**cCHAPTER 1**

# INTRODUCTION

### MOTIVATION

Cultural property management is entrusted with the responsibility of protecting and preserving an institution's buildings, collections, operations and occupants. Constant attention is required to minimize adverse impact due to climate, pollution, theft, vandalism, insects, mold and fire. Because of the speed and totality of the destructive forces of fire, it constitutes one of the more serious threats. Vandalized or environmentally damaged structures can be repaired and stolen objects recovered. Items destroyed by fire, however, are gone forever. An uncontrolled fire can obliterate an entire room's contents within a few minutes and completely burn out a building in a couple of hours. Hence it has become very necessary to control and cease the fire to protect the Life and costlier things. For that we purposed to design and fabricate the fire-fighting robot.

Autonomous robots can act on their own, independent of any controller. The basic idea is to program the robot to respond in a certain way to outside stimuli. The very simple bump-and-go robot is a good illustration of how this works.

This sort of robot has a sensor to detect obstacles. When you turn the robot on, it zips along in a straight line. When it finally hits an obstacle, the impact is on sensors, i.e, sansors may get damaged. Using Ultrasonic sensor and programming logic, the robot is guided to turn right and move forward again, when the robot finds an obstacle in its way. In this way, the robot changes direction any time it encounters an obstacle. Advanced robots use more elaborate versions of this same idea. Roboticists create new programs and sensor systems ,to make robots more smarter and more perceptive. Today, robots can effectively navigate in a variety of environments.

### PROJECT OVERVIEW

The project is designed to develop a fire fighting robot using Arduino uno . The robotic vehicle is loaded with water pump which is controlled by servos. An ATMega 328 microcontroller is used for the desired operation. At the transmitting end using commands are sent to the receiver to control the movement of the robot either to move forward, and

left or right etc. At the receiving end tow motors are interfaced to the microcontroller where two of them are used for the movement of the vehicle and the one to position the robot. The ultrasonic sensor adequate range with obstacle detection, while the receiver driver module used to drive DC motors via motor driver IC for necessary work. A water tank along with water pump is mounted on the robot body and its operation is carried out from the microcontroller output through appropriate command from the transmitting end. The whole operation is controlled by an ATmega 328 microcontroller. A motor driver IC is interfaced to the microcontroller through which the controller drives the motors,three ir flame sensors are fixed on robot chassis to sense the fire and to reach the destination to putoff the fire.

### COMPONENTS OVERVIEW

This system uses the following components.

#### Microcontroller

Microcontroller can be described as a computer embedded on a rather small circuit board. To describe the function of a microcontroller more precisely it is a single chip that can perform various calculations and task and send/receive signals from other devices via the available pins. Precisely what tasks and communication with the world it does, is what is governed by what instructions we give to the Microcontroller. It is this job of telling the chip what to do, is what we refer to as programming on it.

However, the microcontroller by itself, cannot accomplish much, it needs several external inputs, power, for one, a steady clock signal, for another. Also, the job of programming it has to be accomplished by an external circuit. So typically, a microcontroller is used along with a circuit which provides these things to it; this combination is called a microcontroller board. The Arduino Uno that you have received is one such microcontroller board. The actual microcontroller at its heart is the chip called **Atmega328**. The advantages that Arduino offers over other microcontroller boards are largely in terms of reliability of the circuit hardware as well as the ease of programming and using it.

#### Power Supply

7805 is a voltage regulation IC which is used to supply 5V Direct current to the microcontroller

#### Motor Driver IC

L293D is a dual H-bridge motor driver integrated circuit (IC). They are use to control they the 4 motor used in project. There are 2 motor driver IC used in the project one to control front motor and other for rear motors.

#### Computer Interface

Finally, this project uses IDE compiler for interfacing the arduino with a PC. This interface is used to setup and compile the Arduino.

**CHAPTER 2**

Department of ECE, VBIT 4



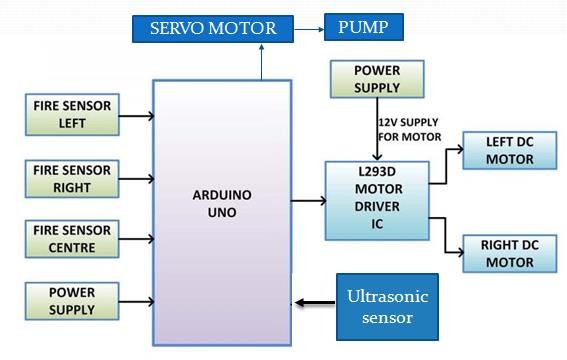
Fig 2.1: Overview of system

From Fig 2.1, there are at least five interfacing circuits, L293d driver module, Arduino-uno with Microcontroller ,flame sensors,ultrasonic sensors,servo motor and 5v pump.here arduino uno acts a heart of our project,in the above block diagram we can see that there are three flame sensors and ultrasonic sensor which acts as input interface to the microcontroller and sevomotor,pump,driver module acts a ouput interface to the

microcontroller,here the input and output interface can be indicated with the arrow lines

# PROBLEM DEFINITION

As explained in the introduction chapter, the realization of complete potential of the sensors and the wired medium in information transfer is the major issue that the following thesis of the following project deals with.



power supply

with the respective the microcontroller performs with the respective commands and delay which is programmed on arduino software.

