

CROP PROTECTION USING IOT

Submitted in partial fulfilment of the requirements for the award of

Bachelor of Engineering Degree in

Electronics and Communication Engineering

By

VEERAMALLU SIVANAGA CHANDU (38130234)

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

MARCH 2022



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **V. SHIVANAGA CHANDU** (Reg. No. 38130234) who carried out the project entitled "**CROP PROTECTION USING IOT**" under our supervision from December 2021 to March 2022.

Internal Guide
(Dr. A.ANNE FRANK JOE)

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

Head of the Department

Submitted for Viva voce Examination held on _____

Name:

Signature:

Internal Examiner

External Examiner

DECLARATION

I **V.SHIVA NAGA CHANDU** (38130234) hereby declare that the Project Report entitled "**CROP PROTECTION USING IOT**" done under the guidance of **Dr.A.ANNE FRANK JOE** at Sathyabama Institute of Science and Technology, Chennai- 600119 is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

1.

SIGNATURE OF CANDIDATE

DATE :

PLACE :

ACKNOWLEDGEMENT

We are pleased to acknowledge our 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 thanks to **Dr.N.M.NANDITHA, Dean, School of Electrical and Electronics Engineering** and **Dr.T.RAVI, Head of the Department, Dept. of Electrical and Electronics Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would also like to express our sincere and deep sense of gratitude to our internal guide **Dr.A.ANNE FRANK JOE** for her valuable guidance and suggestions during the progressive reviews for our project.

ABSTRACT:

Agriculture assumes a significant job for advancement in nourishment creation and crop protection in India. Here, agriculture relies upon disproportionate rain which thereby affects India's agriculture.

There arises a need for effective irrigation for the agricultural production. The control over how much water is to be supplied and when it is to be applied determines the uniformity which is key to maximizing the irrigation efforts.

The proper irrigation management takes careful consideration and vigilant observations. It has many benefits. Keen water irrigation and protection system framework is in this way accepted to be a significant arrangement.

The paper along these lines presents an effective water system framework that advances the accessible water in the water supply and in this manner giving an effective and powerful mechanism for the irrigation purposes.

Irrigation framework would automatically begin/stop water siphons, on the agricultural site depending upon the dampness content obtained by the moisture sensor as soon as it senses the level of water in the reservoir. The deliberate sensor estimates are sent to the Arduino Uno microcontroller for arranging the controlled calculation. The protection is done through the voice detection and movement detection methods to enable high frequency sound, hence protecting the crops from insects, pests and small animals.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE
	Introduction	8
	Literature survey	9
13	Existing and proposed system	
14	Block diagram	
1	EMBEDDED SYSTEMS	
14		
14	1.1 Overview of embedded system	
16	1.1.2 block diagram of embedded system	
16	1.1.3 characteristics of embedded system	
2	IOT	
18		
19	2.1 Motivation	
19	2.1.1 IoT definition	

20	2.2	Challenges
21	2.3	IoT based middleware
22	2.3.1	Interface protocols
23	2.3.2	Device abstraction
24	2.3.3	Context detection & Management
24	2.3.4	Application abstraction
25	2.4	conclusion
3	IR SENSOR	26
3.1	IR Sensor Circuit	27
3.2	IR Imaging Devices	28
3.3	Flame Sensor	30
3.4	Schematic Diagram	31
4	SOUND SENSOR	32
4.1	Sound Sensor Pin Configuration	33

LIST OF FIGURES

FIGURE NO		TITLE
	PAGE NO	
1.1	Block diagram of embedded system	16
2.1	Layer architecture of system	20
2.2	Functional components for IoT-based systems	22
3.1	IR sensor circuit diagram	27
3.2	IR imaging device	29
3.3	flame sensor	30
3.4	Schematic diagram of flame sensor	31
4.1	Sound sensor configuration	33
5.1	UBIDOTS data hierarchy	50

CHAPTER 1

Crop protection system using IOT

Introduction:

Agriculture assumes a significant job for advancement in nourishment creation and crop protection in India. Here, agriculture relies upon disproportionate rain which thereby affects India's agriculture.

There arises a need for effective irrigation for the agricultural production. The control over how much water is to be supplied and when it is to be applied determines the uniformity which is key to maximizing the irrigation efforts.

The proper irrigation management takes careful consideration and vigilant observations. It has many benefits. Keen water irrigation and protection system framework is in this way accepted to be a significant arrangement.

The paper along these lines presents an effective water system framework that advances the accessible water in the water supply and in this manner giving an effective and powerful mechanism for the irrigation purposes.

Irrigation framework would automatically begin/stop water siphons, on the agricultural site depending upon the dampness content obtained by the moisture sensor as soon as it senses the level of water in the reservoir. The deliberate sensor estimates are sent to the Arduino Uno microcontroller for arranging the controlled calculation. The protection is done through the voice detection and movement detection methods to enable high frequency sound, hence protecting the crops from insects, pests and small animals.

CHAPTER 2

LITERATURE SURVEY:

Agriculture is the unquestionably the largest livelihood provider in India. With rising population, there is a need for increased agricultural production. In order to support greater production in farms, the requirement of the amount of fresh water used in irrigation also rises. Currently, agriculture accounts 83% of the total water consumption in India.

Unplanned use of water inadvertently results in wastage of water. This suggests that there is an urgent need to develop systems that prevent water wastage without imposing pressure on farmers. Over the past 15 years, farmers started using

computers and software systems to organize their financial data and keep track of their transactions with third parties and also monitor their crops more effectively.

In the Internet era, where information plays a key role in people's lives, agriculture is rapidly becoming a very data intensive industry where farmers need to collect and evaluate a huge amount of information from a diverse number of devices (eg. sensors, farming machinery etc.) in order to become more efficient in production and communicating appropriate information .

Generally most of the irrigation systems are manually operated one. These traditional techniques are being replaced with semi-automated and automated techniques suggested an automated concept of irrigation to use the water efficiently and effectively Automated Drip Irrigation system is implemented either based on the soil humidity or based on the user input via SMS commanding systems.

Former method is an isolated irrigation system where the farmer doesn't updated with the irrigation status and later lags in smart utilization of water due to user command without considering the condition of soil.

From that ever growing requirement of the population, modern techniques are introduced to control the system. To give proper attention to the land located far away from the human settlement, supervisory automatic control systems like multi-terminal control systems are used since in many processes, factors like soil, salinity, irrigation, temperature, light intensity, etc. needs repeated tasks and have to work in abnormal environmental conditions of the soil and to overcome the flaws in the existing system here we are irrigating the land based on the soil humidity and at the same time the status of the irrigation is updated wirelessly to Server via serial Communication.

The automated irrigation system with IOT is feasible and cost effective for optimizing water resources for agricultural production. Using the automated irrigation system we can prove that the use of water can be reduced for different agricultural usage. The irrigation system provide only required amount of water to crop field.

This automated irrigation system with IOT allows it to be scaled up for larger open fields. An automated irrigation system was developed to minimize water use for agricultural crops. The system has a distributed wireless networks of soil moisture and Temperature sensor placed in the root zone of the plants and water level sensor

is placed in tank for monitoring the water level in tank. In addition a gateway unit handles sensor information, triggers actuators, and transmits data to a web application.

An algorithm was developed with threshold value of temperature, soil moisture and water level that was programmed into a micro-controller based on that motor is operated with the help of Arduino node mcu.

In A Remote Measurement and Control System for Greenhouse Based on GSM-SMS the proposed system introduced a GSM-SMS remote measurement and control system for greenhouse based on PC-based database system connected with base station. Base station is developed by using a microcontroller, GSM module, sensors and actuators. In practical operation, the central station receives and sends messages through GSM module. Criterion value of parameters to be measured in every base station is set by central station, and then in base stations parameters including the air temperature, the air humidity. Indu et al. (2013) mainly focuses on reviews in the field of remote monitoring and control, the technology used and their potential advantages. The paper proposes an innovative GSM/Bluetooth based remote controlled embedded system for irrigation. The system sets the irrigation time depending on the temperature and humidity reading from sensors and type of crop and can automatically irrigate the field when unattended. Information is exchanged between far end and designed system via SMS on GSM network. A Bluetooth module is also interfaced with the main microcontroller chip which eliminates the SMS charges when the user is within the limited range of few meters to the designated system. The system informs users about many conditions like status of electricity, dry running motor, increased temperature, water content in soil and smoke via SMS on GSM network or by Bluetooth. In, Suresh et al. (2014) mentioned about using automatic microcontroller based rain gun irrigation system in which the irrigation will take place only when there will be intense requirement of water that save a large quantity of water. These systems bring a change to management of field resource where they developed a software stack called Android is used for devices that include an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. Mobile phones have

almost become an integral part of us serving multiple needs of humans. This application makes use of the GPRS feature of mobile phone as a solution for irrigation control system. These system covered lower range of agriculture land and not economically affordable. In IOT SMS alarm system based on SIM900A, an IOT alarm system based on SIM900A module of SIMCOM Company was designed for greenhouse. The system can gather environmental parameters such as air temperature and air humidity. Meanwhile, with the use of AT command, this system can also realize SMS automatic sending and receiving, environmental parameters overrun alarm and insufficient balance alarm. Through the system setting, the alarm message can be sent to the user-specified mobile phone automatically no matter what the users' location is. This system as a typical application of IOT in the agriculture has got some satisfactory results in the actual operation.

IOT BASED AUTOMATIC DRIP IRRIGATION SYSTEM Ameya Bhale, Suryakant Sawant, Surya Durbha, J.Adinarayana

Irrigation is heart of agriculture, and is used to assist growing crops in the fields during the inadequate rainfall period. Irrigation is one of the areas in agriculture domain where Information, communication and dissemination technologies (ICDTs) can be employed to open up new exciting directions for research and business. Drip irrigation in particular has the potential to change the way the farms are irrigated if employed with ICDTs. For this purpose design and development of IOT middleware for remote, anytime monitoring and management of smart drip irrigation system is proposed. The system calculates the crop water requirement for the farm based on soil moisture, humidity and other sensor values. These values are used to decide and control the amount of water required to supply through valves and then to drippers. The smart automatic drip irrigation system based on Internet of Things is a new mode to ensure smart farming practices for irrigation purpose. Drip irrigation is an irrigation method that saves water and fertilizer by allowing water to drip slowly to the roots of plants, either onto the soil surface or directly onto the root zone through a network of valves, pipes, tubing and emitters. The Internet of Things (IOT) paradigm aims to bring intelligent interconnection of objects in the physical world through information gathering and dissemination devices such as sensors connected to internet using network protocol and information systems. Drip irrigation by automation helps the farmers to apply right amount of water at right time and place

regardless of availability of labour. IOT technology has diverse applications for connecting devices and making smart decision that have societal applications. Implementing drip irrigation through IOT technology has a wide potential to improve the way farms are currently irrigated. Making use of open hardware system for implementing smart drip irrigation enables changes in the system easily if needed. An IOT prototype testbed has been developed to demonstrate precision irrigation.

Arduino Based Smart Drip Irrigation System Using Internet of Things G. Parameswaran¹, K.Sivaprasath²

Nowadays water scarcity is a big concern for farming. This project helps the farmers to irrigate the farmland in an efficient manner with automated irrigation system based on soil humidity. Humidity sensor is used to find the soil humidity and based on this microcontroller drives the solenoid valve. Irrigation status is updated to the server or local host using Personal Computer. Java platform is used here for getting information via serial communication from microcontroller and to update in the server. In addition for better cropping system, fertilizers required for the crops, best crops to cultivate for the particular climatic and soil conditions are updated to server at regular basis by monitoring soil PH level, Temperature level of the field area etc., By using PC host, crop is continuously monitored. Also LCD is used to display the PH, temperature and moisture level. This will improve the cultivation method and leads to better productivity.

IOT based Smart Irrigation System Srishti Rawal

Automation of farm activities can transform agricultural domain from being manual and static to intelligent and dynamic leading to higher production with lesser human supervision. This paper proposes an automated irrigation system which monitors and maintains the desired soil moisture content via automatic watering. Microcontroller ATMEGA328P on arduino Uno platform is used to implement the control unit. The setup uses soil moisture sensors which measure the exact moisture level in soil. This value enables the system to use appropriate quantity of water which avoids over/under irrigation. IOT is used to keep the farmers updated about the status of sprinklers. Information from the sensors is regularly updated on a webpage using GSM-GPRS SIM900A modem through which a farmer can check whether the water sprinklers are ON/OFF at any given time. Also, the sensor readings are transmitted to a Thing speak channel to generate graphs for analysis.

Existing system:

The already existing we have to Monitor each and every process and may lead to some crop is going to rotting and we does not know why the crop is rotting some small mistake that make lot of economy problems there are major disadvantage in the existing system.

Proposed system:

- The system described in the paper is capable to protect the crops with a two-mode protection system for day and night along with automated irrigation with IoT monitoring.
- The movement detection system and the sound detection system ensure the crop protection with the help of high frequency sound emitted on the algorithm basis.
- In the night, along with the movement and sound detection, a lower frequency sound emitter for small insects is activated to protect the crops from pests and insects which come out in the night.

Block diagram:

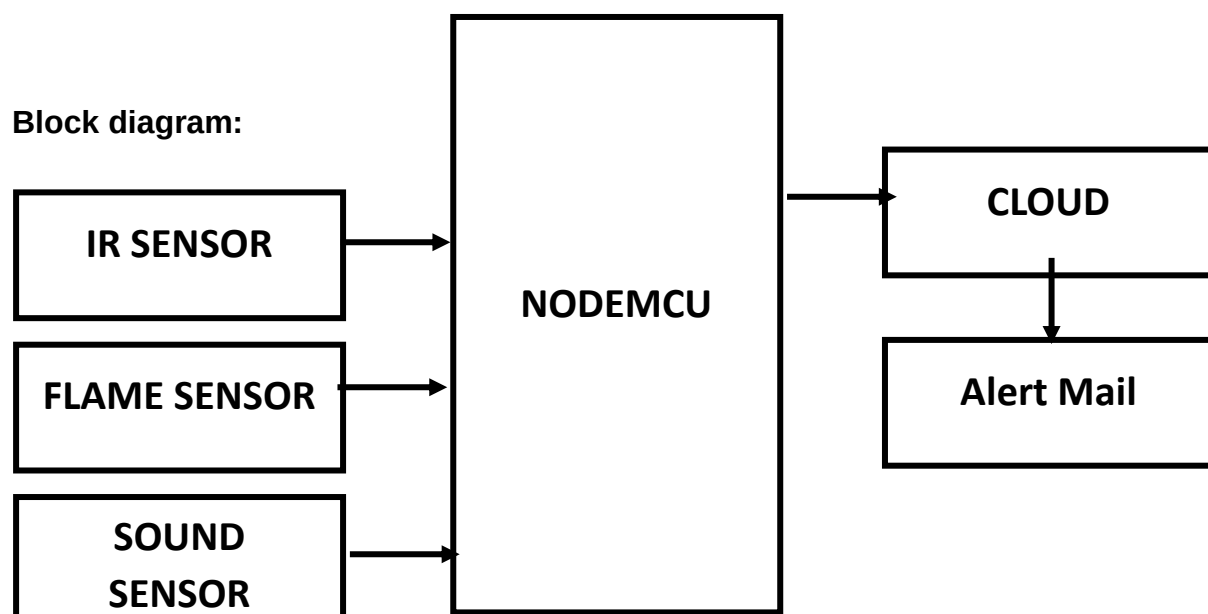


Fig. no:2.1

1. EMBEDDED SYSTEMS

1.1.1 Overview of embedded systems

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems have become very important today as they control many of the common devices we use.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

Embedded systems provide several functions

Monitor the environment; embedded systems read data from input sensors. This data is then processed and the results displayed in some format to a user or users

Control the environment; embedded systems generate and transmit commands for actuators.

Transform the information; embedded systems transform the data collected in some meaningful way, such as data compression/decompression

Although interaction with the external world via sensors and actuators is an important aspect of embedded systems, these systems also provide functionality specific to their applications. Embedded systems typically execute applications such as control laws, finite state machines, and signal processing algorithms. These systems must also detect and react to faults in both the internal computing environment as well as the surrounding electromechanical systems.

There are many categories of embedded systems, from communication devices to home appliances to control systems. Examples include;

Communication devices

e.g.: modems, cellular phones

Home Appliances

e.g.: CD player, VCR, microwave oven

Control Systems

e.g.: Automobile anti-lock braking systems, robotics, and satellite control

1.1.2 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 . A typical embedded system is shown in Fig 1.1

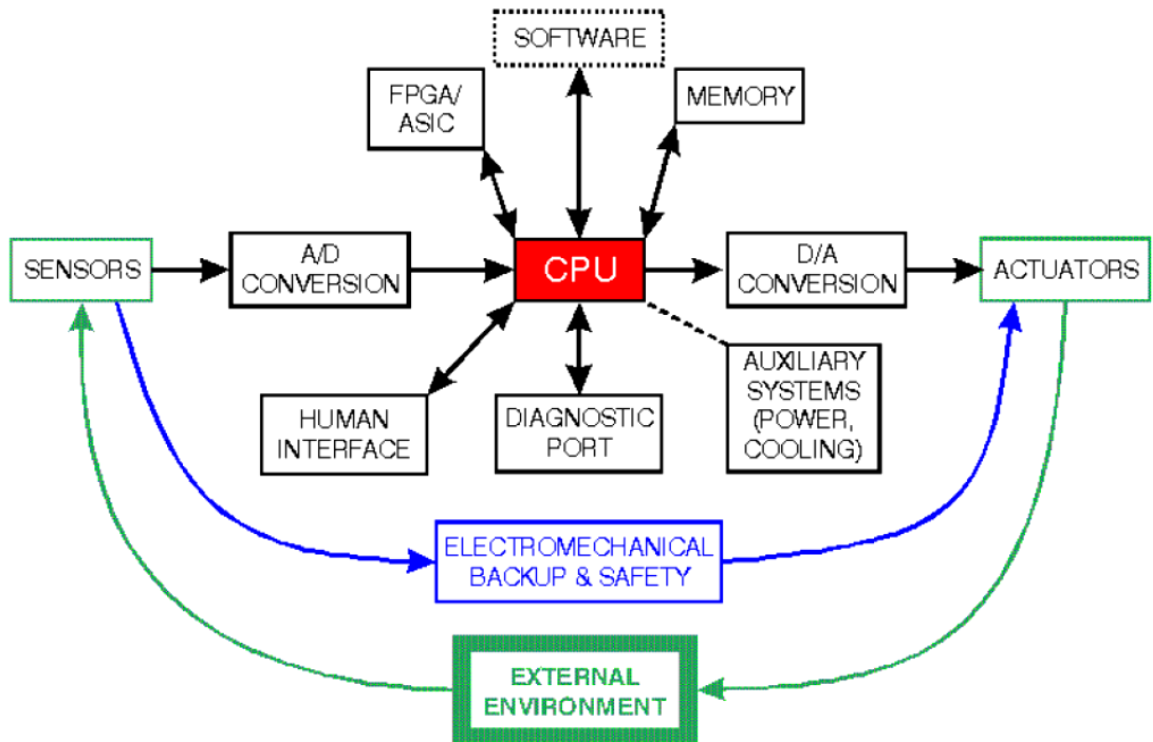


Fig 2.1.1 Block diagram of a typical embedded system

1.1.3 Characteristics of embedded systems

Embedded systems are characterized by a unique set of characteristics. Each of these characteristics imposed a specific set of design constraints on embedded systems designers. The challenge to designing embedded systems is to conform to the specific set of constraints for the application.

Application Specific Systems:

Embedded systems are not general-purpose computers. Embedded system designs are optimized for a specific application. Many of the job characteristics are known before the hardware is designed. This allows the designer to focus on the specific design constraints of a well-defined application. As such, there is limited user reprogrammability. Some embedded systems, however, require the flexibility of reprogrammability. Programmable DSPs are common for such applications.

Reactive Systems

As mentioned earlier, a typical embedded systems model responds to the environment via sensors and control the environment using actuators. This requires embedded systems to run at the speed of the environment. This characteristic of embedded system is called “reactive”. Reactive computation means that the system (primarily the software component) executes in response to external events. External events can be either periodic or aperiodic. Periodic events make it easier to schedule processing to guarantee performance. Aperiodic events are harder to schedule. The maximum event arrival rate must be estimated in order to accommodate worst case situations. Most embedded systems have a significant reactive component. One of the biggest challenges for embedded system designers is performing an accurate worst case design analysis on systems with statistical performance characteristics (e.g., cache memory on a DSP or other embedded processor). Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. Systems with this requirement must often design to worst case performance. But accurately predicting the worst case may be difficult on complicated architectures. This often leads to overly pessimistic estimates erring on the side of caution. Many embedded systems have a significant requirement for real time operation in order to meet external I/O and control stability requirements. Many real-time systems are also reactive systems.

Distributed Systems

A common characteristic of an embedded system is one that consists of communicating processes executing on several CPUs or ASICs which are connected by communication links. The reason for this is economy. Economical 4 8-bit microcontrollers may be cheaper than a 32-bit processors. Even after adding the cost of the communication links, this approach may be preferable. In this approach, multiple processors are usually required to handle multiple time-critical tasks. Devices under control of embedded systems may also be physically distributed.

Heterogeneous Architectures

Embedded systems often are composed of heterogeneous architectures. They may contain different processors in the same system solution. They may also be mixed signal systems. The combination of I/O interfaces, local and remote memories, and sensors and actuators makes embedded system design truly unique. Embedded

systems also have tight design constraints, and heterogeneity provides better design flexibility.

2. chapter

IOT:

The structure of this chapter is the following: Section 1.1 defines the motivation for this work. Section 1.2 shows the research objectives and questions. Section 1.3 presents the steps required to achieve our objectives. Section 1.4 shows the structure of the report.

1.1 Background

The Future Internet goal is to provide an infrastructure to have an immediate access to information about the physical world and its objects. Physical objects can be applicable to different application domains, such as e-health, warehouse management, etc. Each application domain may have different types of physical devices. Each physical device can have its own specifications, which is required to use in order to interact with it. To achieve the future Internet goal, a layered vision is required that can facilitate data access. Internet of Things (IoT) is a vision that aims to integrate the virtual world of information to the real world of devices through a layered architecture.

The term „Internet of Things“ consists of two words, namely *Internet* and *Things*. *Internet* refers to the global network infrastructure with scalable, configurable capabilities based on interoperable and standard communication protocols. *Things* are physical objects or devices, or virtual objects, devices or information, which have identities, physical attributes and virtual personalities, and use intelligent interfaces. For instance, a virtual object can represent an abstract unit of sensor nodes that contains metadata to identify and discover its corresponding sensor nodes. Therefore, IoT refers to the *things* that can provide information from the physical environment through the Internet.

Middleware is as an interface between the hardware layer and the application layer, which is responsible for interacting with devices and information management. The

role of a middleware is to present a unified programming model to interact with devices. A middleware is in charge of masking the heterogeneity and distribution problems that we face when interacting with devices

2.1 Motivation

IoT-based system is in charge of providing knowledge from an environment to a non-expert user. IoT-based system can be used in different environments, so it needs to be able to address many heterogeneous devices. Thus, a major concern within developing an IoT-based system is how to handle the interaction with the heterogeneous devices for non-expert users. This concern can be addressed by a middleware layer between devices and non-expert users. This layer is responsible to hide the diversity of devices from the user perspective, and provides access transparency to the devices for the end users.

The idea of creating abstractions of devices been addressed in the literature. The middleware we found in the literature can provide satisfaction by facilitating the interaction with devices, but they do not support low-level device configuration.

2.1.1 IoT definition

In this section, we explain some of the IoT definitions. Also, we explain the layered architecture for IoT.

Internet of Things (IoT) has increasingly gained attention in industry to interact with different types of devices. IoT can have influence on industry and society by integrating physical devices into information networks. IoT impacts can be on different perspectives, namely for private and business users. From the perspective of a private user, IoT has effect on both working and personal fields, such as smart homes and offices, e-health and assisted living. From the aspect of a business user the impacts would be in fields such as automation and industrial manufacturing, logistics, business process management, intelligent transportation of people and goods.

IoT integrates physical things into information networks. IoT covers the overall infrastructure, including software, hardware and services, which is used to support these information networks. The integrated physical things can exchange data about the physical properties and information that they sense in their environment. To identify devices, we can use identification technologies like for example RFID, which

allow each device be uniquely identified.

International Telecommunication Union (ITU)¹ defines IoT as “A *global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies*”.

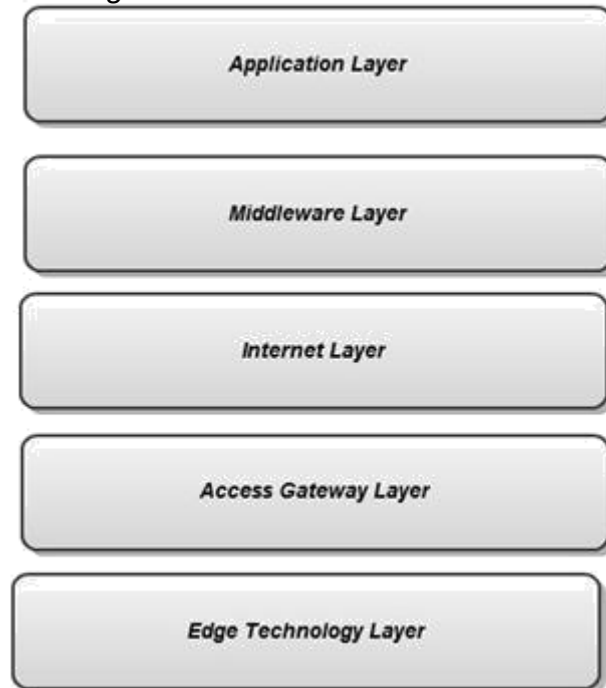


Figure 2-1 Layered architecture of the Internet of Things (IoT).

2.2 Challenges

IoT-based systems can be used for different purposes and areas, so that, we have to face different challenges. In this section, we explain some of the challenges that need to be considered in research activities:

□ Edge technology

At the hardware level, more research efforts are required to develop the technology of embedded devices, sensors, actuators and (passive and active) identification, since an IoT-based system must be able to gather sufficient information about the real world by employing a large variety of devices and

environments. Thus, there more work is still required to connect heterogeneous devices and deploy them in IoT applications, and to provide support for new devices.

- Networking technologies

In IoT, *things* are connected through different kinds of networks, i.e. mobile, wired and wireless network. These networks support bi-directional communication at different levels among the real world objects, applications and services that are employed by the IoT applications. This highly distributed structure should provide interconnection with low energy consumption, while distributed data can cause privacy issues.

- Middleware system

In IoT, we have heterogeneous devices and networks. Their heterogeneity can potentially increase with new technologies. To facilitate the use of these devices by IoT applications, we should shield their heterogeneity. Therefore, we need to develop a secure, scalable and semantically enriched service-oriented middleware to cope with the heterogeneity of devices.

- Platform services

They support a superior management of all involved devices in an integrated way, ensuring scalability, high availability, and the safe and secure execution of the requested functionality from devices.

In continue, we focus on the middleware challenges, because we are looking for the functionalities that a middleware can provide for the application layer in the IoT-based systems.

2.3 IoT_based Middleware

Bandyopadhyay, S. et. al. have studied the middleware systems that have been

applied in IoT-based systems. They classify the required functionality of middleware to manage interaction with a variety of devices in four functional components, namely (1) interface protocols, (2) device abstraction, (3) central control, context detection & management, and (4) application abstraction. In the following, we explain these components in details.

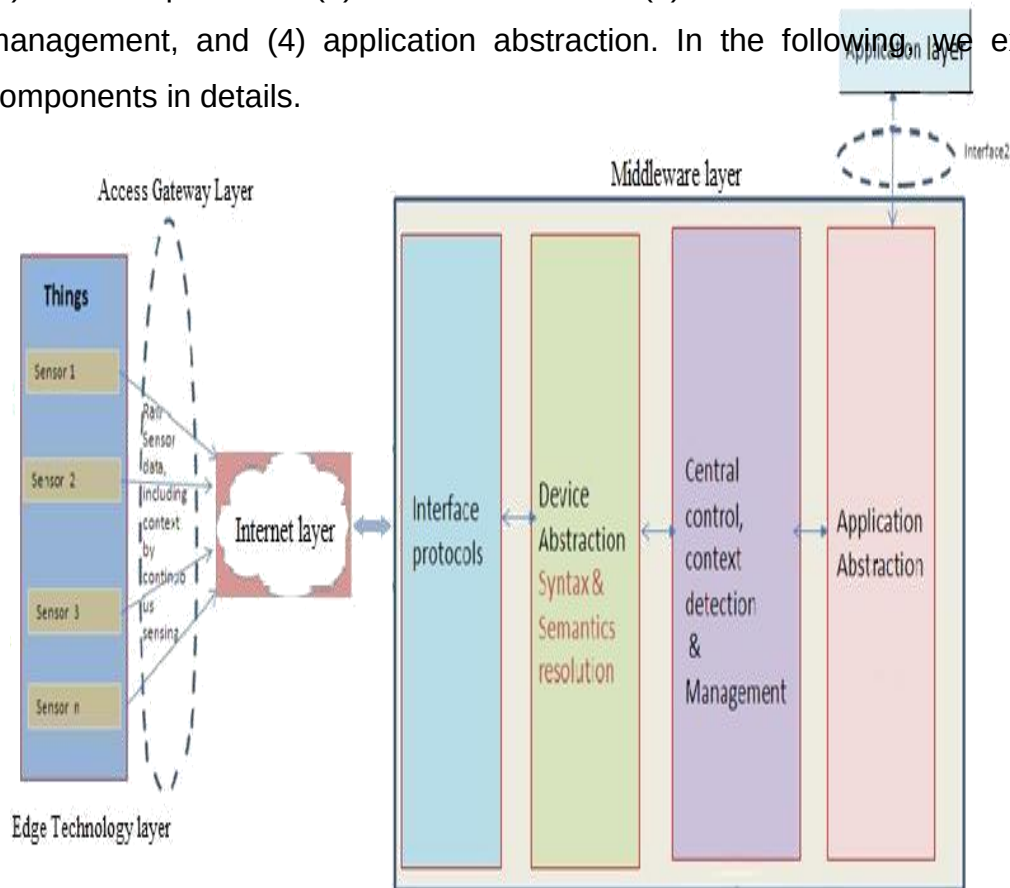


Figure 2.2 Functional components of a middleware for IoT-based systems

2.3.1 Interface protocols

This component is in charge of providing *technical interoperability*. Interoperability in the context of *Interface protocols* means: *the ability of two systems to interoperate by using the same communication protocols*. According to ETSI (European Telecommunications Standards Institute) [14] *technical interoperability* is defined as *the association of hardware or software components, systems and platforms that enable machine-to-machine communication to take place. This kind of interoperability is often centered on (communication) protocols and the infrastructure needed for those protocols to operate*[14].

The Interface Protocol component defines protocols for exchanging information among different networks that may work based on different communication protocols,

in order to allow technical interoperability. This component is responsible for handling basic connectivity in the physical and data link, network, transport, and sometimes the application layer of the TCP/IP stack.

To cope with the heterogeneity of devices, we can use a wrapper for each device to translate the protocol supported by the device to a common protocol. This wrapper can be placed either on the device side or the middleware side. If we want to have direct interaction with devices, we should place the wrapper in the middleware side. Devices usually have limited capability of computational process, so this would be a reason to implement wrapper on the middleware side. In contrast, in case of indirect interaction with devices we can develop an intermediary wrapper between the middleware and the devices. The interface protocol component is responsible to allow the middleware to support both direct and indirect interactions.

2.3.2 Device Abstraction (DA)

This component is responsible for providing an abstract format to facilitate the interaction of the application components with devices. This abstraction provides *syntactic* and *semantic* interoperability, which are defined by ETSI [14] as follows:

- *Syntactic interoperability* is associated with data formats. The messages transferred by communication protocols must have a well-defined syntax and encoding format, which can be represented by using high-level transfer syntaxes such as, HTML and XML.
- *Semantic interoperability* is usually associated with the meaning of the content of message which is understandable for human. Thus, interoperability on this level means that there is a common understanding among people on the meaning of the content (information) being exchanged among them. However, since DA does not communicate directly with human, semantic interoperability in the context of DA is in charge of providing this common understanding for applications.

Semantic interoperability relies on semantic models which tends to be domain

specific. For example, one way to provide semantic interoperability in Service Oriented (SOA) [15] based middleware is by using Devices Profile for Web Services (DPWS) [16] . In this context, each device type refers to a distinguished service type.

2.3.3 Central control, Context detection & Management (CCM)

Context characterizes the situation of an entity, which can be a place, a person or an object that is relevant to the user, applications and their interactions [1]. The CCM functional component is responsible to support context-aware computation that is a computational style that take to account the context of the entities that interact with the system. A middleware for IoT-based systems must be context-aware to work in smart environments [1]. Smart environments refer to a *physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network* [17]. Context-awareness includes two functionalities:

- 1) Context detection, which consists of collecting data from resources, and selecting the information that can have an impact on the computation.
- 2) Context processing, to use the gathered information to perform task or make a decision.

2.3.4 Application Abstraction

This functional component provides an interface for both high-level applications and end-users to interact with devices. For instance, this interface (nodeid) nearby and temperatures that are sensed by this sensor during the past 10 seconds before executing the query:

```
SELECT nodeid, temp  
FROM sensors
```

```
SAMPLE PERIOD 1s FOR 10s
```

2.4 Conclusion

In this chapter, we defined IoT and identified common IoT layers. Furthermore, we discussed a reference middleware architecture for IoT-based systems. This architecture has been proposed by Bandyopadhyay, S. et. al based on a study on the existing middleware frameworks for IoT-based systems [1].

Review of IoT-based Middleware

The term *Internet of Things (IoT)* refers to a wide set of applications and research areas such as, distributed computing and knowledge management. Much of the IoT research we found in the literature is related to the field of *pervasive computing*. World Wide Web Consortium (W3C)³ defines *pervasive computing* as a *vision about our future computing life style. In this vision, computer systems will be seamlessly integrated into our everyday life, anticipating our needs and providing relevant services and information for us in an anytime-and-anywhere fashion. Pervasive Computing is all about making the human's life easier by exploiting the available computing technologies* [20]. This motivates scientists to develop new technologies for the everyday people's lives.

Since the vision of IoT is almost similar to Pervasive computing, we reviewed the literatures that address both IoT and Pervasive systems. To select the middleware to discuss we had the following approaches:

1) We looked at typical IoT middleware such as ISMP [21], ASPIRE [22] GSN [23]. We selected HYDRA [24] to review, because it is the most popular and well-documented middleware in comparer to the mentioned middleware. Also, based on the study of Bandyopadhyay, S. et. al [3], those middleware does not support the discussed functional components in Chapter 3, while HYDRA supports them.

2) We looked at typical Pervasive system middleware such as [25], [26], [27]. We selected the following middleware to review:

- AURA [28] because this middleware focus on elaboration and manipulation of the gathered data from devices. we aim to provide ease of configuration and deployment for end user and developer. Since to meet this goal, we require to facilitate the gathered data manipulation, we reviewed AURA that was the only middleware among the reviewed paper that considered the data manipulation.

- TinyDB [19] focus on gathering data from devices. Since in an IoT-based system we need to gather data from environment through different devices, we reviewed TinyDB that is a popular middleware.
- WiseMID [29] is the only middleware among the reviewed middleware that is specific for energy saving purpose. As saving the energy of devices is important issue, we reviewed this middleware.

In this chapter, we the mentioned middleware in more detail. Also, we analyze the discussed middleware with respect to the functional components mentioned in Chapter 2.

IR (Infrared) Sensor

An infrared sensor is an electronic device that emits in order to sense some aspects of the surroundings. An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measures only infrared radiation, rather than emitting it that is called as a passive IR sensor. Usually in the infrared spectrum, all the objects radiate some form of thermal radiations. These types of radiations are invisible to our eyes that can be detected by an infrared sensor. The emitter is simply an IR LED (Light Emitting Diode) and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, the resistances and these output voltages, change in proportion to the magnitude of the IR light received.

IR Sensor Circuit Diagram and Working Principle

An infrared sensor circuit is one of the basic and popular sensor module in an electronic device. This sensor is analogous to human's visionary senses, which can be used to detect obstacles and it is one of the common applications in real time. This circuit comprises of the following components

- LM358 IC 2 IR transmitter and receiver pair
- Resistors of the range of kilo ohms.
- Variable resistors.
- LED (Light Emitting Diode).

In this project, the transmitter section includes an IR sensor, which transmits continuous IR rays to be received by an IR receiver module. An IR output terminal of the receiver varies depending upon its receiving of IR rays. Since this variation cannot be analyzed as such, therefore this output can be fed to a comparator circuit. Here an operational amplifier (op-amp) of LM 339 is used as comparator circuit. When the IR receiver does not receive a signal, the potential at the inverting input goes higher than that non-inverting input of the comparator IC (LM339). Thus the output of the comparator goes low, but the LED does not glow. When the IR receiver module receives signal to the potential at the inverting input goes low. Thus the output of the comparator (LM 339) goes high and the LED starts glowing.

3.1 IR Sensor circuit

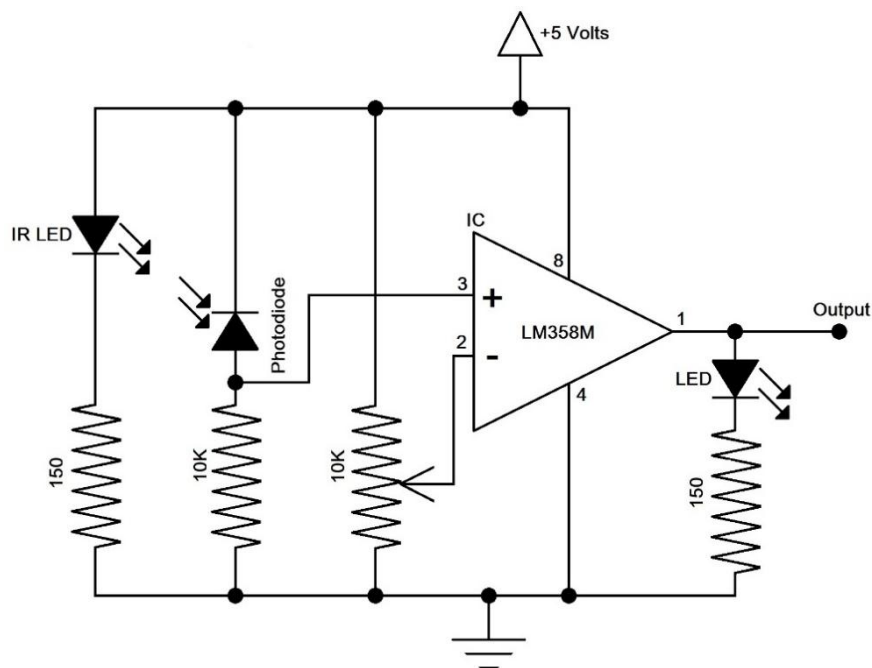


Fig3.1.1. IR Sensor Circuit diagram

Resistor R1 (100 Ω), R2 (10k Ω) and R3 (330 Ω) are used to ensure that minimum 10 mA current passes through the IR LED Devices like Photodiode and normal LEDs respectively. Resistor VR2 (preset=5k) is used to adjust the output terminals. Resistor VR1 (preset=10k) is used to set the sensitivity of the circuit Diagram. Read more about IR sensors.

Different Types of IR Sensors and Their Applications

IR sensors are classified into different types depending on the applications. The speed sensor is used for synchronizing the speed of multiple motors. The temperature sensor is used for industrial temperature control. PIR sensor is used for automatic door opening system and Ultrasonic sensor are used for distance measurement.

IR Sensor Applications

IR sensors are used in various Sensor based projects and also in various electronic devices which measures the temperature that are discussed in the below.

Radiation Thermometers

IR sensors are used in radiation thermometers to measure the temperature depend upon the temperature and the material of the object and these thermometers have some of the following features

- Measurement without direct contact with the object
- Faster response
- Easy pattern measurements

Flame Monitors

These types of devices are used for detecting the light emitted from the flames and to monitor how the flames are burning. The Light emitted from flames extend from UV to IR region types. PbS, PbSe, Two-color detector, pyro electric detector are some of the commonly employed detector used in flame monitors.

Moisture Analyzers

Moisture analyzers use wavelengths which are absorbed by the moisture in the IR region. Objects are irradiated with light having these wavelengths (1.1 μm , 1.4 μm , 1.9 μm , and 2.7 μm) and also with reference wavelengths. The Lights reflected from the objects depend upon the moisture content and is detected by analyzer to measure moisture (ratio of reflected light at these wavelengths to the reflected light at reference wavelength). In GaAs PIN photodiodes, Pbs photoconductive detectors are employed in moisture analyzer circuits.

3.2 IR Imaging Devices

IR image device is one of the major applications of IR waves, primarily by virtue of its property that is not visible. It is used for thermal imagers, night vision devices, etc.



For examples Water, rocks, soil, vegetation, an atmosphere, and human tissue all features emit IR radiation. The Thermal infrared detectors measure these radiations in IR range and map the spatial temperature distributions of the object/area on an image. Thermal imagers usually composed of a Sb (indium antimonite), Gd Hg (mercury-doped germanium), Hg Cd Te (mercury-cadmium-telluride) sensors.

An electronic detector is cooled to low temperatures using liquid helium or liquid nitrogen's. Then the Cooling the detectors ensures that the radiant energy (photons) recorded by the detectors comes from the terrain and not from the ambient temperature of objects within the scanner itself an IR imaging electronic devices.

Thus, this is all about IR sensor circuit with working and applications. These sensors are used in many sensor based electronics projects.

FIRE/FLAME SENSOR MODULE

A key aspect of fire protection is to identify a developing fire emergency in a timely manner, and to alert the building's occupants and fire emergency organizations. This is the role of fire detection and alarm systems. Depending on the anticipated fire scenario, building and use type, number and type of occupants, and criticality of contents and mission, these systems can provide several main functions. First they provide a means to identify a developing fire through either manual or automatic methods and second, they alert building occupants to a fire condition and

the need to evacuate. Another common function is the transmission of an alarm notification signal to the fire department or other emergency response organization. They may also shut down electrical, air handling equipment or special process operations, and they may be used to initiate automatic suppression systems.

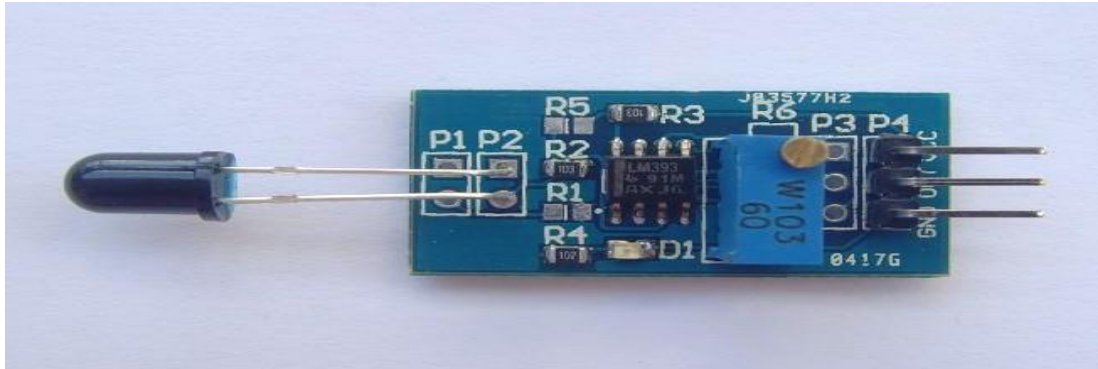


Fig 3.3 flame sensor

GENERAL DESCRIPTION:

Flame sensor is the most sensitive to ordinary light that is why its reaction is generally used as flame alarm purposes. This module can detect flame or wavelength in 760 nm to 1100 nm range of light source. Small plate output interface can and single-chip can be directly connected to the microcomputer IO port. The sensor and flame should keep a certain distance to avoid high temperature damage to the sensor.

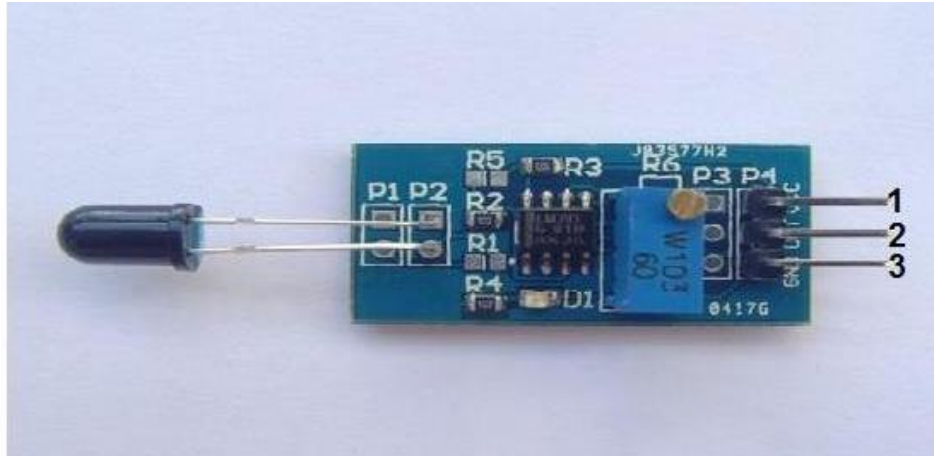
The shortest test distance is 80 cm, if the flame is bigger, test it with farther distance. The detection angle is 60 degrees so the flame spectrum is especially sensitive. The detection angle is 60 degrees so the flame spectrum is especially sensitive

SPECIFICATIONS:

- ☐ On-board LM393 voltage comparator chip and infrared sensing probe.
- ☐ Support 5V/3.3V voltage input.
- ☐ On-board signal output indication, output effective signal is high level, and the same time the indicator light up, output signal can directly connect with

microcontroller IO.

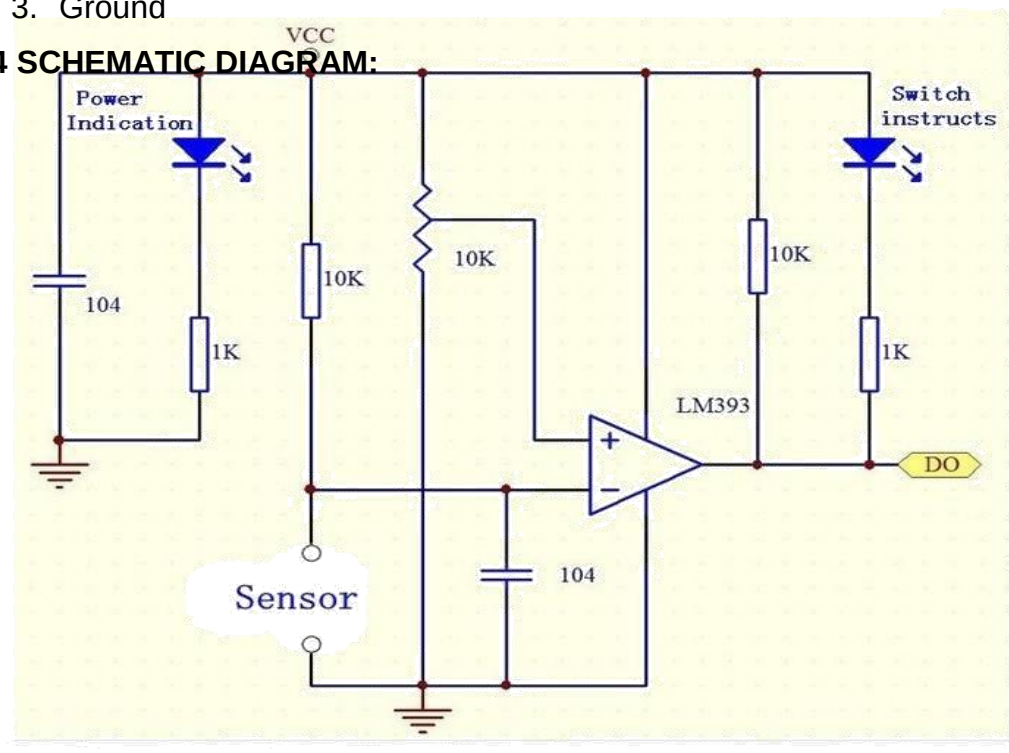
- ☐ Signal detection sensitivity can be adjusted.
- ☐ Reserved a line voltage compare circuit (P3 is leaded out).
- ☐ PCB size: 30(mm) x15(mm).



PIN CONFIGURATION:

1. VCC
2. Output
3. Ground

3.4 SCHEMATIC DIAGRAM:



Sound Sensor Working and Its Applications

Nowadays, a lot of security events are initiated due to some sort of sound which includes gunshots, aggressive behavior, breaking the glass. But cameras with inbuilt sound exposure facilities can add huge value to the security system. Because they give an alert automatically when real and potential incidents occur. Then immediately they activate quick and appropriate actions to reduce the consequences. This article discusses an overview of the sound sensor module.

What is a Sound Sensor?

The sound sensor is one type of module used to notice the sound. Generally, this module is used to detect the intensity of sound. The applications of this module mainly include switch, security, as well as monitoring. The accuracy of this sensor can be changed for the ease of usage.

This sensor employs a microphone to provide input to buffer, peak detector and an amplifier. This sensor notices a sound, & processes an o/p voltage signal to a microcontroller. After that, it executes required processing.

This sensor is capable to determine noise levels within DB's or decibels at 3 kHz 6 kHz frequencies approximately wherever the human ear is sensitive. In smartphones, there is an android application namely decibel meter used to measure the sound level.

4.1 Sound Sensor Pin Configuration

This sensor includes three pins which include the following



Fig 4.1 sound-sensor-module

- Pin1 (VCC): 3.3V DC to 5V DC
- Pin2 (GND): This is a ground pin
- Pin3 (DO): This is an output pin

Working Principle

The working principle of this sensor is related to human ears. Because human ears include a diaphragm and the main function of this diaphragm is, it uses the vibrations and changes into signals. Whereas in this sensor, it uses a microphone and the main function of this is, it uses the vibrations and changes into current otherwise voltage.

Generally, it includes a diaphragm which is designed with magnets that are twisted with metal wire. When sound signals hit the diaphragm, then magnets within the sensor vibrate & simultaneously current can be stimulated from the coils.

Features

The features of the sound sensor include the following

- These sensors are very simple to use
- It gives analog o/p signal
- Simply incorporates using logic modules on the input area

Specifications

The specifications of the sound sensor include the following

- The range of operating voltage is $3\frac{3}{5}$ V
- The operating current is 4~5 mA
- The voltage gain 26 dB ((V=6V, f=1kHz)
- The sensitivity of the microphone (1kHz) is 52 to 48 dB
- The impedance of the microphone is 2.2k Ohm
- The frequency of m microphone is 16 to 20 kHz
- The signal to noise ratio is 54 dB

Applications

The applications of the sound sensor include the following.

This sensor can be used to build various electronic projects with the help of an Arduino board. For instance, this project uses a grove sensor, which fundamentally gives your Arduino's ears. In this project, a microphone can be attached to an analog pin of the board. This can be used to notice the noise level within the nearby surroundings.

The grove sensors support platforms like Arduino, Raspberry Pi, BeagleBone Wio, and LinkIt ONE. This sensor plays an essential role while activating the light in your office or house by detecting a precise whistle or clap sound.

Some more applications of this sensor include the following.

- Security system for Office or Home
- Spy Circuit
- Home Automation

- Robotics
- Smart Phones
- Ambient sound recognition
- Audio amplifier
- Sound level recognition (not capable to obtain precise dB value)

This article discusses an overview of the sound sensor. There are different types of microphones which are known as sound sensors like dynamic, condenser, ribbon, and carbon, etc. From the above information finally, we can conclude that these sensors can be used easily, gives analog o/p signal, simply integrates by logic modules. Here is a question for you, what are the advantages of a sound sensor?

Software used: Arduino ide , Embedded C

SOFTWARE DESIGN

ARDUINO UNO AND ITS PROGRAMMING

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be communicate with software running on your computer. The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

2.1 Overview

The Arduino microcontroller is an easy to use yet powerful single board computer that has gained considerable traction in the hobby and professional market. The Arduino is open-source, which means hardware is reasonably priced and development software is free. This guide is for students in ME 2011, or students anywhere who are confronting the Arduino for the first time. For advanced Arduino users, prowl the web; there are lots of resources.

This guide covers the Arduino Uno board (Spark fun DEV-09950, \$29.95), a good choice for students and educators. With the Arduino board, you can write programs and create interface circuits to read switches and other sensors, and to control motors and lights with very little effort. Many of the pictures and drawings in this guide were taken from the documentation on the

This is what the Arduino ide looks like.



The Duemilanove board features an Atmel ATmega328 microcontroller operating at 5 V with 2 Kb of RAM, 32 Kb of flash memory for storing programs and 1 Kb of EEPROM for storing parameters. The clock speed is 16 MHz, which translates to about executing about 300,000 lines of C source code per second. The board has 14 digital I/O pins and 6 analog input pins. There is a USB connector for talking to the

host computer and a DC power jack for connecting an external 6-20 V power source, for example a 9 V battery, when running a program while not connected to the host computer. Headers are provided for interfacing to the I/O pins using 22 g solid wire or header connectors.

The Arduino programming language is a simplified version of C/C++. If you know C, programming the Arduino will be familiar. If you do not know C, no need to worry as only a few commands are needed to perform useful functions.

An important feature of the Arduino is that you can create a control program on the host PC, download it to the Arduino and it will run automatically. Remove the USB cable connection to the PC, and the program will still run from the top each time you push the reset button. Remove the battery and put the Arduino board in a closet for six months. When you reconnect the battery, the last program you stored will run. This means that you connect the board to the host PC to develop and debug your program, but once that is done, you no longer need the PC to run the program.

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Arduino Software (IDE)

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with

buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

4 Writing Sketches

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.

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension `.pde`. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the `.ino` extension on save.



Verify

Checks your code for errors compiling it.



Upload

Compiles your code and uploads it to the configured board. See [uploading](#) below for details.

Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"



New

Creates a new sketch.



Open

Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File | Sketchbook** menu instead.



Save

Saves your sketch.



Serial

Monitor

Opens the [serial monitor](#).

Additional commands are found within the five menus: **File**, **Edit**, **Sketch**, **Tools**, **Help**. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

File

- *New*

Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.

- *Open*

Allows to load a sketch file browsing through the computer drives and folders.

- *Open*

Recent

Provides a short list of the most recent sketches, ready to be opened.

- *Sketchbook*

Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.

- *Examples*

Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.

- *Close*
Closes the instance of the Arduino Software from which it is clicked.
- *Save*
Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
- *Save* *as...*
Allows to save the current sketch with a different name.
- *Page* *Setup*
It shows the Page Setup window for printing.
- *Print*
Sends the current sketch to the printer according to the settings defined in Page Setup.
- *Preferences*
Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
- *Quit*
Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

- *Undo/Redo*
Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
- *Cut*
Removes the selected text from the editor and places it into the clipboard.
- *Copy*
Duplicates the selected text in the editor and places it into the clipboard.
- *Copy* *for* *Forum*
Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
- *Copy* *as* *HTML*
Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.

- *Paste*
Puts the contents of the clipboard at the cursor position, in the editor.
- *Select* *All*
Selects and highlights the whole content of the editor.
- *Comment/Uncomment*
Puts or removes the // comment marker at the beginning of each selected line.
- *Increase/Decrease* *Indent*
Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
- *Find*
Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
- *Find* *Next*
Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
- *Find* *Previous*
Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

Sketch

- *Verify/Compile*
Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
- *Upload*
Compiles and loads the binary file onto the configured board through the configured Port.
- *Upload* *Using* *Programmer*
This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a *Tools -> Burn Bootloader* command must be executed.
- *Export* *Compiled* *Binary*
Saves a .hex file that may be kept as archive or sent to the board using other tools.

- *Show Sketch Folder*
Opens the current sketch folder.
- *Include Library*
Adds a library to your sketch by inserting `#include` statements at the start of your code. For more details, see [libraries](#) below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
- *Add File...*
Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right side of the toolbar.

Tools

- *Auto Format*
This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
- *Archive Sketch*
Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
- *Fix Encoding & Reload*
Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
- *Serial Monitor*
Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.
- *Board*
Select the board that you're using. See below for [descriptions of the various boards](#).
- *Port*
This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
- *Programmer*
For selecting a hardware programmer when programming a board or chip and not

using the onboard USB-serial connection. Normally you won't need this, but if you're [burning a bootloader](#) to a new microcontroller, you will use this.

- *Burn* *Bootloader*

The items in this menu allow you to burn a [bootloader](#) onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino or Genuino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the **Boards** menu before burning the bootloader on the target board. This command also set the right fuses.

Help

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

- *Find* *in* *Reference*

This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

Sketchbook

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the **File > Sketchbook** menu or from the **Open** button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the **Preferences** dialog.

Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

Uploading

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Port** menus. The [boards](#) are described below. On the Mac, the serial port is probably something like **/dev/tty.usbmodem241** (for an Uno or Mega2560 or Leonardo) or **/dev/tty.usbserial-1B1** (for a Duemilanove or earlier USB board), or **/dev/tty.USA19QW1b1P1.1** (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be **/dev/ttyACMx** , **/dev/ttyUSBx** or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload** item from the **Sketch** menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino **bootloader**, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its **#include** statements from the top of your code.

There is a [list of libraries](#) in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you can import a library from a zip file and use it in an open sketch. See these [instructions for installing a third-party library](#).

To write your own library, see [this tutorial](#).

Third-Party Hardware

Support for third-party hardware can be added to the **hardware** directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the **hardware** directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

For details on creating packages for third-party hardware, see the [Arduino IDE 1.5 3rd party Hardware specification](#).

Serial Monitor

This displays serial sent from the Arduino or Genuino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to **Serial.begin** in your sketch. Note that on Windows, Mac or Linux the board will reset (it will rerun your sketch) when you connect with the serial monitor.

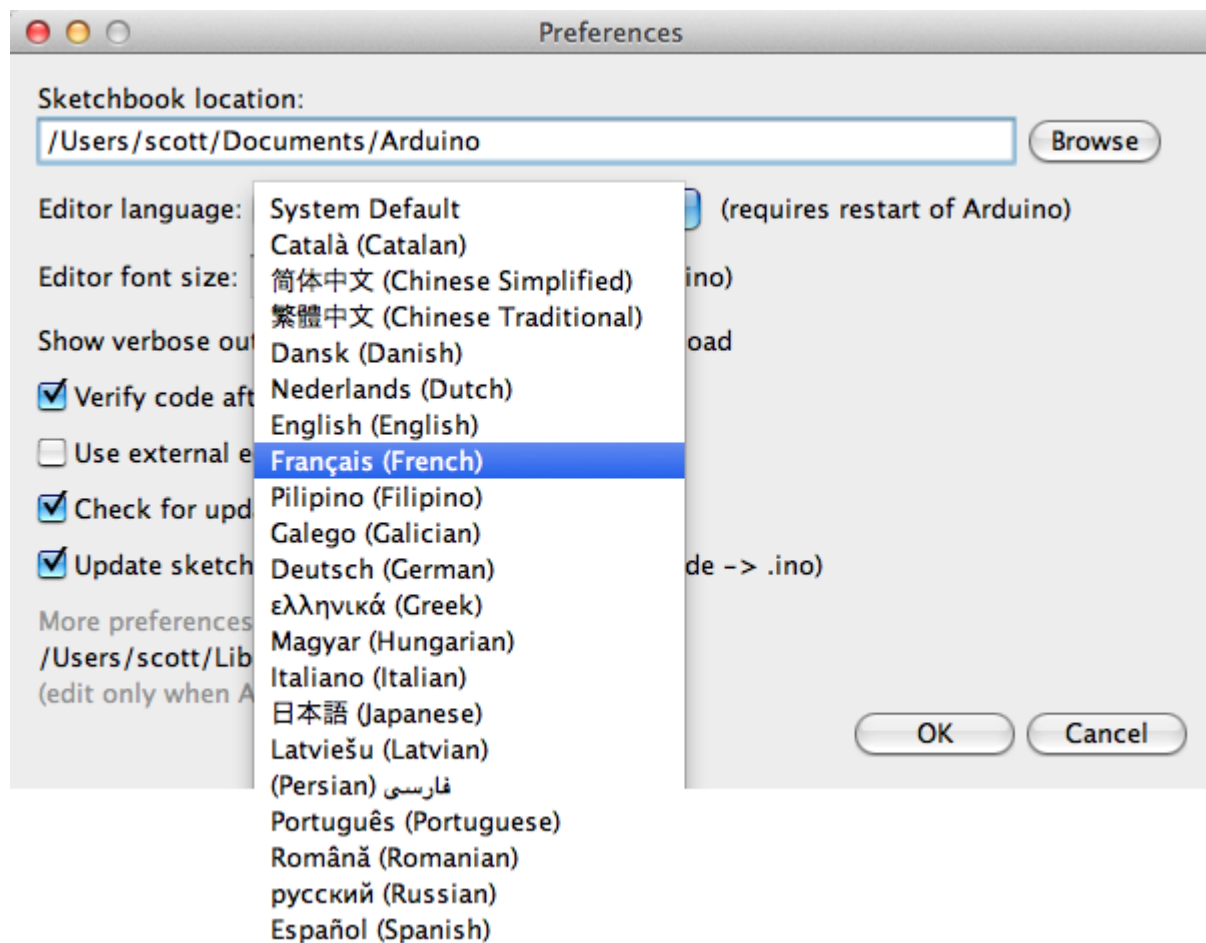
Please note that the Serial Monitor does not process control characters; if your sketch needs a complete management of the serial communication with control characters, you can use an external terminal program and connect it to the COM port assigned to your Arduino board.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the [interfacing page](#) for details).

Preferences

Some preferences can be set in the preferences dialog (found under the **Arduino** menu on the Mac, or **File** on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

Language Support



Since version 1.0.1 , the Arduino Software (IDE) has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.)

If you would like to change the language manually, start the Arduino Software (IDE) and open the **Preferences** window. Next to the **Editor Language** there is a dropdown menu of currently supported languages. Select your preferred language from the menu, and restart the software to use the selected language. If your operating system language is not supported, the Arduino Software (IDE) will default to English.

You can return the software to its default setting of selecting its language based on your operating system by selecting **System Default** from the **Editor Language** dropdown. This setting will take effect when you restart the Arduino Software (IDE). Similarly, after changing your operating system's settings, you must restart the Arduino Software (IDE) to update it to the new default language.

Boards

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader. You can find a comparison table between the various boards [here](#).

Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The [Boards Manager](#) included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

- *Arduino* *Yùn*
An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino/Genuino* *Uno*
An ATmega328P running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino* *Diecimila* *or* *Duemilanove* *w/ ATmega168*
An ATmega168 running at 16 MHz with auto-reset.
- *Arduino* *Nano* *w/ ATmega328P*
An ATmega328P running at 16 MHz with auto-reset. Has eight analog inputs.
- *Arduino/Genuino* *Mega* *2560*
An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino* *Mega*
An ATmega1280 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino* *Mega* *ADK*
An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
- *Arduino* *Leonardo*
An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino/Genuino* *Micro*
An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
- *Arduino* *Esplora*
An ATmega32u4 running at 16 MHz with auto-reset.
- *Arduino* *Mini* *w/ ATmega328P*
An ATmega328P running at 16 MHz with auto-reset, 8 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino* *Ethernet*
Equivalent to Arduino UNO with an Ethernet shield: An ATmega328P running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino* *Fio*
An ATmega328P running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328P, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino* *BT* *w/ ATmega328P*
ATmega328P running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board bluetooth module, 6 Analog In, 14 Digital I/O and 6 PWM..
- *LilyPad Arduino* *USB*
An ATmega32u4 running at 8 MHz with auto-reset, 4 Analog In, 9 Digital I/O and 4 PWM.
- *LilyPad Arduino*
An ATmega168 or ATmega132 running at 8 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino Pro or Pro Mini (5V, 16 MHz)* *w/ ATmega328P*
An ATmega328P running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328P; 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino NG or older* *w/ ATmega168*
An ATmega168 running at 16 MHz *without* auto-reset. Compilation and upload is equivalent to Arduino Diecimila or Duemilanove w/ ATmega168, but the bootloader burned has a slower timeout (and blinks the pin 13 LED three times on reset); 6 Analog In, 14 Digital I/O and 6 PWM.
- *Arduino* *Robot* *Control*
An ATmega328P running at 16 MHz with auto-reset.
- *Arduino* *Robot* *Motor*
An ATmega328P running at 16 MHz with auto-reset.
- *Arduino* *Gemma*
An ATtiny85 running at 8 MHz with auto-reset, 1 Analog In, 3 Digital I/O and 2 PWM.

For instructions on installing support for other boards, see [third-party hardware](#) above.

Last revision 2015/09/07 by SM

The text of the Arduino getting started guide is licensed under a [Creative Commons Attribution-ShareAlike 3.0 License](#). Code samples in the guide are released into the public domain.

UBIDOTS EXPLORER:

Introduction

Welcome to the Ubidots REST API hardware docs. Here you would reference how to send or retrieve data from your hardware devices using our supported communication protocols: **HTTP**, **MQTT** and **TCP/UDP**.

The purpose of this section is to help you understand what happens in the backstage when communicating with Ubidots, so you can replicate this in your firmware. For this reason, we avoid the use of examples using libraries. However, if you'd like to use one of our libraries (Python, C, and more), check out our [compatible devices](#) and [API clients](#) sections.

How Ubidots Works?

Every time a device updates a sensor value in a variable, a data-point or "**dot**" is created. Ubidots stores dots that come from your devices inside variables, and these stored dots have corresponding **timestamps**:

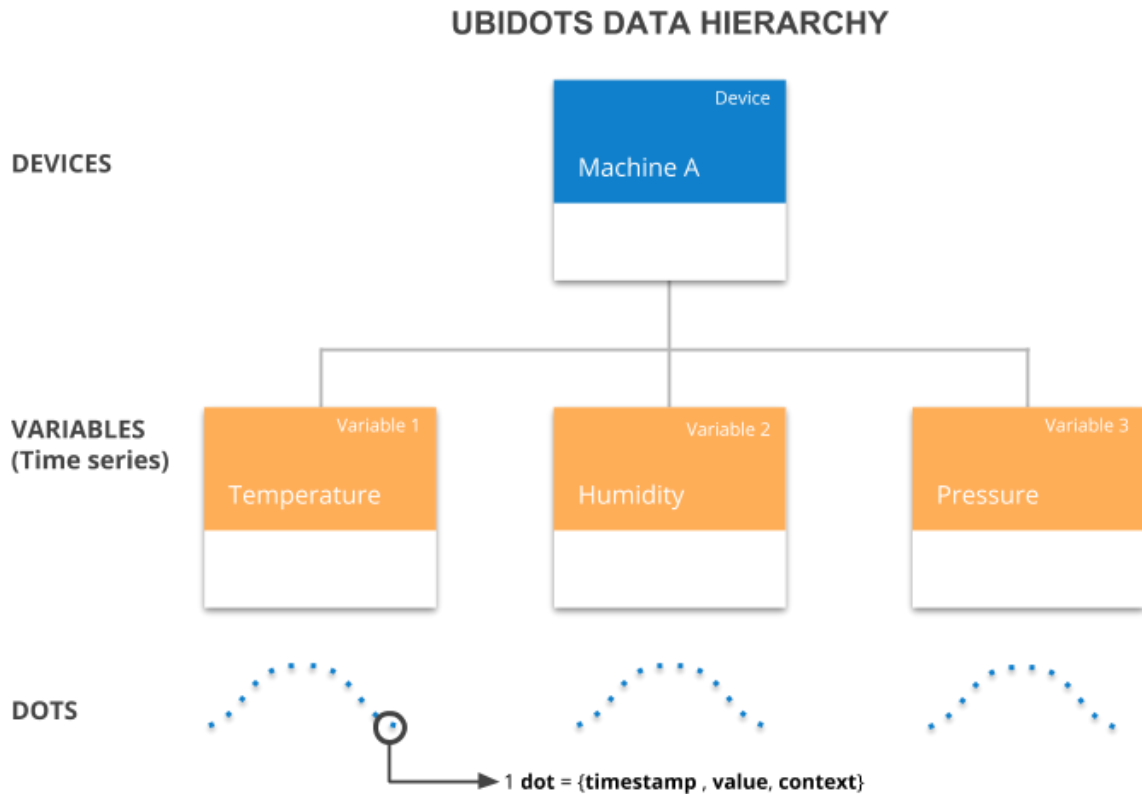


Fig 5.1 UBIDOTS data hierarchy

Each dot contains these items:

Item	Description	Mandatory
value	A numerical value. Ubidots accepts up to 16 floating-point length numbers.	YES
timestamp	Unix Epoch time, in milliseconds. If not specified, then our servers will assign one upon reception.	NO
context	An arbitrary collection of key-value pairs. Mostly used to store the latitude and longitude coordinates of GPS devices.	NO

Values

A numerical value. Ubidots accepts up to 16 floating-point length numbers.

```
{"value" : 34.87654974}
```

Timestamps

A **timestamp**, as best described [here](#), is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Therefore, the unix time stamp is merely the number of seconds between a particular date and the Unix Epoch. Please keep in mind that when you send data to Ubidots, you must set the timestamp in milliseconds; also, if you retrieve a dot's timestamp, it will be in milliseconds.

```
"timestamp" : 1537453824000
```

The above timestamp corresponds to Thursday, September 20, 2018 2:30:24 PM.

PRO-TIP: A useful tool to convert between Unix timestamps and human-readable dates is [Epoch Converter](#).

Context

Numerical values are not the only data type supported; you can also store *string* or *char* data types inside what we call **context**. The **context** is a key-value object that allows you to store not only numerical but also string values. An example use of the **context** could be:

```
"context" : {"status" : "on", "weather" : "sunny"}
```

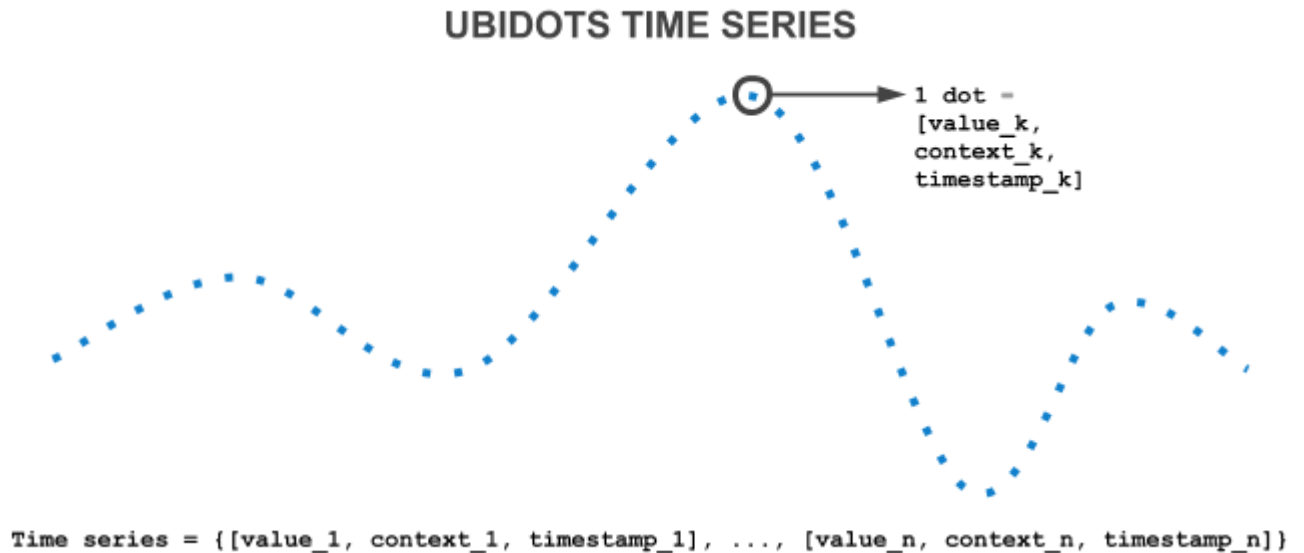
A **context** is commonly used to store the latitude and longitude coordinates of your device for GPS/tracking application use cases. All Ubidots maps uses the *lat* and *lng* keys from a dot's **context** to extract the coordinates of your device, in that way you just need to send a single dot with the coordinates values in the variable context to plot a map instead of sending separately both latitude and longitude in two different variables. Below you can find a typical **context** with coordinates values:

```
"context" : {"lat":-6.2, "lng":75.4, "weather" : "sunny"}
```

Please note that you can mix both string and numerical values in the **context**. If your application is for geo-localization purposes, make sure that the coordinates are set in decimal degrees.

Time Series

Based on the above, we can illustrate an Ubidots time series like this:



Ubidots is an agnostic platform, this means that it does not really care what hardware device you are using, as long as you're able to interact with us through at least one of these protocols:

- [HTTP](#)
- [MQTT](#)
- [TCP/UDP](#)

HTTP

HTTP is the main internet protocol for information exchange in the Internet. It also supports data encryption, which is usually called *secure HTTP* or *HTTPS*. When you use a web browser (MS Edge, Firefox, Google Chrome, etc) you are sending and receiving packages using *HTTP* or *HTTPS*. Because of this, it was not surprising to see *HTTPS* become a popular option for data exchange within IoT.

When a developer is to implement *HTTP* communication, then he should look for a *RESTful Application Programming Interface*, or *REST API*, which exposes all the endpoints required to interact with a third party application (like Ubidots). In this section, you can reference our REST API to understand how to send and retrieve data from our servers.

Note: Our goal with this section is to help you understand what's happening behind when communicating with Ubidots, enabling you to replicate it within your own firmware. Because of this, we avoid using our custom libraries in all the examples.

HTTP requests

The following methods are specified within the *HTTP* standard:

HTTP Method	Description
GET	Used to retrieve information
POST	Used to create a new element
PATCH	Used to update existing elements
DELETE	Used to delete existing elements

HTTP is a **request/response** protocol, this means that every request that you make is answered by the server. This response includes a number (response code) and a body. For example, when you make a request to retrieve a file on a webpage "(e.g. *"Get me the file 'webside.html'"*)", you build a GET request. If the request is correct, the server will typically return a 200 response code, along with the file requested (body).

Under the same logic, when a device sends a GET request to the Ubidots' server (i.e. requesting the last value of a variable), then the server sends back a response with the status of the request (response code), and a body, which would be the a value with its context and timestamp.

An *HTTP* request also needs the parameters below:

- **Host:** Specifies the server you will be making HTTP requests to.
- **Path:** This is typically the remaining portion of the URL that specifies the resource you want to consume, be it a variable or a device. For example, if an API endpoint is: `industrial.api.ubidots.com/api/v1.6/devices/my-device` then the path would be `/api/v1.6/devices/my-device`
- **Headers:** Define the operating parameters of the *HTTP* request such as authentication, Content-Type, Content-Length, etc.

- **Body/payload:** In the case of POST and PATCH requests, this is the data sent by your device to the server. GET requests typically do not have a body because they are meant to request data, not to send data.

Ubidots accepts data as *JavaScript Object Notation* or *JSON*. JSON is a typical *HTTP* data type, it is a collection of name/value pairs. In various programming languages, this is treated as an object, record, struct, dictionary, hash table, keyed list, or associative array. It is also human readable and language independent. An example of a JSON data type that Ubidots accepts can be referenced below:

```
{"temperature": {"value":10, "timestamp": 1534881387000, "context": {"machine": "1st floor"}}
```

A typical *HTTP* request to Ubidots should be set as below:

```
POST {PATH} HTTP/1.1<CR><LN>
Host: {HOST}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
```

Where:

- {PATH}: Path to the resource to consume
Example: /api/v1.6/variables/ to get user's variables information.
- {HOST}: Host URL.
Example: industrial.api.ubidots.com
- {USER_AGENT}: An optional string used to identify the type of client, be it by application type, operating system, software vendor or software version of the requesting user agent.
Examples: ESP8266/1.0, Particle/1.2
- {TOKEN}: Unique key that authorizes your device to ingest data inside your Ubidots account.
- {PAYLOAD_LENGTH}: The number of characters of your payload.
Example: The payload {"temperature": 20} will have a content-length of 19.

- {PAYLOAD}: Data to send.

Example: {"temperature": 20}

PRO TIP: The easy way of handle *HTTP* and *HTTPS* requests is by using **cURL**, a command line tool. To learn how to install cURL on Windows, MacOSX or Linux please refer to this [guide](#).

API URLs

API access can be over *HTTP* or secure *HTTP*, *HTTPS*, using the following endpoints based on your Ubidots Account type.

Security Note: We strongly advise to use *HTTPS* to make sure your data travels encrypted, avoiding the exposure of your API token and/or sensor data.

HTTP

Ubidots Account	Endpoint	Port
Educational	http://things.ubidots.com	80
Industrial	http://industrial.api.ubidots.com	80

HTTPS

Ubidots Account	Endpoint	Port
Educational	https://things.ubidots.com	443
Industrial	https://industrial.api.ubidots.com	443

Ubidots supports SSL v1.1, TLS v1.2 and v1.3. You can download our root certificates in different formats:

- [PEM file](#): A certificate chain with two root certificates from our certificate authorities (CAs).
- [DER file](#): Same as the PEM file, with an alternative encoding.

- [CRT file](#): Same as the PEM file, with a different extension. Often referred to as .crt, .cert or .cer.

NOTE: To avoid future inconveniences, we advise to **use both certificates** in your device's auth routines. While the second is the current valid certificate, we will use the first after 2021.

Fixed IPs

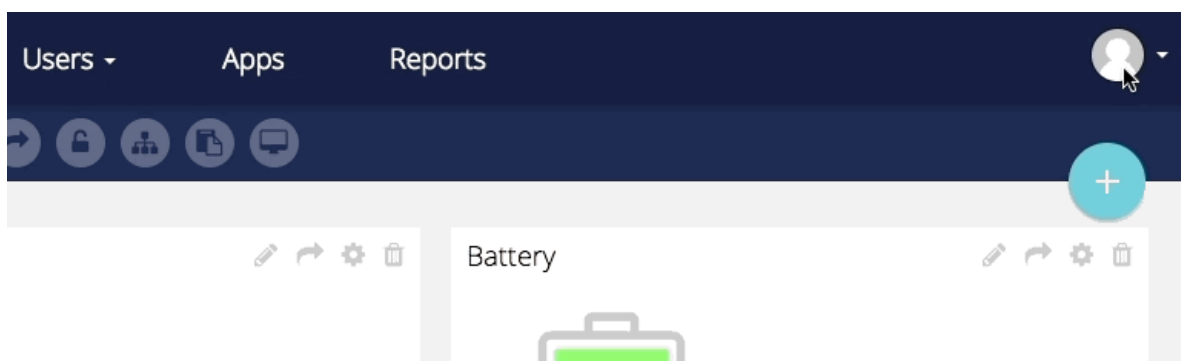
Ubidots also supports fixed IPs address for data ingestion, but we highly advise to our users to implement their firmware routines using the dns endpoints instead of fixed IPs, as these last ones may change in the future.

Ubidots Account	ipv4	ipv6	Port
Educational	50.23.124.68	2607:f0d0:2101:39::2	80/443
Industrial	169.55.61.243	2607:f0d0:3:b::2	80/443

HTTP Authentication

Every request requires a *token*. A *token* is an unique key that authorizes your device to ingest data inside your Ubidots account.

The easiest way to get yours is clicking on "**API Credentials**" under your profile tab:



Also, 6 hour valid *tokens* can be generated using your account *API-KEY*. If you desire to learn how to do it, please refer the [Ubidots REST Software API reference](#).

There are two ways to send a *token* in a request:

1. Send the *token* as parameter in the endpoint, setting it with the `?token=` expression. This is the easiest one to implement but the most unsecure method to send your authentication credentials.
2. Send the *token* as a request header. For this, use the X-Auth-Token header.

Security Note: While sending your *token* as a URL parameter can be quite comfortable in some devices, we only advise to do this for prototyping stages. For production stages, we strongly advise sending the token using the X-Auth-Token header.

```
GET
https://industrial.api.ubidots.com/api/
```

Send data

Our REST API allows you send data to the platform in two different ways:

- Send data to a **device**
- Send data to a **variable**

If you're looking for more advanced API capabilities, such as editing a device name, or listing all of your devices, please refer to our Ubidots Software REST API Reference.

Send data to a device

Request structure:

```
POST /api/v1.6/devices/{DEVICE_LABEL} HTTP/1.1<CR><LN>
Host: {Host}<CR><LN>
User-Agent: {USER_AGENT}<CR><LN>
X-Auth-Token: {TOKEN}<CR><LN>
Content-Type: application/json<CR><LN>
Content-Length: {PAYLOAD_LENGTH}<CR><LN><CR><LN>
{PAYLOAD}
<CR><LN>
```

Expected Response:

```
HTTP/1.1 200 OK<CR><LN>
Server: nginx<CR><LN>
```

```

Date: Tue, 04 Sep 2018 22:35:06 GMT<CR><LN>
Content-Type: application/json<CR><LN>
Transfer-Encoding: chunked<CR><LN>
Vary: Cookie<CR><LN>
Allow: GET, POST, HEAD, OPTIONS<CR><LN><CR><LN>
{PAYLOAD_LENGTH_IN_HEXADECIMAL}<CR><LN>
{"{VARIABLE_LABEL}": [{"status_code": 201}]}<CR><LN>
0<CR><LN>

```

The easiest way to **send values to Ubidots** is by specifying your Device label in the request *path* and making a POST request to it.

Replace the {DEVICE_LABEL} with the unique label of the desired device to send data to. If the device does not exist in your Ubidots account, it will be automatically created. Device labels can contain any alphanumerical character, without blank spaces, i.e *weather-station*, *my-home*, *esp8266*.

To set the variables values, just create a *JSON* data payload as below

Nº Variables	JSON structure
One	{"variable-label": VALUE}
Multiple	{"variable-label-1": {"value": VALUE, "timestamp": TIMESTAMP, "context": {"key": VALUE}}, ... , "variable-label- n": {"value": VALUE, "timestamp": TIMESTAMP, "context": {"key": VALUE}}}

timestamp and *context* parameters are optional.

NOTE 1: By default, the Ubidots backend will take the latitude and longitude values inside any variable labeled as **"gps"**, **"location"**, **position** or **"coordinates"** automatically to get your device coordinates in all of the map widgets. Alternatively, you change the default location variable.

NOTE 2: All timestamps must be in milliseconds. If you set a timestamp for a variable value, it will be stored with the date represented by it. You can easily convert dates to timestamps values [here](#).

Conclusion:

The proposed authorization IoT model provides a safe environment with security measures that ensures the privacy of the communicating IoT-enabled devices and nodes and their shared data. Upon successful implementation and correct configuration, the users can be sure that their systems assets are protect.

References:

- [1] Kumar, G & Raju, Boni & Nandan,. (2020). A Review on the Smart Irrigation System.
- [2] Sahu, Chandan&Behera, Pramitee. (2015). A low-cost smart irrigation control system. 10.1109/ECS.2015.7124763.
- [3] Rani, Mercy &Thengal, A. &Hangsing, Paominhao& Ray, Lala I.P. & Rajesh, Thangaswamy&Datta, Kamalika. (2019). A Simulation Model for the Smart Irrigation System Using Arduino. 8. 7-. 10.17148/IJARCCE.2019.8202.
- [4] Kunal, Kumar & Hussain, Md & Srinivasan, N & Mayan, J.. (2019). Smart Irrigation and Tank Monitoring System. IOP Conference Series: Materials Science and Engineering. 590. 012035. 10.1088/1757- 899X/590/1/012035.
- [5] Freeda, J. & Subramanian, Malathi. (2019). IoT Based Innovation Schemes in Smart Irrigation System with Pest Control. 10.1007/978- 3- 030-32150-5_64.
- [6] Hussain, Shahadat&Nahar, Nazmun&Pathan, Refat& Pathak, Abhijit&Tutul, Amaz& Abedin, Md. Jainal. (2019). IoT based Smart System to Support Agricultural Parameters: A Case Study. Procedia Computer Science. 155. 648-653.
- [7] Karan Kansara, Vishal Zaveri, Shreyans Shah, Sandip Delwadkar, and Kaushal Jani "Sensor based Automated Irrigation System with IOT: A Technical Review", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (6) , 2015, 5331-5333 [8] Sumeet. S. Bedekar, Monoj. A. Mechkul, and Sonali. R. Deshpande "IOT based Automated Irrigation System", IJSRD - International Journal for Scientific Research& Development| Vol. 3, Issue 04, 2015 | ISSN (online): 2321-0613 .