.close()
```

- In this example, the code connects to a Bluetooth device with the MAC address "AA:BB:CC:DD:EE:FF" and listens for incoming data. When data is received, it is printed to the console. The connection is closed when the program is terminated.

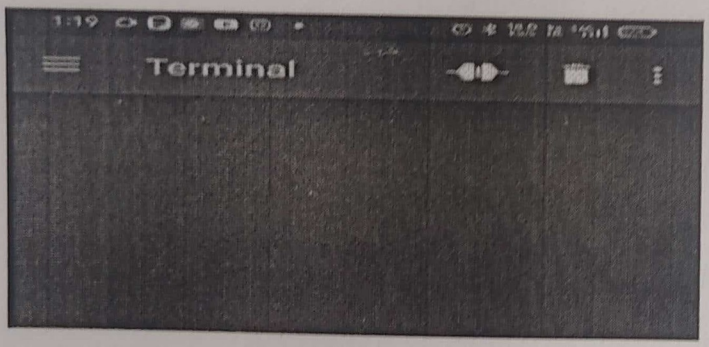
Programming to send "HELLO WORLD" via Bluetooth Communication

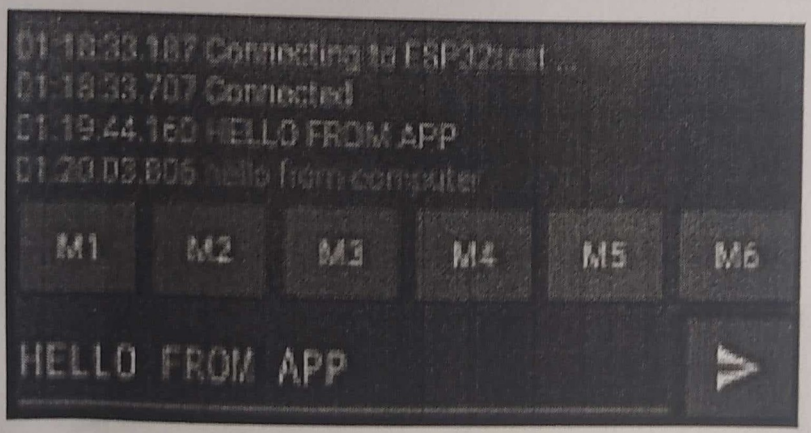
```
#include "BluetoothSerial.h"
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif
BluetoothSerial SerialBT;
void setup() {
Serial.begin(115200);
SerialBT.begin("ESP32test"); //Bluetooth device name
Serial.println("The device started, now you can pair it with bluetooth!");
}
void loop() {
if (Serial.available()) {
SerialBT.write(Serial.read());
}
if (SerialBT.available()) {
Serial.write(SerialBT.read());
}
delay(20);
}
```

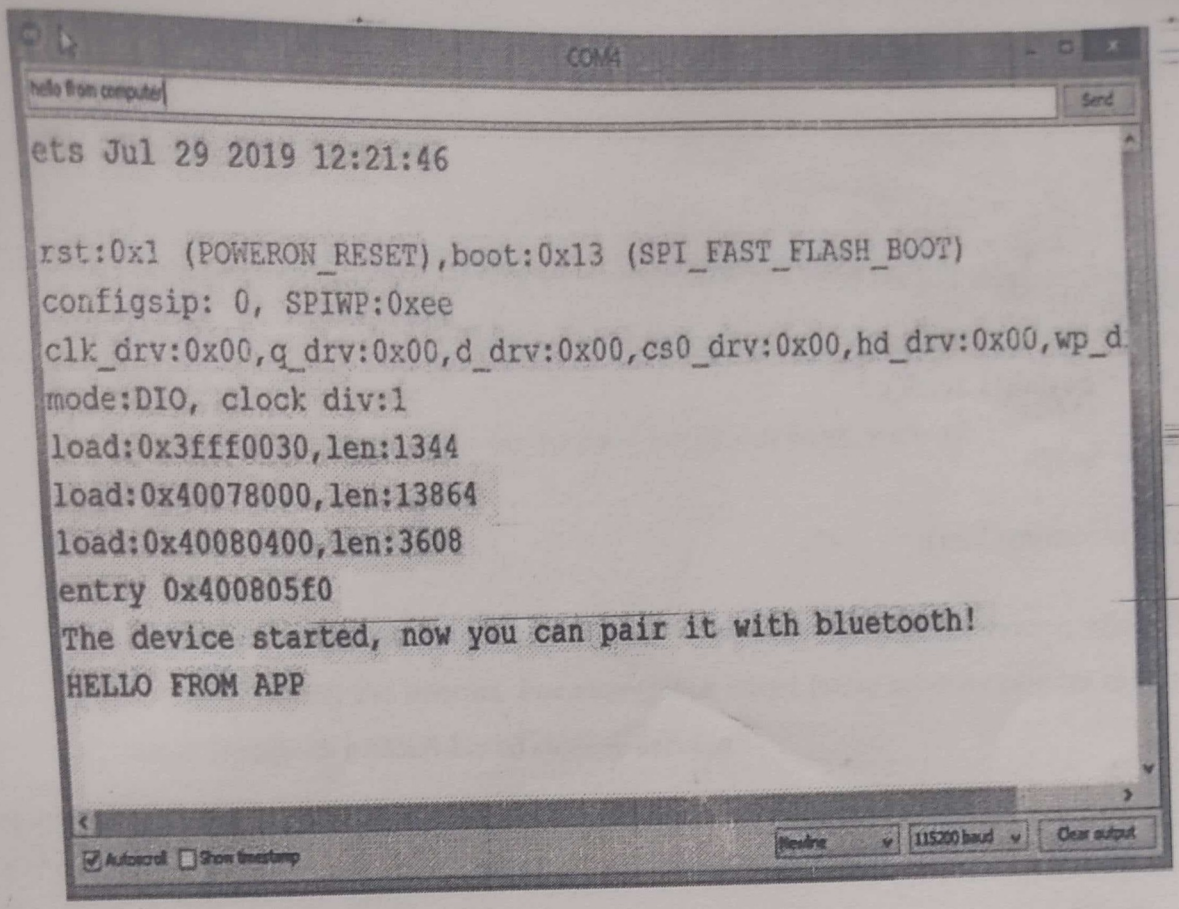
Serial Bluetooth Terminal

Kai Morich

In-app purchases








```
#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)

#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it

#endif

#define BUZZER 2

BluetoothSerial SerialBT;

// Handle received and sent messages

String message = "";

char incomingChar;

void setup() {

pinMode(BUZZER, OUTPUT);

Serial.begin(115200);

SerialBT.begin("ESP32"); //Bluetooth device name
```

```
Serial.println("The device started, now you can pair it with bluetooth!");
}
void loop() {
if (SerialBT.available()){
char incomingChar = SerialBT.read();
if (incomingChar != '\n'){
message += String(incomingChar);
}
else{
message = "";
}
Serial.write(incomingChar);
}
// Check received message and control output accordingly
if (message =="on"){
digitalWrite(BUZZER, HIGH);
}
else if (message =="off"){
digitalWrite(BUZZER, LOW);
}
delay(20);
}
```

2.6. Communication through Wi-Fi in IoT :

Wi-Fi is another popular wireless communication technology that is widely used in Internet of Things (IoT) applications. Wi-Fi is particularly useful for IoT devices that require high bandwidth communication or need to transfer large amounts of data over longer distances.

In an IoT system, Wi-Fi can be used in several ways, such as:

1. **Cloud Connectivity:**

Wi-Fi can be used to connect IoT devices to cloud-based services, allowing them to send and receive data over the internet. For example, a smart home security camera might use Wi-Fi to upload video footage to a cloud-based storage service.

2. **Local Network Connectivity:**

Wi-Fi can also be used to connect IoT devices to a local network, such as a home or office Wi-Fi network. This allows the devices to communicate with each other and with other devices on the network.

3. **Remote Control and Monitoring:**

Wi-Fi can be used to remotely control and monitor IoT devices using a smart phone or computer. For example, a smart thermostat might allow users to adjust the temperature settings from their smart phone.

- Here's an example code in Python for establishing a Wi-Fi connection between an IoT device and a Wi-Fi network using the PyWiFi library:

```
import pywifi
wifi = pywifi.PyWiFi()
iface = wifi.interfaces()[0] # Select the first available Wi-Fi interface
iface.disconnect() # Disconnect from any currently connected networks
                   fi.Profile() # Create a new Wi-Fi profile
```

```
profile.ssid = "MyWi-Fi-Network" # Set the SSID of the Wi-Fi network

profile.auth = pywifi.const.AUTH_ALG_OPEN # Set the authentication algorithm to open

profile.akm.append(pywifi.const.AKM_TYPE_WPA2PSK) # Set the AKM (Authentication and
Key Management) type to WPA2-PSK

profile.cipher = pywifi.const.CIPHER_TYPE_CCMP # Set the cipher type to CCMP (Advanced
Encryption Standard)

profile.key = "MyWi-Fi-Password" # Set the Wi-Fi password

iface.remove_all_network_profiles() # Remove any existing Wi-Fi profiles

iface.add_network_profile(profile) # Add the new Wi-Fi profile

iface.connect() # Connect to the Wi-Fi network

while not iface.status() == pywifi.const.IFACE_CONNECTED:

  pass # Wait for the connection to be established

print("Connected to Wi-Fi network", iface.ssid())
```

- In this example, the code connects to a Wi-Fi network with the SSID "MyWi-Fi-Network" and the password "MyWi-Fi-Password". The connection is established using the WPA2-PSK security protocol with AES encryption. The code waits for the connection to be established before printing a message to the console.

****