### 2.1 CIRCUIT DIAGRAM OF THE SYSTEM

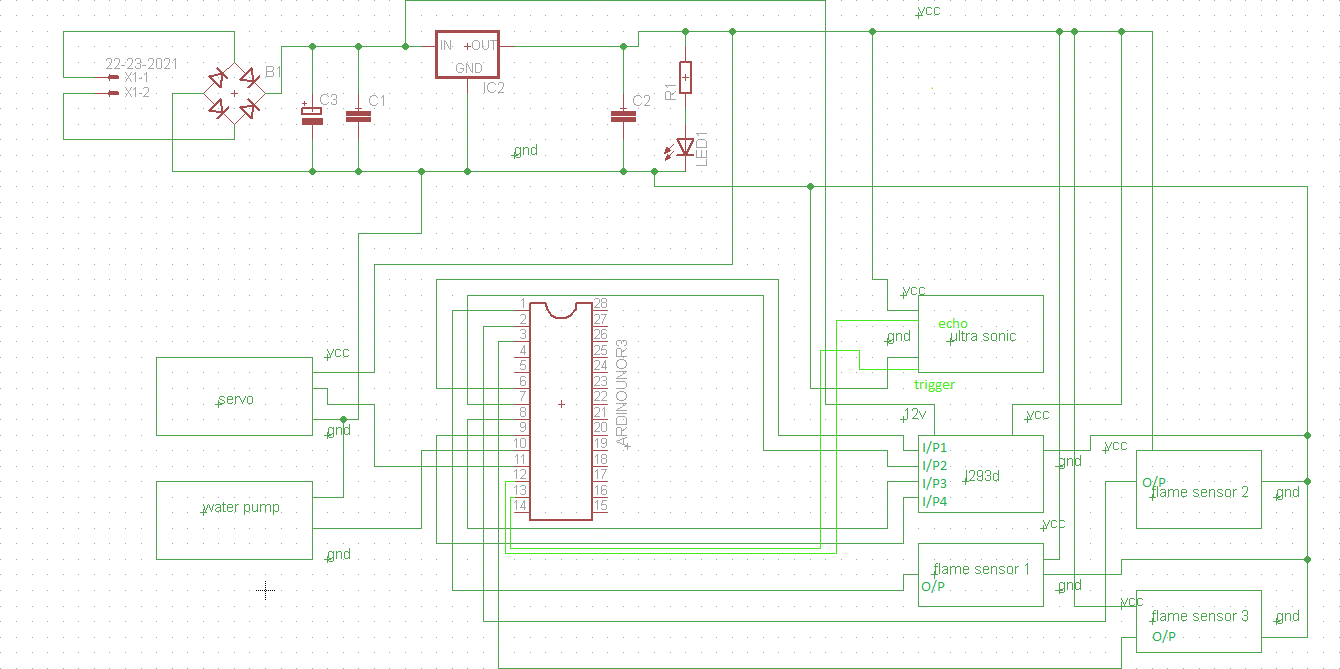


Fig 2.2: Circuit diagram of the system.

As you can see these sensors have an **IR Receiver (Photodiode)** which is used to detect the fire. How is this possible? When fire burns it emits a small amount of Infra-red light, this light will be received by the IR receiver on the sensor module. Then we use an Op- Amp to check for change in voltage across the IR Receiver, so that if a fire is detected the output pin will give 0V(LOW) and if the is no fire the output pin will be 5V(HIGH).So, we place three such sensors in three directions of the robot to sense on which direction the fire is burning. We detect the direction of the fire we can use the motors to move near the fire by driving our motors through the **L293D module**. When near a fire we have to put it out using water. Using a small container we can carry water,

a 5V pump is also placed in the container and the whole container is placed on top of a **servo motor** so that we can control the direction in which the water has to be sprayed.

**CHAPTER 3**

# SYSTEM REQUIREMENT SPECIFICATION

### HARDWARE REQUIREMENTS

#### L293D Driver module

The Motor Driver is a module for motors that allows you to control the working speed and direction of two motors simultaneously .This Motor Driver is designed and developed based on L293D IC.L293D is a 16 Pin Motor Driver IC. This is designed to provide bidirectional drive currents at voltages from 5 V to 36 V.

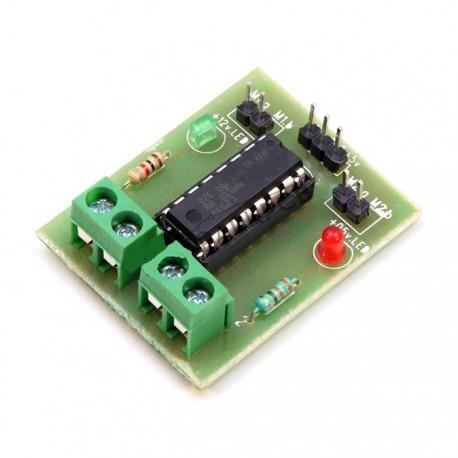


Figure 3.1: L293D motor driver module

L293D is a dual [H-bridge](http://www.engineersgarage.com/electronic-circuits/h-bridge-motor-control) motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop

the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

Enable pins 1 and 9 (corresponding to the two motors) must be high for motors to start operating. When an enable input is high, the associated driver gets enabled. As a result, the outputs become active and work in phase with their inputs. Similarly, when the enable input is low, that driver is disabled, and their outputs are off and in the high-impedance state.

#### Features of L293D driver module Hardware features

* + - * + can be used to run Two DC motors with the same IC.
        + Speed and Direction control is possible
        + Motor voltage Vcc2 (Vs): 4.5V to 36V
        + Maximum Peak motor current: 1.2A
        + Maximum Continuous Motor Current: 600mA
        + Supply Voltage to Vcc1(vss): 4.5V to 7V
        + Transition time: 300ns (at 5Vand 24V)
        + Automatic Thermal shutdown is available
        + Available in 16-pin DIP, TSSOP, SOIC packages

#### Applications

* + - * + Used to drive high current Motors using Digital Circuits
        + Can be used to drive Stepper motors
        + High current LED’s can be driven
        + Relay Driver module (Latching Relay is possible)

#### Pin description

The L293D driver module has 16pins. They are as follows:

#### ENABLE:

When enable is pulled low, the module is disabled which means the module will not turn on and it fails to drive motors. When enable is left open or connected to 3.3V, the module is enabled i.e the module remains on and driving of motors also takes place.

**VCC:**

#### GND:

Supply voltage 3.3v to 5v

Ground pin.

#### INPUT & OUTPUT:

These two pins acts as an input and output interface for communication.

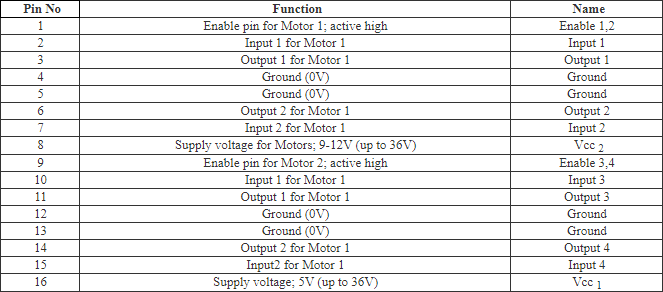


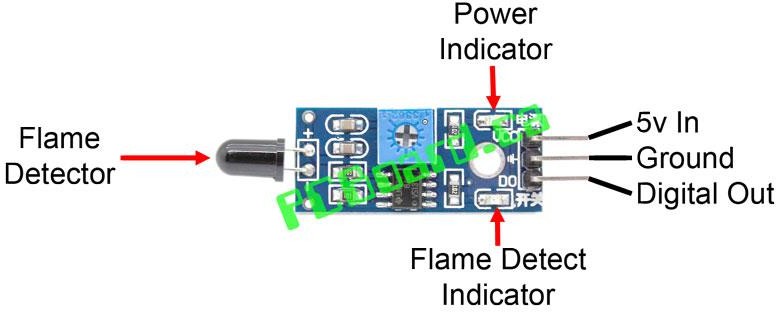
Table pin description table

#### Flame Sensor Module

A flame sensor module that consists of a flame sensor (IR receiver), resistor, capacitor, potentiometer, and comparator LM393 in an integrated circuit. It can detect infrared light with a wavelength ranging from 700nm to 1000nm.The far-infrared flame probe converts

the light detected in the form of infrared light into current changes. Sensitivity is adjusted through the onboard variable resistor with a detection angle of 60 degrees.

Working voltage is between 3.3v and 5.2v DC, with a digital output to indicate the presence of a signal. Sensing is conditioned by an LM393 comparator.



### Figure 3.2: flame sensor module

#### Photo diode

A photodiode is one type of light detector, used to convert the light into current or voltage based on the mode of operation of the device. It comprises of optical filters, built-in lenses and also surface areas. These diodes have a slow response time when the surface area of the photodiode increases. Photodiodes are alike to regular semiconductor diodes, but that they may be either visible to let light reach the delicate part of the device. [Several diodes](https://www.elprocus.com/3-different-types-diodes/) [intended for](https://www.elprocus.com/3-different-types-diodes/)use exactly as a photodiode will also use a PIN junction somewhat than the usual PN junction.

some photodiodes will look like [a light emitting diode.](https://www.elprocus.com/led-light-sources/) They have two terminals coming from the end. The smaller end of the diode is the cathode terminal, while the longer end of

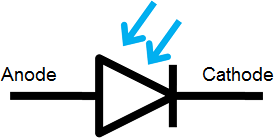
the diode is the anode terminal. See the following schematic diagram for the anode and cathode side. Under forward bias condition, conventional current will flow from the anode to the cathode, following the arrow in the diode symbol. Photocurrent flows in the reverse direction.

Fig Photo diode symbol

#### Applications of Photodiode

* + - * + The applications of photodiodes involve in similar applications of photodetectors like charge-coupled devices, photoconductors, and photomultiplier tubes.
        + These diodes are used in consumer electronics devices like [smoke detectors](https://www.elprocus.com/tutorial-on-smoke-detectors-and-fire-alarms/), compact disc players, and televisions and remote controls in VCRs.
        + In other consumer devices like clock radios, camera light meters, and street lights, photoconductors are more frequently used rather than photodiodes.
        + Photodiodes are frequently used for exact measurement of the intensity of light in science & industry. Generally, they have an enhanced, more linear response than photoconductors.
        + Photodiodes are also widely used in [numerous medical applications](https://www.elprocus.com/nanorobots-and-its-application-in-medicine/) like instruments to analyze samples, detectors for computed tomography and also used in blood gas monitors.
        + These diodes are much faster & more complex than normal PN junction diodes and hence are frequently used for lighting regulation and in optical communications.

#### DC Motor:

Motors convert electrical energy into mechanical energy.A **DC motor** is an electric motor that runs on direct current (DC) electricity.



Fig dc motor

In any electric motor, operation is based on simple electromagnetism. A current-carrying

conductor generates a magnetic field; when this is then placed in an external magnetic field, it will experience a force proportional to the current in the conductor, and to the strength of the external magnetic field. As you are well aware of from playing with magnets as a kid, opposite (North and South) polarities attract, while like polarities (North and North, South and South) repel. The internal configuration of a DC motor is designed to harness the magnetic interaction between a current-carrying conductor and an external magnetic field to generate rotational motion.

Direct current (DC) motors are widely used to generate motion in a variety of products. Permanent magnet DC (direct current) motors are enjoying increasing popularity in applications requiring compact size, high torque, high efficiency, and low power consumption.

In a brushed DC motor, the brushes make mechanical contact with a set of electrical contacts provided on a commutator secured to an armature, forming an electrical circuit between the DC electrical source and coil windings on the armature. As the armature rotates on an axis, the stationary brushes come into contact with different sections of the rotating commutator.

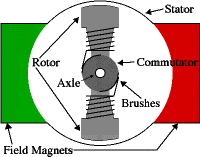


Fig internal architecture of dc motor

Permanent magnet DC motors utilize two or more brushes contacting a commutator which provides the direct current flow to the windings of the rotor, which in turn provide the desired magnetic repulsion/attraction with the permanent magnets located around the periphery of the motor.

The brushes are conventionally located in brush boxes and utilize a U-shaped spring which biases the brush into contact with the commutator. Permanent magnet brushless dc motors are widely used in a variety of applications due to their simplicity of design, high efficiency, and low noise. These motors operate by electronic commutation of stator windings rather than the conventional mechanical commutation accomplished by the pressing engagement of brushes against a rotating commutator.

A brushless DC motor basically consists of a shaft, a rotor assembly equipped with one or more permanent magnets arranged on the shaft, and a stator assembly which incorporates a stator component and phase windings. Rotating magnetic fields are formed by the currents applied to the coils.

The rotator is formed of at least one permanent magnet surrounded by the stator, wherein the rotator rotates within the stator. Two bearings are mounted at an axial distance to each other on the shaft to support the rotor assembly and stator assembly relative to each other. To achieve electronic commutation, brushless dc motor designs usually include an electronic controller for controlling the excitation of the stator windings.

#### Water Pump

The water pump is operated at 5v which can be interfaced with Arduino



Fig 5v water pump

#### Servo Motor

A servo is a small DC motor with the following components added: some gear reduction, a position sensor on the motor shaft, and an electronic circuit that controls the motor's operation. In other words, a servo is to a DC motor what the Arduino is the ATmega microcontroller---components and housing that make the motor easy to use. This will become abundantly clear when we work with unadorned DC motors next week.

The gear reduction provided in a servo is large; the basic hobby servo has a 180:1 gear ratio. This means that the DC motor shaft must make 180 revolutions to produce 1 revolution of the servo shaft. This large gear ratio reduces the speed of the servo and proportionately increases its torque. What does this imply about small DC motors? Servo motors are typically used for angular positioning, such as in radio control airplanes. They have a movement range of 0 up to 180 degrees, but some extend up to 210 degrees. Typically, a potentiometer measures the position of the output shaft at all times so the controller can accurately place and maintain its position.



Fig servo motor

PPM uses 1 to 2ms out of a 20ms timeperiod to encode its information. The servo expects to see a pulse every 20milliseconds (.02 seconds). The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse will make the motor turn to the 90 degree position(often called the neutral position). If the pulse is shorter than 1.5 ms, then the motor

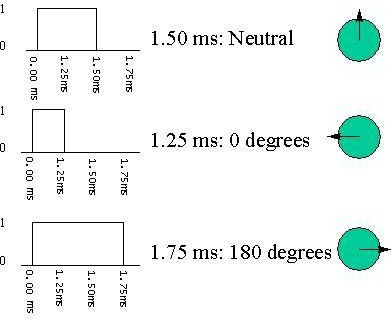
will turn the shaft to closer to 0degrees. If the pulse is longer than 1.5ms,the shaft turns closer to 180 degrees.The amount of power applied to the motor is proportional to the distance it needs to travel. So, if the shaft needs to turn a large distance, the motor will run at full speed. If it needs to turn only a small amount, the motor will run at a slower speed.

Fig rotating mechanism of servo motor

* 1. MICROCONTROLLER ATMEGA 328

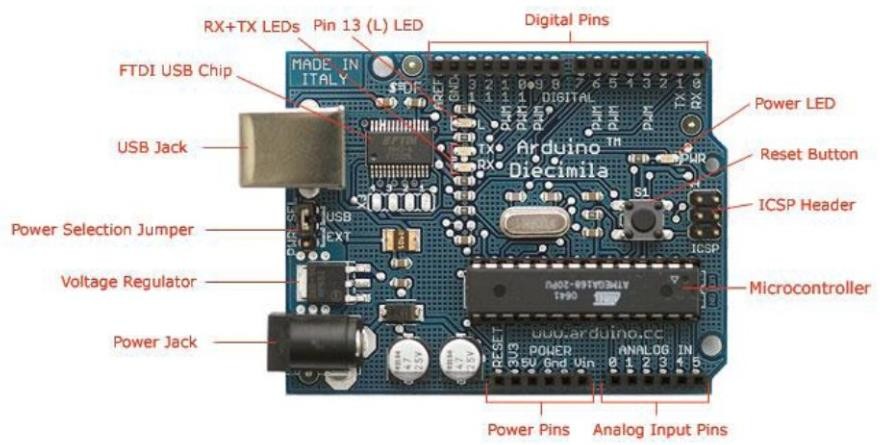


Figure 3.3: Arduino UNO board

The Atmel 8-bit AVR RISC-based microcontroller combines 32 KB ISP flash memory with read-while-write capabilities, 1 KB EEPROM, 2 KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte- oriented 2-wire serial interface, SPI serial port, 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts. The device achieves throughputs approaching 1 MIPS.

#### Applications

Today the ATmega328 is commonly used in many projects and autonomous systems where a simple, low-powered, low-cost micro-controller is needed. Perhaps the most common implementation of this chip is on the popular [Arduino](https://en.wikipedia.org/wiki/Arduino) development platform, namely the [Arduino Uno](https://en.wikipedia.org/wiki/Arduino_Uno) and [Arduino Nano](https://en.wikipedia.org/wiki/Arduino_Nano) models.

#### Features

* + - * 28-pin AVR Microcontroller
      * Flash Program Memory: 32 kbytes
      * EEPROM Data Memory: 1 kbytes
      * SRAM Data Memory: 2 kbytes
      * I/O Pins: 23
      * Timers: Two 8-bit / One 16-bit
      * A/D Converter: 10-bit Six Channel
      * PWM: Six Channels
      * RTC: Yes with Separate Oscillator
      * MSSP: SPI and I²C Master and Slave Support
      * USART: Yes
      * External Oscillator: up to 20MHz

The Atmega328 is a very popular microcontroller chip produced by Atmel. It is an 8-bit microcontroller that has 32K of flash memory, 1K of EEPROM, and 2K of internal SRAM.

The Atmega328 is one of the microcontroller chips that are used with the popular Arduino Duemilanove boards. The Arduino Duemilanove board comes with either 1 of 2 microcontroller chips, the Atmega168 or the Atmega328. Of these 2, the Atmega328 is the upgraded, more advanced chip. Unlike the Atmega168 which has 16K of flash program memory and 512 bytes of internal SRAM, the Atmega328 has 32K of flash program memory and 2K of Internal SRAM.

The Atmega328 has 28 pins, It has 14 digital I/O pins, of which 6 can be used as PWM outputs and 6 analog input pins. These I/O pins account for 20 of the pins.

### PIN DIAGRAM OF ATMEGA328

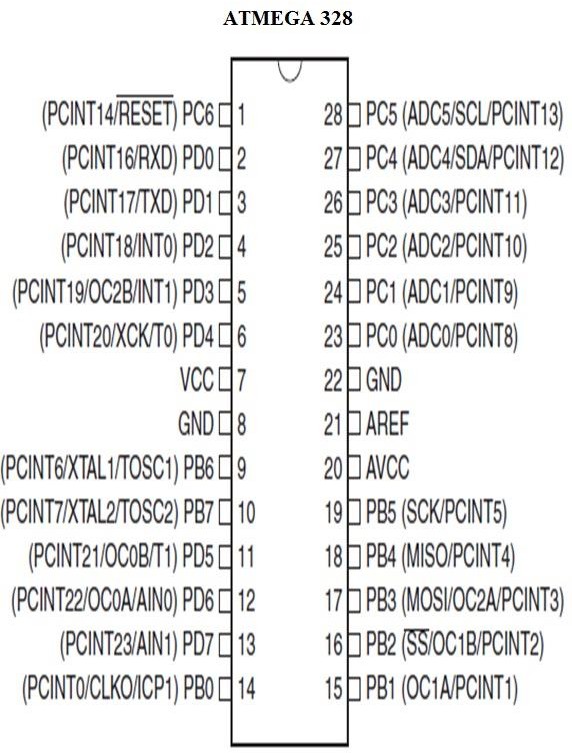
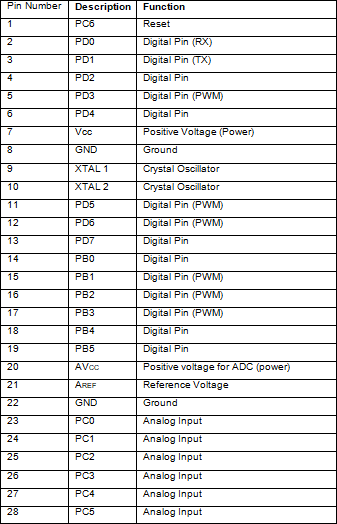


Figure 3.4: Pin diagram of Atmega328

Table 3.3: Description of each pins of ATmega328



As stated before, 20 of the pins function as I/O ports. This means they can function as an input to the circuit or as output. Whether they are input or output is set in the software. 14 of the pins are digital pins, of which 6 can function to give PWM output. 6 of the pins are for analog input/output. Two of the pins are for the crystal oscillator, this is to provide a clock pulse for the Atmega chip. A clock pulse is needed for synchronization so that

communication can occur in synchrony between the Atmega chip and a device that it is connected to.

The Atmega328 chip has an analog-to-digital converter (ADC) inside of it. This must be or else the Atmega328 wouldn't be capable of interpreting analog signals. Because there is an ADC, the chip can interpret analog input, which is why the chip has 6 pins for analog input. The ADC has 3 pins set aside for it to function- AVCC, AREF, and GND. AVCC is the power supply, positive voltage, that for the ADC. The ADC needs its own power supply in order to work. GND is the power supply ground. AREF is the reference voltage that the ADC uses to convert an analog signal to its corresponding digital value. Analog voltages higher than the reference voltage will be assigned to a digital value of 1, while analog voltages below the reference voltage will be assigned the digital value of 0. Since the ADC for the Atmega328 is a 10-bit ADC, meaning it produces a 10-bit digital value, it converts an analog signal to its digital value, with the AREF value being a reference for which digital values are high or low. Thus, a portrait of an analog signal is shown by this digital value; thus, it is its digital correspondent value. The last pin is the RESET pin. This allows a program to be rerun and start over. And this sums up the pin out of an Atmega328 chip.

### ARCHITECTURE

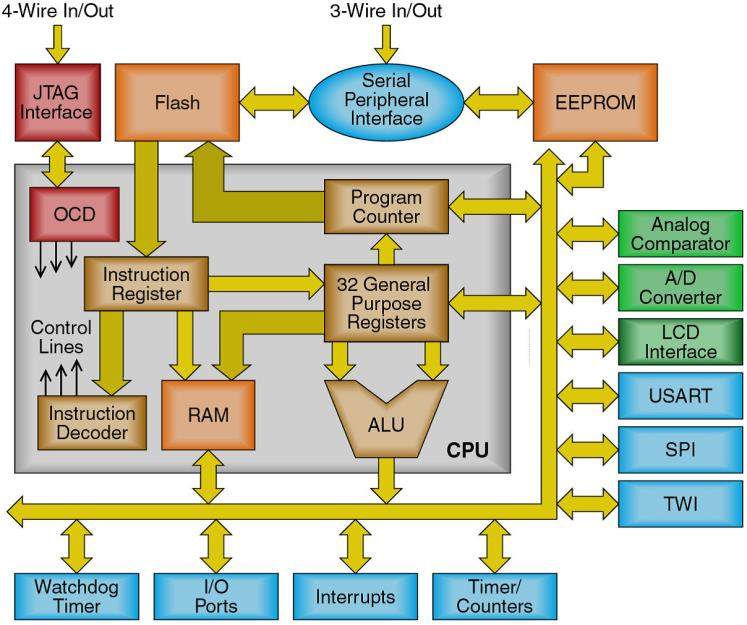


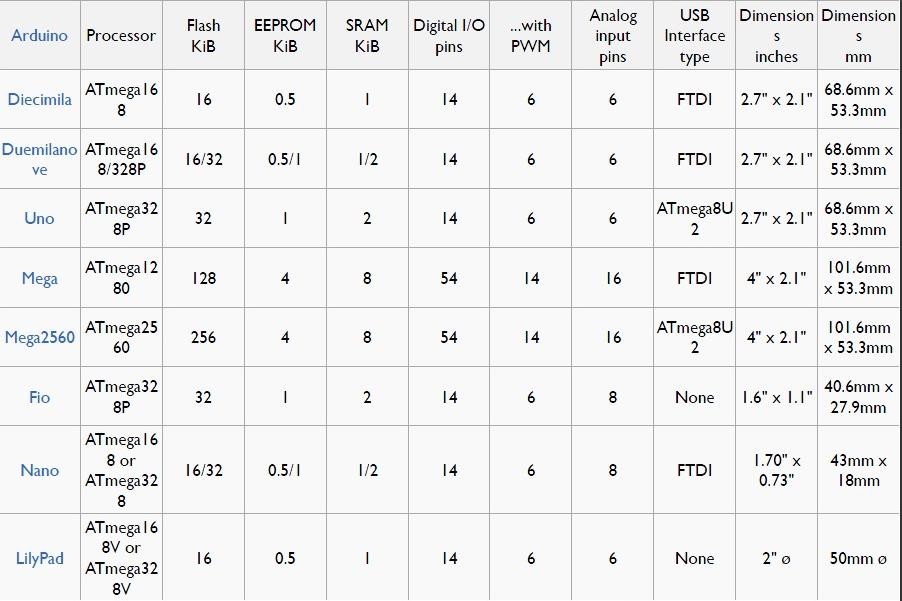
Figure 3.5: Architecture of AVR

### FEATURES OF AVR.

* High-performance, Low-power AVR® 8-bit Microcontroller
* Advanced RISC Architecture
  + 131 Powerful Instructions – Most Single-clock Cycle Execution
  + 32 x 8 General Purpose Working Registers
  + Fully Static Operation
  + Up to 16 MIPS Throughput at 16 MHz
  + On-chip 2-cycle Multiplier
* Nonvolatile Program and Data Memories
  + 32K Bytes of In-System Self-Programmable Flash
* Endurance: 10,000 Write/Erase Cycles
  + Optional Boot Code Section with Independent Lock Bits
* In-System Programming by On-chip Boot Program
* True Read-While-Write Operation
  + 1024 Bytes EEPROM
* Endurance: 100,000 Write/Erase Cycles
  + 2K Byte Internal SRAM
  + Programming Lock for Software Security
* JTAG (IEEE std. 1149.1 Compliant) Interface
  + Boundary-scan Capabilities According to the JTAG Standard
  + Extensive On-chip Debug Support
  + Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
* Peripheral Features
  + Two 8-bit Timer/Counters with Separate Pre scalers and Compare Modes
  + One 16-bit Timer/Counter with Separate Pre scaler, Compare Mode, and Capture Mode
  + Real Time Counter with Separate Oscillator
  + Four PWM Channels
  + 8-channel, 10-bit ADC
* 8 Single-ended Channels
* 7 Differential Channels in TQFP Package Only
* 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  + Byte-oriented Two-wire Serial Interface
  + Programmable Serial USART
  + Master/Slave SPI Serial Interface
  + Programmable Watchdog Timer with Separate On-chip Oscillator
  + On-chip Analog Comparator
* Special Microcontroller Features
  + Power-on Reset and Programmable Brown-out Detection
  + Internal Calibrated RC Oscillator
  + External and Internal Interrupt Sources
  + Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
* I/O and Packages
  + 32 Programmable I/O Lines
  + 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
* Operating Voltages
  + 2.7 - 5.5V for ATmega32L
  + 4.5 - 5.5V for ATmega32
* Speed Grades
  + 0 - 8 MHz for ATmega32L
  + 0 - 16 MHz for ATmega32
* Power Consumption at 1 MHz, 3V, 25°C for ATmega32L
  + Active: 1.1 mA
  + Idle Mode: 0.35 mA
  + Power-down Mode: < 1 μA

### DIFFERENT FLAVOURS OF ARDUINO WITH THEIR CONFIGURATION

Table 3.4: Different Flavors of ARDUINO with their Configuration



### BASIC TERMINOLOGIES IN ARDUINO

#### Analog to Digital Converter (ADC)

* + - * The process of Analog to digital conversion is shown in figure.
      * The Arduino has 10 bits of Resolution when reading analog signals.
      * 210=1024 increments.
      * Influence also by how fast you sample.

#### Pulse Width Modulation (PWM)

* + - * The Arduino has 8bit of resolution, when outputting a signal using PWM.
      * The range of output voltage is from 0 to 5 Volts.
      * 28=255 Increments
      * Average of on/off(digital signals to make an average voltage),Duty cycle in 100% of 5Volts.

### 3. 8 LANGUAGE REFERENCES:

The Microcontroller on the board is programmed using the Arduino programming languag**e** (based on wiring) and the arduino development environment (based on processing).

#### Arduino programming language (APL) (based on wiring)

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

#### Wiring

Wiring is an open-source programming framework for microcontrollers. Wiring allows writing cross-platform software to control devices attached to a wide range of

microcontroller boards to create all kinds of creative coding, interactive objects, spaces or physical experiences. The framework is thoughtfully created with designers and artists in mind to encourage a community where beginners through experts from around the world share ideas, knowledge and their collective experience. There are thousands of students, artists, designers, researchers, and hobbyists who use Wiring for learning, prototyping, and finished professional work production.

### ARDUINO DEVELOPMENT ENVRONMENT

#### Processing

Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

### SOFTWARE REQUIREMENTS

#### Embedded C

Embedded C is a set of language extensions for the C Programming language by the C Standards committee to address commonality issues that exist between C extensions for different embedded systems. Historically, embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed- point arithmetic, multiple distinct memory banks, and basic I/O operations.

#### Difference between C and Embedded C

Though C and embedded C appear different and are used in different contexts, they have more similarities than the differences. Most of the constructs are same; the difference lies in their applications.

C is used for desktop computers, while embedded C is for microcontroller based applications. C takes more resources of a desktop PC like memory, OS, etc. while programming on desktop systems what embedded C cannot. Embedded C has to use the limited resources (RAM, ROM, I/O’s) on an embedded processor. Thus, program code must fit into the available program memory. If code exceeds the limit, the system is likely to crash.

Compilers for C (ANSI C) typically generate OS dependent executable files. Embedded C requires compilers to create files to be downloaded to the microcontrollers/microprocessors where it needs to run. Embedded compilers give access to all resources which is not provided in compilers for desktop computer applications.

Embedded systems often have the real-time constraints, which is usually not there with desktop computer applications.

Embedded systems often do not have a console, which is available in case of desktop applications.

The C programming language is perhaps the most popular programming language for programming embedded systems. C continues to be a very popular language for micro- controller developers/programmers due to the code efficiency and reduced overhead and development time. C offers low-level control and is considered more readable than assembly language which is a little difficult to understand. Assembly language requires more code writing, whereas C is easy to understand and requires less coding. Plus, using C increases portability, since C code can be compiled for different types of processors. We can program microcontrollers using Atmel Atmega328, AVR or PIC.

Here by developing the programs as per the electronic hardware using Atmel Atmega328 micro controller. For the operations like: blink led, increment decrement counters, token displays etc.

Most C programmers are spoiled because they program in environments where not only there is a standard library implementation, but there are frequently a number of other

libraries available for use. The cold fact is, that in embedded systems, there rarely are many of the libraries that programmers have grown used to, but occasionally an embedded system might not have a complete standard library, if there is a standard library at all. Few embedded systems have capability for dynamic linking, so if standard library functions are to be available at all, they often need to be directly linked into the executable. Oftentimes, because of space concerns, it is not possible to link in an entire library file, and programmers are often forced to "brew their own" standard c library implementations if they want to use them at all. While some libraries are bulky and not well suited for use on microcontrollers, many development systems still include the standard libraries which are the most common for C programmers.

C remains a very popular language for micro-controller developers due to the code efficiency and reduced overhead and development time. C offers low-level control and is considered more readable than assembly. Many free C compilers are available for a wide variety of development platforms. The compilers are part of an IDEs with ICD support, breakpoints, single-stepping and an assembly window. The performance of C compilers has improved considerably in recent years, and they are claimed to be more or less as good as assembly, depending on who you ask. Most tools now offer options for customizing the compiler optimization. Additionally, using C increases portability, since C code can be compiled for different types of processors.

#### Software

The software used by the arduino is Arduino IDE. The Arduino IDE is a cross- platform application written in [Java,](http://en.wikipedia.org/wiki/Java_(programming_language)) and is derived from the IDE for the [Processing](http://en.wikipedia.org/wiki/Processing_(programming_language)) [programming language](http://en.wikipedia.org/wiki/Processing_(programming_language)) and the Wiring project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as [syntax highlighting,](http://en.wikipedia.org/wiki/Syntax_highlighting) [brace matching,](http://en.wikipedia.org/wiki/Brace_matching) and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit [make files](http://en.wikipedia.org/wiki/Makefiles) or run programs on a [command-](http://en.wikipedia.org/wiki/Command-line_interface) [line interface.](http://en.wikipedia.org/wiki/Command-line_interface) Although building on command-line is possible if required with some

third-party tools such as [Ino](http://inotool.org/).

The Arduino IDE comes with a [C](http://en.wikipedia.org/wiki/C_(programming_language))/C++ library called "Wiring" (from the project of the same name), which makes many common input/output operations much easier. Arduino programs are written in C/C++, although users only need define two functions to make a runnable program:

* + - * setup() – a function run once at the start of a program that can initialize settings
      * loop() – a function called repeatedly until the board powers off

A typical first program for a microcontroller simply blinks a LED on and off. In the Arduino environment, the user might write a program like this:

#define LED\_PIN 13

void setup (){

pinMode(LED\_PIN, OUTPUT);*//* enable pin 13 for digital output

}

void loop (){

digitalWrite(LED\_PIN, HIGH);// turn on the LED delay(1000);// wait one second (1000 milliseconds)



Figure 3.6: A Screenshot of Arduino IDE

For the above code to work correctly, the positive side of the LED must be connected to pin 13 and the negative side of the LED must be connected to ground. The above code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple [main() function](http://en.wikipedia.org/wiki/Main_function) at the bottom, to make it a valid C++ program.

The Arduino IDE uses the [GNU tool chain](http://en.wikipedia.org/wiki/GNU_toolchain) and [AVR Libc](http://en.wikipedia.org/w/index.php?title=AVR_Libc&action=edit&redlink=1) to compile programs, and uses [AVR dude](http://en.wikipedia.org/w/index.php?title=Avrdude&action=edit&redlink=1) to upload programs to the board.

For educational purposes there is third party graphical development environment called Mini blog available under a different open source license.

PROGRAM COMPILING

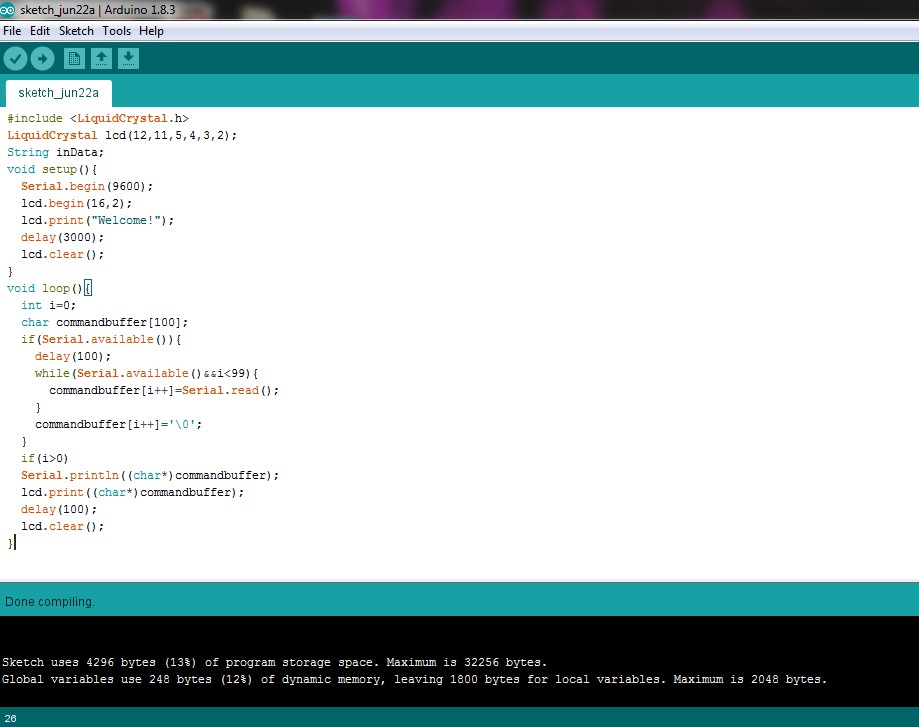


Figure 3.7: Program compiling using arduino IDE.

SELECTING BOARD

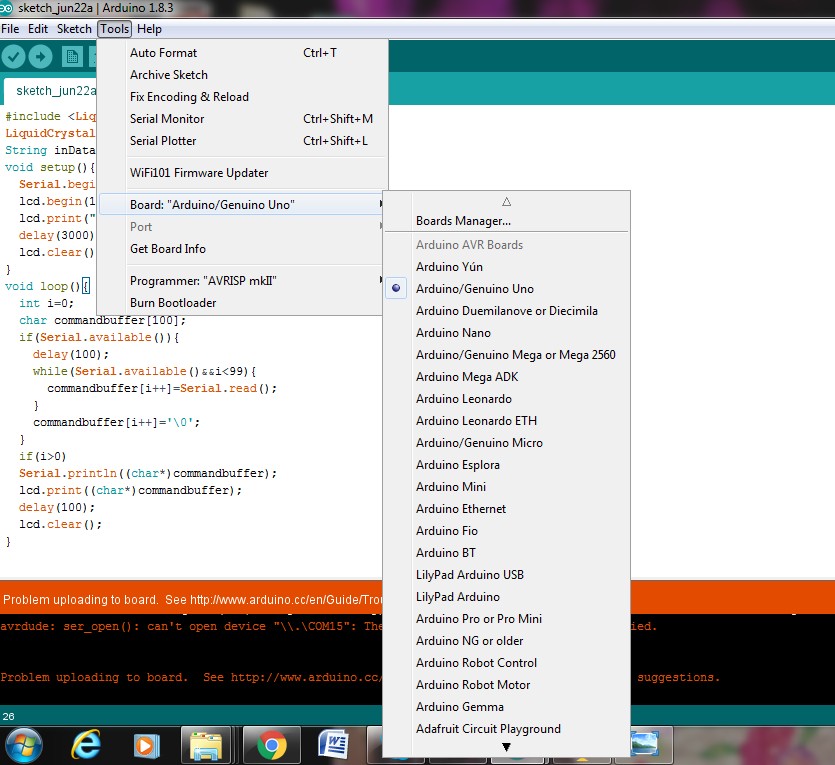


Figure 3.8: Selecting the board from Tools menu SELECTING PORT

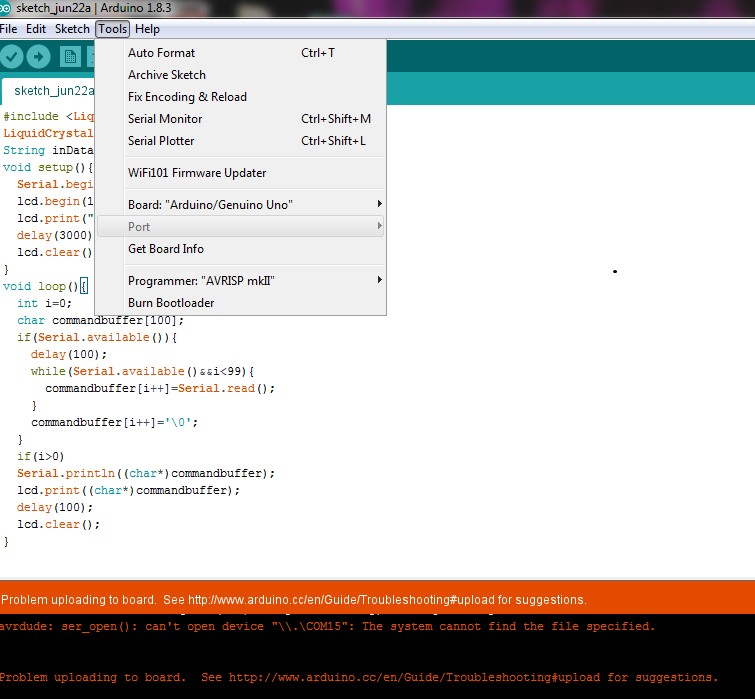


Figure 3.9: Selecting the port

UPLOADING PROGRAM

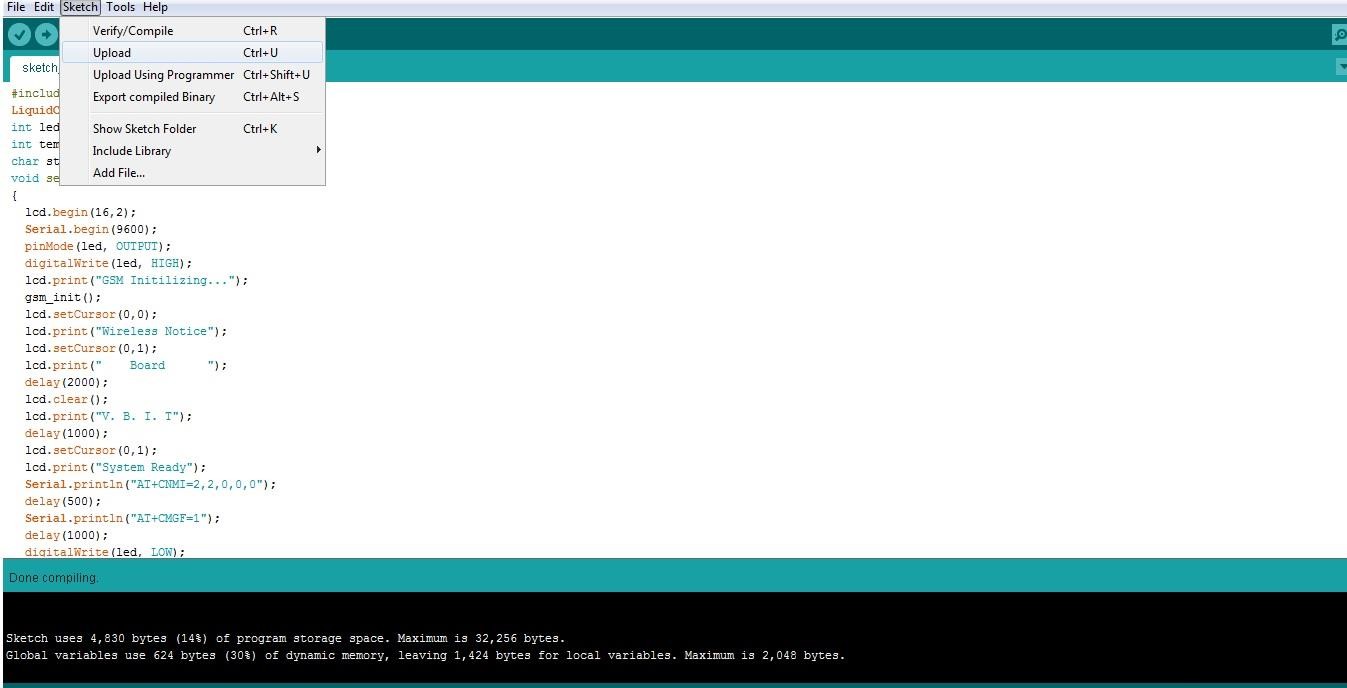


Figure 3.10: Uploading program to the arduino.

#### Language Reference

Arduino programs can be divided in three main parts: structure*,* values (variables and constants), and functions*.*

Available data types in ARDUINO IDE are

* void
* boolean
* char **( 0 – 255)**
* byte - **8 bit data ( 0 – 255)**

#### int - 16-bit data (32,767 - -32,768)

* long – **32 bit data (2,147,483,647 to -2,147,483,648**)
* float
* double
* string - char array
* String - object
* array

### AT COMMANDS

AT commands are used to control MODEMs. AT is the abbreviation for Attention. These commands come from Hayes commands that were used by the Hayes smart modems. The Hayes commands started with AT to indicate the attention from the MODEM. The dial up and wireless MODEMs (devices that involve machine to machine communication) need AT commands to interact with a computer. These include the Hayes command set as a subset, along with other extended AT commands**.**

AT commands with a mobile phone can be used to access following information and services:

1. Information and configuration pertaining to mobile device or Bluetooth module.
2. SMS services.
3. MMS services.
4. Fax services.
5. Data and Voice link over mobile network.

The Hayes subset commands are called the basic commands and the commands specific to a Bluetooth network are called extended AT commands.

### TYPES OF AT COMMANDS:

There are four types of AT commands:

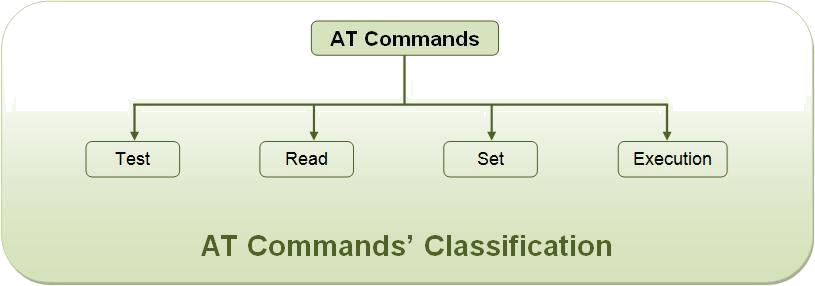


Fig 3.11: AT commands classification

#### Explanation of commonly used AT commands

**1) AT** - This command is used to check communication between the module and the computer.

For example, AT OK

The command returns a result code OK if the computer (serial port) and module are connected properly. If any of module or SIM is not working, it would return a result code ERROR.

**2) +CMGF** - This command are used to set the SMS mode. Either text or PDU mode can be selected by assigning 1 or 0 in the command.

SYNTAX: AT+CMGF=<mode>

0: for PDU mode

1: for text mode

The text mode of SMS is easier to operate but it allows limited features of SMS. The PDU (protocol data unit) allows more access to SMS services but the operator requires bit level knowledge of TPDUs. The headers and body of SMS are accessed in hex format in PDU mode so it allows availing more features.

For example, AT+CMGF=1 OK

**3) +CMGW** - This command is used to store message in the SIM.

SYNTAX: AT+CMGW=” Phone number”>Message to be stored Ctrl+z

As one types AT+CMGW and phone number, „>‟ sign appears on next line where one can type the message. Multiple line messages can be typed in this case. This is why the message is terminated by providing a „Ctrl+z‟ combination. As Ctrl+z is pressed, the following information response is displayed on the screen.+CMGW: Number on which message has been stored

**4) +CMGS** - This command is used to send a SMS message to a phone number.

SYNTAX: AT+CMGS= serial number of message to be send.

As the command AT+CMGS and serial number of message are entered, SMS is sent to the particular SIM.

For example, AT+CMGS=1 OK

1. **ATD** - This command is used to dial or call a number.

SYNTAX: ATD<Phone number> (Enter)

For example, ATD123456789

1. **ATA** - This command is used to answer a call. An incoming call is indicated by a message „RING‟ which is repeated for every ring of the call. When the call ends „NO CARRIER‟ is displayed on the screen.

SYNTAX: ATA (Enter)

As ATA followed by enter key is pressed, incoming call is answered. For example, RING

RING ATA

1. **ATH** - This command is used to disconnect remote user link with the GSM module.

SYNTAX: ATH (Enter)

### LIST OF AT COMMANDS

The AT commands for both, bluetooth module and the mobile phone, are listed below. Some of these commands may not be supported by all the bluetooth modules available. Also there might be some commands which won’t be supported by some mobile handsets.

TABLE 3.5: AT Commands

|  |  |
| --- | --- |
| **Command** | **Description** |
| **AT** | Checking communication between  the module and computer. |

### CALL CONTROL

|  |  |
| --- | --- |
| **Command** | **Description** |
| **ATA** | Answer command |
| **ATD** | Dial command |
| **ATH** | Hang up call |
| **ATL** | Monitor speaker mode |
| **ATM** | Monitor speaker mode |
| **ATO** | Go on-line |
| **ATP** | Set pulse dial as default |
| **ATT** | Set tone dial as default |
| **AT+CSTA** | Select type of address |
| **AT+CRC** | Cellular result codes |

DATA CARD CONTROL

|  |  |
| --- | --- |
| **Command** | **Description** |
| **ATI** | Identification |
| **ATS** | Select an S-register |
| **ATZ** | Recall stored profile |
| **AT&F** | Restore factory settings |
| **AT&V** | View active configuration |
| **AT&W** | Store parameters in given profile |
| **AT&Y** | Select set as power up option |
| **AT +CLCK** | Facility lock command |

|  |  |
| --- | --- |
| **AT +COLP** | Connected line identification  presentation |
| **AT +GCAP** | Request complete capabilities list |
| **AT+GMI** | Request manufacturer  identification |
| **AT+GMM** | Request model identification |
| **AT+GMR** | Request revision identification |
| **AT+GSN** | Request product serial number  identification (IMEI) |

PHONE CONTROL

|  |  |
| --- | --- |
| **Command** | **Description** |
| **AT+CBC** | Battery Charge |
| **AT+CGMI** | Request manufacturer  identification |
| **AT+CGMM** | Request model identification |
| **AT+CGMR** | Request revision identification |
| **AT+CGSN** | Request product serial number  identification |
| **AT+CMEE** | Report mobile equipment error |
| **AT+CPAS** | Phone activity status |
| **AT+CPBF** | Find phone book entries |
| **AT+CPBR** | Read phone book entries |
| **AT+CPBS** | Select phone book memory storage |
| **AT+CPBW** | