**SMART PARKING MANAGEMENT SYSTEM USING IOT FOR REAL-TIME SLOT AVAILABILITY AND MONITORING**

**ABSTRACT:**

This project presents a Smart Parking Management System that utilizes Internet of Things (IoT) technology to monitor and display real-time parking slot availability. The system employs two essential sensors—an **ultrasonic sensor** to detect vehicle presence and a **IR sensor** to confirm slot entry/exit activity. These sensors are integrated with an IoT-enabled microcontroller, which collects and transmits the data to a cloud-based platform for visualization through a mobile or web dashboard. Users can check parking availability remotely, reducing time spent searching for parking and improving traffic flow. This solution offers a cost-effective, scalable, and energy-efficient approach to modernizing urban parking infrastructure.

**INTRODUCTION:**

With rapid urbanization and the increasing number of vehicles, finding an available parking space has become a significant challenge in many cities. Traditional parking systems often lead to congestion, wasted time, and increased fuel consumption due to the lack of real-time information. To address this issue, smart parking solutions are being developed using Internet of Things (IoT) technologies to enhance parking efficiency and user convenience.

This project introduces a Smart Parking Management System that leverages IoT and sensor-based technology to monitor and display the real-time status of parking slots. By utilizing just two key sensors—an ultrasonic sensor to detect the presence of a vehicle in a slot, and an IR sensor to monitor entry or exit—the system efficiently manages parking data. The collected information is transmitted to a cloud platform, enabling users to access live parking availability through a mobile or web interface. This not only simplifies the parking experience but also contributes to reducing traffic congestion and optimizing space usage in urban areas.

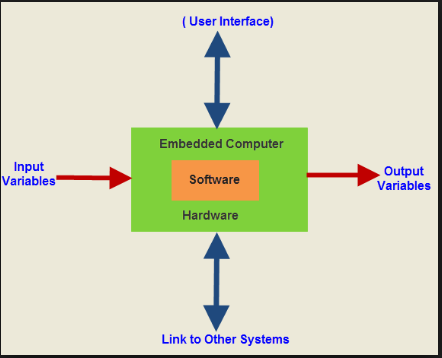
**1.1 EMBEDDED SYSTEM:**

An embedded system is a controller programmed and controlled by a [real-time operating system](https://en.wikipedia.org/wiki/Real-time_operating_system" \o "Real-time operating system) (RTOS) with a dedicated function within a larger mechanical or electrical system, often with [real-time consumption of embedded systems computing](https://en.wikipedia.org/wiki/Real-time_computing" \o "Real-time computing) constraints.

  It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all [microprocessors](https://en.wikipedia.org/wiki/Microprocessor" \o "Microprocessor) are manufactured to serve as embedded system component.

Examples of properties of typical embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with.

However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power.



**Fig 1.1 Embedded system**

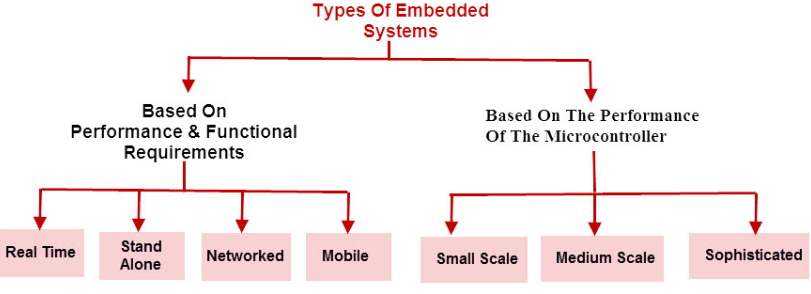
Embedded Computer Sub-Assembly for Electronic Voting Machine.

Embedded systems are commonly found in consumer, industrial, automotive, medical, commercial and military applications.

Telecommunications systems employ numerous embedded systems from [telephone switches](https://en.wikipedia.org/wiki/Telephone_switch" \o "Telephone switch) for the network to [cell phones](https://en.wikipedia.org/wiki/Cell_phone" \o "Cell phone) at the end user. Computer networking uses dedicated [routers](https://en.wikipedia.org/wiki/Router_(computing)" \o "Router (computing)) and [network bridges](https://en.wikipedia.org/wiki/Network_bridge" \o "Network bridge) to route data.

[Consumer electronics](https://en.wikipedia.org/wiki/Consumer_electronics" \o "Consumer electronics) include [MP3 players](https://en.wikipedia.org/wiki/MP3_player" \o "MP3 player), mobile phones, [video game consoles](https://en.wikipedia.org/wiki/Video_game_console" \o "Video game console), [digital cameras](https://en.wikipedia.org/wiki/Digital_camera" \o "Digital camera), [GPS](https://en.wikipedia.org/wiki/Global_Positioning_System" \o "Global Positioning System) receivers, and [printers](https://en.wikipedia.org/wiki/Computer_printer" \o "Computer printer). Household appliances, such as [microwave ovens](https://en.wikipedia.org/wiki/Microwave_oven" \o "Microwave oven), [washing machines](https://en.wikipedia.org/wiki/Washing_machine" \o "Washing machine) and [dishwashers](https://en.wikipedia.org/wiki/Dishwashers" \o "Dishwashers), include embedded systems to provide flexibility, efficiency and features.

**CLASSIFICATIONS OF EMBDDED SYSTEMS:**



**Fig 1.2 Classification of Embedded system**

Advanced [HVAC](https://en.wikipedia.org/wiki/HVAC" \o "HVAC) systems use networked [thermostats](https://en.wikipedia.org/wiki/Thermostat" \o "Thermostat) to more accurately and efficiently control temperature that can change by time of day and [season](https://en.wikipedia.org/wiki/Season" \o "Season). [Home automation](https://en.wikipedia.org/wiki/Home_automation" \o "Home automation) uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Like [traffic lights](https://en.wikipedia.org/wiki/Traffic_light" \o "Traffic light), [factory controllers](https://en.wikipedia.org/wiki/Programmable_logic_controller" \o "Programmable logic controller), and largely complex systems like [hybrid vehicles](https://en.wikipedia.org/wiki/Hybrid_vehicles" \o "Hybrid vehicles), [MRI](https://en.wikipedia.org/wiki/MRI" \o "MRI), and [avionics](https://en.wikipedia.org/wiki/Avionics" \o "Avionics) Embedded systems range from portable devices such as [digital watches](https://en.wikipedia.org/wiki/Digital_watch" \o "Digital watch) and [MP3 players](https://en.wikipedia.org/wiki/Digital_audio_player" \o "Digital audio player), to large stationary installations. Complexity varies from low, with a single [microcontroller](https://en.wikipedia.org/wiki/Microcontroller" \o "Microcontroller)

**BLOCK DIAGRAM OF AN EMBEDDED SYSTEM:**

An embedded system usually contains an embedded processor. Many appliances that have a digital interface microwaves, VCRs, cars utilize embedded systems. Some embedded systems include an operating system. Others are very specialized resulting in the entire logic being implemented as a single program. These systems are embedded into some device for some specific purpose other than to provide general purpose computing.



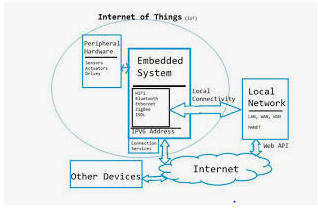
**Fig 1.3 Block diagram of a typical embedded system**

**EMBEDDED SYSTEMS APPLICATIONS:**

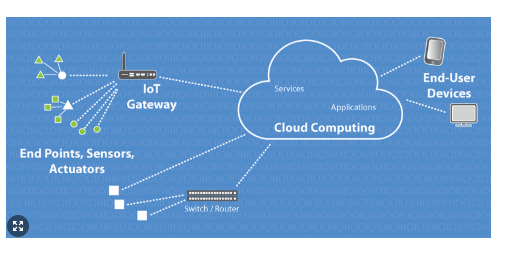
* Embedded systems in automobiles include motor control, cruise control, body safety, engine safety, robotics in an assembly line, car multimedia, car entertainment, E-com access, mobiles etc.
* Embedded systems in telecommunications include networking, mobile computing, and wireless communications, etc.
* Embedded system in smart cards include banking, telephone and security systems.
* Embedded Systems in satellites and missiles include defence, communication, and aerospace
* Embedded systems in computer networking & peripherals include image processing, networking systems, printers, network cards, monitors and displays
* Embedded Systems in digital consumer electronics include set-top boxes, DVDs, high-definition TVs and digital cameras.

**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.

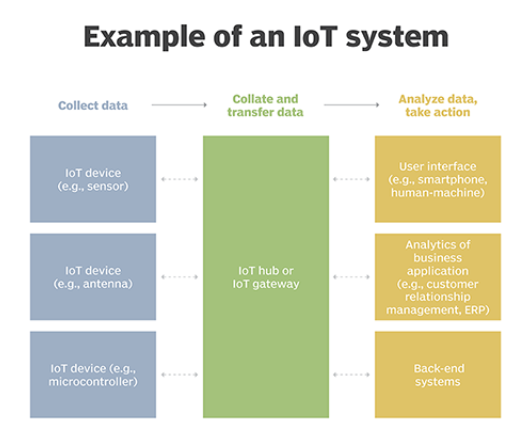


IoT devices are implemented using both hardware and software components. Dedicated hardware components are used to implement the interface with the physical world, and to perform tasks which are more computationally complex. Microcontrollers are used to execute software that interprets inputs and controls 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](https://internetofthingsagenda.techtarget.com/definition/IoT-device) 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.



### 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

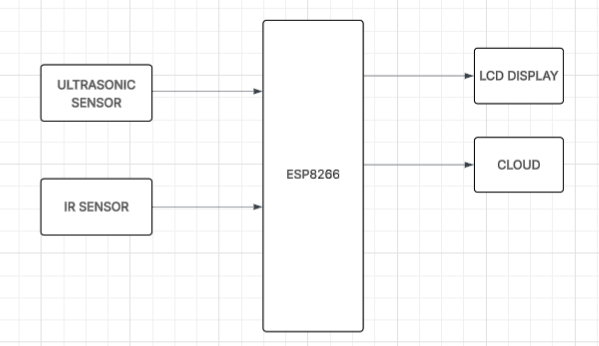
### There are numerous real-world applications of the internet of things, ranging from consumer IoT and enterprise IoT to manufacturing and industrial IoT (IoT). IoT applications span numerous verticals, including automotive, telco, energy and more.

In the consumer segment, for example, [smart homes](https://internetofthingsagenda.techtarget.com/definition/smart-home-or-building) 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.

**BLOCK DIAGRAM:**



**HARDWARE REQUIREMENT:**

* Ultrasonic Sensor
* IR Sensor
* Esp8266

**SOFTWARE REQUIREMENT:**

* Arduino IDE
* Embedded C
* IOT Platform

**HARDWARE COMPONENTS DETAILS**

**NODE MCU ESP8266:**

Node MCU is an open-source Lua based firmware and development board specially targeted for IoT based Applications. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Express if Systems, and hardware which is based on the ESP-12 module.

**Node MCU ESP8266 Specifications & Features:**

* Microcontroller: Tensilica 32-bit RISC CPU Xtensa LX106
* Operating Voltage: 3.3V
* Input Voltage: 7-12V
* Digital I/O Pins (DIO): 16
* Analog Input Pins (ADC): 1
* UARTs: 1
* SPIs: 1
* I2Cs: 1
* Flash Memory: 4 MB
* SRAM: 64 KB
* Clock Speed: 80 MHz
* USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
* PCB Antenna
* Small Sized module to fit smartly inside your IoT projects

**Brief About Node MCU ESP8266:**

The Node MCU ESP8266 development board comes with the ESP-12E module containing ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. Node MCU has 128 KB RAM and 4MB of Flash memory to store data and programs. Its high processing power with in-built Wi-Fi / Bluetooth and Deep Sleep Operating features make it ideal for IoT projects.

Node MCU can be powered using Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface.



**Fig 4.3 Node-MCU diagram**

**Power Requirement:**

As the operating voltage range of ESP8266 is 3V to 3.6V, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as 80mA during RF transmissions. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components.

Power to the ESP8266 Node MCU is supplied via the on-board Micro B USB connector. Alternatively, if you have a regulated 5V voltage source, the VIN pin can be used to directly supply the ESP8266 and its peripherals.

**Peripherals and I/O:**

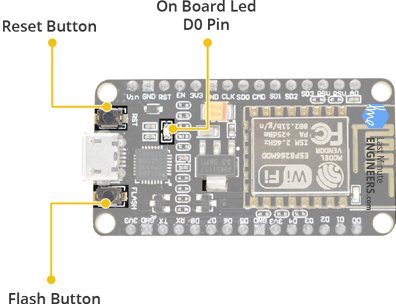
The ESP8266 Node MCU has total 17 GPIO pins broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

* ADC channel – A 10-bit ADC channel.
* UART interface – UART interface is used to load code serially.
* PWM outputs – PWM pins for dimming LEDs or controlling motors.
* SPI, I2C & I2S interface – SPI and I2C interface to hook up all sorts of sensors and peripherals.
* I2S interface – I2S interface if you want to add sound to your project.

**On-board Switches & LED Indicator:**

The ESP8266 Node MCU features two buttons. One marked as RST located on the top left corner is the Reset button, used of course to reset the ESP8266 chip. The other FLASH button on the bottom left corner is the download button used while upgrading firmware.

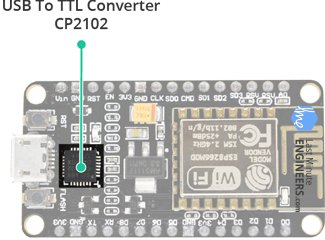
The board also has a LED indicator which is user programmable and is connected to the D0 pin of the board.



**Fig 4.4 Switch diagram**

**Serial Communication:**

The board includes CP2102 USB-to-UART Bridge Controller from Silicon Labs, which converts USB signal to serial and allows your computer to program and communicate with the ESP8266 chip.

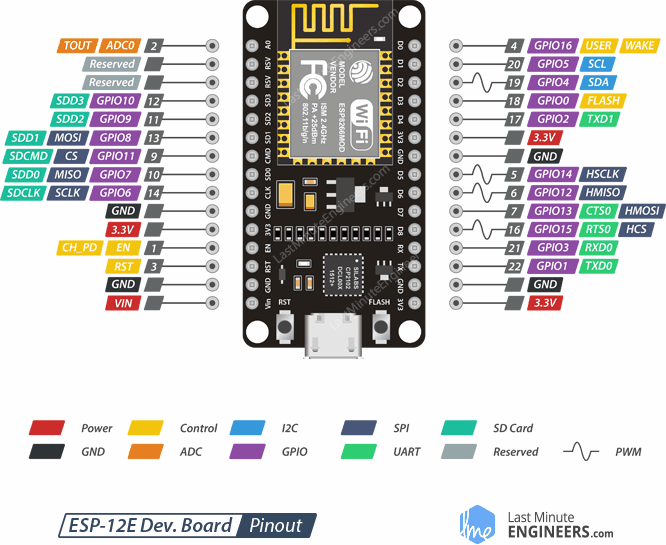


**Fig 4.5 Serial communication diagram**

If you have an older version of CP2102 driver installed on your PC, we recommend upgrading now.

**ESP8266 Node MCU Pinout:**

The ESP8266 Node MCU has total 30 pins that interface it to the outside world. The connections are as follows:



**Fig 4.6 pin out diagram**

For the sake of simplicity, we will make groups of pins with similar functionalities.

**Power Pins:**

There are four power pins viz. one VIN pin & three 3.3V pins. The VIN pin can be used to directly supply the ESP8266 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pins are the output of an on-board voltage regulator. These pins can be used to supply power to external components.

**GND:**

GND is a ground pin of ESP8266 Node MCU development board.

**I2C Pins:**

I2C Pins are used to hook up all sorts of I2C sensors and peripherals in your project. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

**GPIO Pins:**

ESP8266 Node MCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

**ADC Channel:**

The Node MCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC viz. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time.

**UART Pins:**

ESP8266 Node MCU has 2 UART interfaces, i.e. UART0 and UART1, which provide asynchronous communication (RS232 and RS485), and can communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. It supports fluid control. However, UART1 (TXD1 pin) features only data transmit signal so, it is usually used for printing log.

**SPI Pins:**

ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

* 4 timing modes of the SPI format transfer
* Up to 80 MHz and the divided clocks of 80 MHz
* Up to 64-Byte FIFO

**SDIO Pins:**

ESP8266 features Secure Digital Input/Output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.

**PWM Pins:**

The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 μs to 10000 μs, i.e., between 100 Hz and 1 kHz.

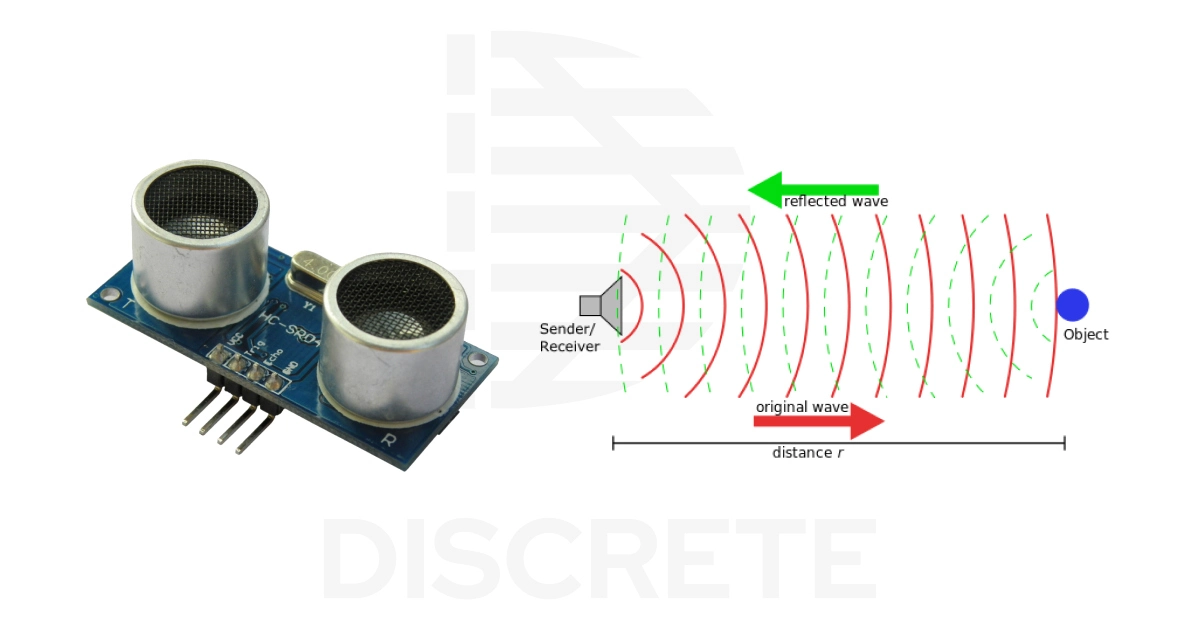
**Control Pins:**

* Control pins are used to control ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.
* EN pin – The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.
* RST pin – RST pin is used to reset the ESP8266 chip.WAKE pin.

**ULTRASONIC SENSOR:**

## What is an Ultrasonic Sensor?

An **ultrasonic sensor** is an electronic device that measures distance by using **ultrasonic sound waves**. It works on the principle similar to **echolocation** used by bats, dolphins, and some robots.



## How Does It Work?

It uses **two main components**:

**Transmitter** – emits a high-frequency sound (ultrasonic pulse)

**Receiver** – listens for the echo of that pulse after it bounces off an object

### Steps:

* **Send**: The transmitter sends an ultrasonic sound wave (usually 40 kHz).
* **Bounce**: The sound wave hits an object and reflects back.
* **Receive**: The receiver detects the reflected wave.
* **Calculate**: The sensor calculates the time between sending and receiving the wave.
* **Distance**: Using the formula:

Distance=Speed of Sound×Time2\text{Distance} = \frac{\text{Speed of Sound} \times \text{Time}}{2}Distance=2Speed of Sound×Time​

(The division by 2 is because the sound travels to the object and back.)

## Example:

Speed of sound = 343 m/s (at room temperature)

Time taken for echo = 0.01 seconds

Distance = 343×0.012=1.715\frac{343 \times 0.01}{2} = 1.7152343×0.01​=1.715 meters



## Applications:

* Obstacle detection in robots
* Water level measurement
* Parking sensors in cars
* People counting systems
* Smart dustbin lid openers

## Types of Ultrasonic Sensors:

* **Proximity Sensors** – Just detect if something is near
* **Ranging Sensors** – Measure exact distance

## Common Ultrasonic Sensor Module: ****HC-SR04****

**Pins:**

* **VCC** – Power supply (5V)
* **Trig** – Trigger input (you send a pulse)
* **Echo** – Echo output (you measure the time pulse stays HIGH)
* **GND** – Ground

**Working:**

* Send 10μs HIGH pulse to Trig pin
* Sensor emits 8 pulses of 40kHz
* Echo pin goes HIGH until echo is received
* Measure how long Echo stays HIGH

### Advantages:

* Non-contact measurement
* Works in darkness
* Simple and inexpensive

### Limitations:

* Affected by soft materials (which absorb sound)
* Not effective in vacuum or with high-noise environments
* Limited to a range (e.g. 2cm to 400cm for HC-SR04)

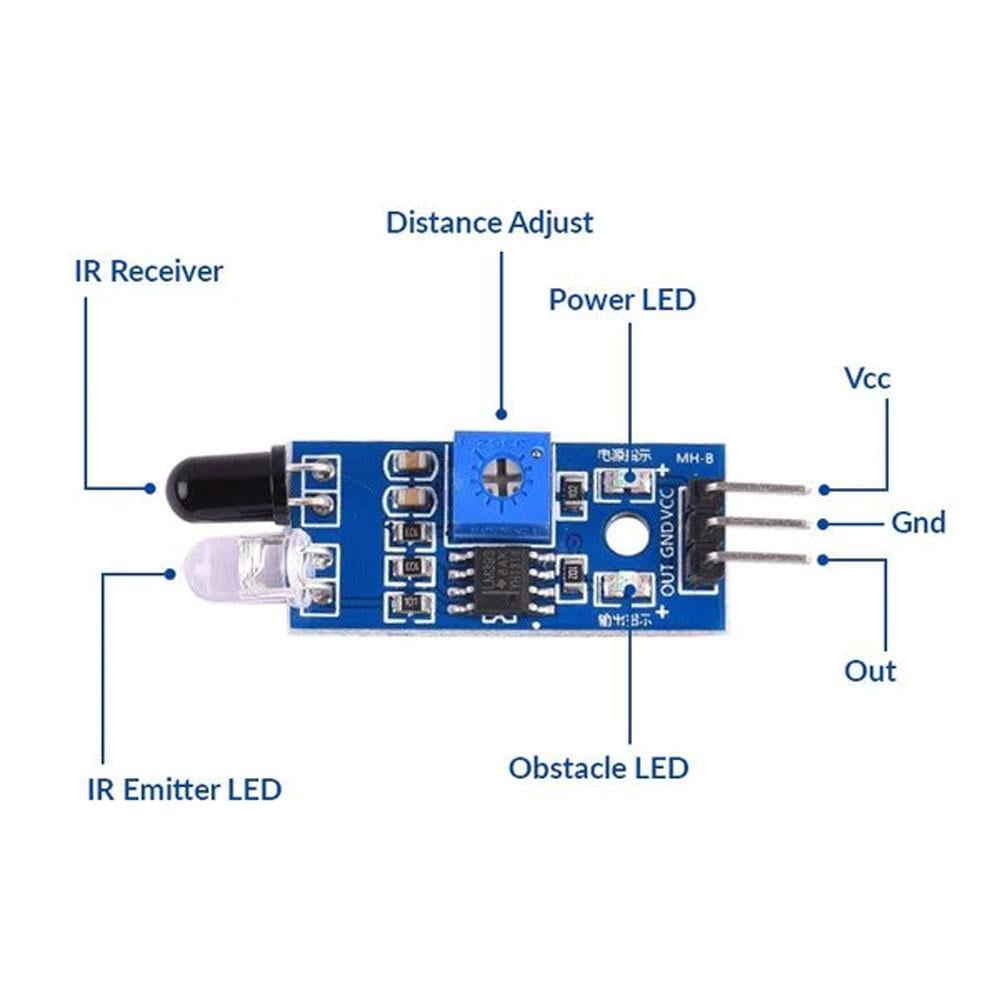
**IR SENSOR:**

## What is an IR Sensor?

* An **IR (Infrared) Sensor** is an electronic device that detects **infrared radiation (IR)** in its surroundings. It is commonly used to **detect objects**, **measure distance**, or **sense motion**.
* IR sensors can be **active** or **passive**, depending on whether they emit IR themselves or just detect it.

## Basic Principle:

All objects emit some amount of **infrared radiation** (heat), which is invisible to the human eye. An IR sensor detects this radiation or measures changes in reflected IR light.



## Types of IR Sensors

### 1. ****Active IR Sensor****

* Contains both an **IR transmitter** (usually an IR LED) and an **IR receiver** (photodiode or phototransistor).
* Works by **emitting** infrared light and detecting the **reflection** from nearby objects.

### 2. ****Passive IR Sensor (PIR)****

* Does **not emit** anything.
* Just detects **infrared radiation** emitted by warm objects like humans or animals.
* Common in motion detectors.

## Components of an Active IR Sensor:

* **IR LED (Transmitter)** – Emits infrared light.
* **Photodiode/Phototransistor (Receiver)** – Detects reflected IR light.
* **Comparator Circuit** – Compares the signal and decides output (used to clean up noise).
* **Output Pin** – HIGH or LOW signal sent to microcontroller.

## How Does It Work?

* IR LED emits invisible infrared light.
* If an object is in front, it reflects some IR light back.
* The photodiode detects the reflected IR light.
* The circuit processes this signal to detect **presence** or **distance**.

## Example: IR Obstacle Sensor Module

**Pins:**

* **VCC** – 3.3V or 5V
* **GND**
* **OUT** – Digital output (HIGH when no object, LOW when object detected)

## Applications

### Active IR Sensor:

* Obstacle detection in robots
* Line follower robots (IR light reflects differently on black/white surfaces)
* Proximity sensors
* Industrial automation

### Passive IR Sensor (PIR):

* Motion detection in security systems
* Automatic lighting
* People counting

**Advantages:**

* Low power consumption
* Simple and cheap
* Non-contact detection
* Reliable for short-range detection

## Limitations:

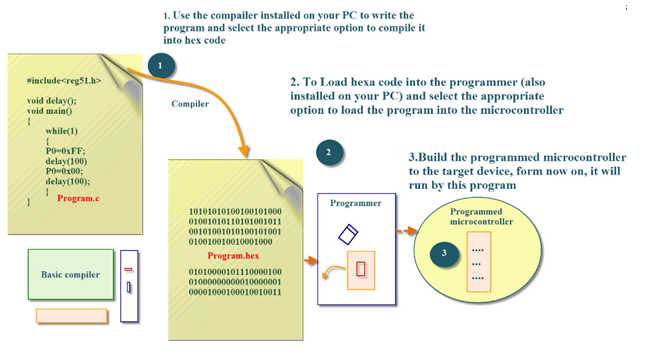
* Affected by ambient IR (sunlight or flame)
* Not suitable for long-range detection
* Reflective surfaces work better than dark/absorbent ones

**SOFTWARE REQUIREMENTS DETAILS**

**EMBEDDED C:**

Embedded C is most popular programming language in software field for developing electronic gadgets. Each processor used in electronic system is associated with embedded software.

Embedded C programming plays a key role in performing specific function by the processor. In day-to-day life we used many electronic devices such as mobile phone, washing machine, digital camera, etc. These all device working is based on microcontroller that are programmed by embedded C.



**Fig 5.1 Block diagram of Embedded C**

The Embedded C code written in above block diagram is used for blinking the LED connected with Port0 of microcontroller.

In embedded system programming C code is preferred over other language. Due to the following reasons:

* Easy to understand
* High Reliability
* Portability
* Scalability

Function is a collection of statements that is used for performing a specific task and a collection of one or more functions is called a programming language.

Most consumers are familiar with [application software](https://en.m.wikipedia.org/wiki/Application_software" \o "Application software) that provide functionality on a computer. Embedded software however is often less visible, but no less complicated. Unlike application software, embedded software has fixed hardware requirements and capabilities, and addition of third-party hardware or software is strictly controlled.

Embedded software needs to include all needed [device drivers](https://en.m.wikipedia.org/wiki/Device_driver" \o "Device driver) at manufacturing time, and the device drivers are written for the specific hardware. The software is highly dependent on the CPU and specific chips chosen. Most embedded software engineers have at least a passing knowledge of reading [schematics](https://en.m.wikipedia.org/wiki/Schematic" \o "Schematic), and reading data sheets for components to determine usage of registers and communication system. Conversion between [decimal](https://en.m.wikipedia.org/wiki/Decimal" \o "Decimal), [hexadecimal](https://en.m.wikipedia.org/wiki/Hexadecimal" \o "Hexadecimal) and [binary](https://en.m.wikipedia.org/wiki/Binary_numeral_system" \o "Binary numeral system) is useful as well as using [bit manipulation](https://en.m.wikipedia.org/wiki/Bit_manipulation" \o "Bit manipulation).

Web applications are rarely used, although XML files and other output may be passed to a computer for display. File systems with folders are typically absent as are SQL databases.

Software development requires use of a [cross compiler](https://en.m.wikipedia.org/wiki/Cross_compiler" \o "Cross compiler), which runs on a computer but produces executable code for the target device. Debugging requires use of an [in-circuit emulator](https://en.m.wikipedia.org/wiki/In-circuit_emulator" \o "In-circuit emulator), [JTAG](https://en.m.wikipedia.org/wiki/JTAG" \o "JTAG) or [SWD](https://en.m.wikipedia.org/wiki/JTAG" \l "Serial_Wire_Debug" \o "JTAG). Software developers often have access to the complete kernel (OS) source code.

Size of the storage memory and RAM can vary significantly. Some systems run in 16 KB of Flash and 4 KB of RAM with a CPU operating at 8 MHz, other systems can rival contemporary computers.[[8]](https://en.m.wikipedia.org/wiki/Embedded_software" \l "cite_note-8) These space requirements lead to more work being done in C or [embedded C++](https://en.m.wikipedia.org/wiki/Embedded_C%2B%2B" \o "Embedded C++), instead of C++. Interpreted languages like [BASIC](https://en.m.wikipedia.org/wiki/BASIC" \o "BASIC) (while e.g. [Parallax Propeller](https://en.m.wikipedia.org/wiki/Parallax_Propeller" \l "BASIC_compiler" \o "Parallax Propeller) can use compiled BASIC) and [Java](https://en.m.wikipedia.org/wiki/Java_programming_language" \o "Java programming language) (Java ME Embedded 8.3[[9]](https://en.m.wikipedia.org/wiki/Embedded_software" \l "cite_note-9) is available for e.g. [ARM Cortex-M4](https://en.m.wikipedia.org/wiki/ARM_Cortex-M4" \o "ARM Cortex-M4), [Cortex-M7](https://en.m.wikipedia.org/wiki/Cortex-M7" \o "Cortex-M7) microcontrollers and older [ARM11](https://en.m.wikipedia.org/wiki/ARM11" \o "ARM11)used in [Raspberry Pi](https://en.m.wikipedia.org/wiki/Raspberry_Pi" \o "Raspberry Pi) and [Intel Galileo](https://en.m.wikipedia.org/wiki/Intel_Galileo" \o "Intel Galileo) Gen. 2) are not commonly used; while an implementation of the interpreted [Python](https://en.m.wikipedia.org/wiki/Python_(programming_language)" \o "Python (programming language)) 3 language – [MicroPython](https://en.m.wikipedia.org/wiki/MicroPython" \o "MicroPython) – is however available expressly for microcontroller use, e.g. [32-bit](https://en.m.wikipedia.org/wiki/32-bit" \o "32-bit) [ARM](https://en.m.wikipedia.org/wiki/List_of_ARM_Cortex-M_development_tools" \o "List of ARM Cortex-M development tools)-based (such as BBC [micro:bit](https://en.m.wikipedia.org/wiki/Micro_Bit" \o "Micro Bit)) and [16-bit](https://en.m.wikipedia.org/wiki/16-bit" \o "16-bit) [PIC microcontrollers](https://en.m.wikipedia.org/wiki/PIC_microcontroller" \o "PIC microcontroller).Communications between processors and between one processor and other components are essential. Besides [direct memory addressing](https://en.m.wikipedia.org/wiki/Memory_address" \o "Memory address), common protocols include [I²C](https://en.m.wikipedia.org/wiki/I%C2%B2C" \o "I²C), [SPI](https://en.m.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus" \o "Serial Peripheral Interface Bus), [serial ports](https://en.m.wikipedia.org/wiki/Serial_port" \o "Serial port), and [USB](https://en.m.wikipedia.org/wiki/Universal_Serial_Bus" \o "Universal Serial Bus).Communications protocols designed for use in [embedded systems](https://en.m.wikipedia.org/wiki/Embedded_systems" \o "Embedded systems) are available as [closed source](https://en.m.wikipedia.org/wiki/Closed_source" \o "Closed source) from companies including [InterNiche Technologies](https://en.m.wikipedia.org/wiki/NicheStack_TCP/IPv4" \o "NicheStack TCP/IPv4) and [CMX Systems](https://en.m.wikipedia.org/wiki/CMX_Systems" \o "CMX Systems). [Open-source](https://en.m.wikipedia.org/wiki/Open-source_model" \o "Open-source model) protocols stem from [uIP](https://en.m.wikipedia.org/wiki/UIP_(micro_IP)" \o "UIP (micro IP)), [lwip](https://en.m.wikipedia.org/wiki/LwIP" \o "LwIP), and others.This program explains how to use structure within structure in C using normal variable. “student\_college\_detail’ structure is declared inside “student\_detail” structure in this program. Both structure variables Ple

**ARDUINO IDE:**

[Arduino](http://arduino.cc/) is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a [microcontroller](http://en.wikipedia.org/wiki/Microcontroller)) and a piece of [software](http://arduino.cc/en/Main/Software), or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

The Arduino hardware and software were designed for artists, designers, hobbyists, hackers, newbies, and anyone interested in creating interactive objects or environments. Arduino can interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet, and even your smart-phone or your TV! This flexibility combined with the fact that the Arduino software is free, the hardware boards are pretty cheap, and both the software and hardware are easy to learn has led to a large community of users who have contributed code and released instructions for a **huge** variety of Arduino-based projects

There are many varieties of Arduino boards ([explained on the next page](https://learn.sparkfun.com/tutorials/what-is-an-arduino/the-arduino-family)) that can be used for different purposes. Some boards look a bit different from the one below, but most Arduinos have the majority of these components in common:

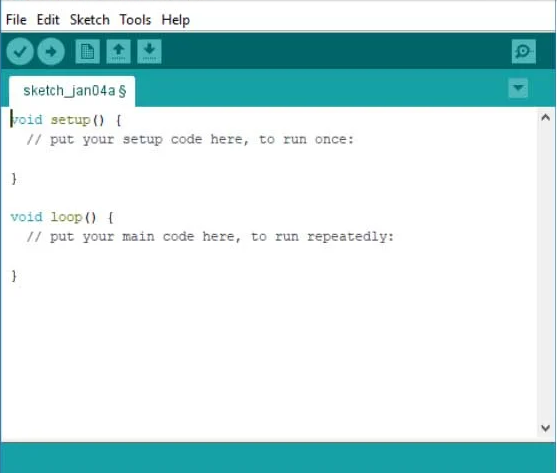


**Fig 5.3 Arduino IDE Diagram**

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor

The Arduino IDE is incredibly minimalistic, yet it provides a near-complete environment for most Arduino-based projects. The top menu bar has the standard options, including “File” (new, load save, etc.), “Edit” (font, copy, paste, etc.), “Sketch” (for compiling and programming), “Tools” (useful options for testing projects), and “Help”. The middle section of the IDE is a simple text editor that where you can enter the program code. The bottom section of the IDE is dedicated to an output window that is used to see the status of the compilation, how much memory has been used, any errors that were found in the program, and various other useful messages.

Projects made using the Arduino are called sketches, and such sketches are usually written in a cut-down version of C++ (a number of C++ features are not included). Because programming a microcontroller is somewhat different from programming a computer, there are a number of device-specific libraries (e.g., changing pin modes, output data on pins, reading analog values, and timers). This sometimes confuses users who think Arduino is programmed in an “Arduino language.” However, the Arduino is, in fact, programmed in C++. It just uses unique libraries for the device.



**Fig 5.4 Arduino IDE Software diagram**

The Arduino IDE supports the languages [C](https://en.wikipedia.org/wiki/C_(programming_language)" \o "C (programming language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B" \o "C++) using special rules of code structuring. The Arduino IDE supplies a [software library](https://en.wikipedia.org/wiki/Software_library" \o "Software library) from the [Wiring](https://en.wikipedia.org/wiki/Wiring_(development_platform)" \o "Wiring (development platform)) project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main() into an executable [cyclic executive](https://en.wikipedia.org/wiki/Cyclic_executive" \o "Cyclic executive) program with the [GNU toolchain](https://en.wikipedia.org/wiki/GNU_toolchain" \o "GNU toolchain), also included with the IDE distribution. The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

**CONCLUSION:**

In conclusion, the proposed Smart Parking Management System effectively leverages IoT technology to enhance urban parking efficiency. By integrating ultrasonic and IR sensors with real-time data transmission, the system provides accurate monitoring of parking slot availability. This not only reduces congestion and the time drivers spend searching for parking but also promotes energy conservation and smarter traffic management. Its scalability, cost-effectiveness, and user-friendly dashboard make it a practical solution for modern cities aiming to upgrade their infrastructure with intelligent, connected systems.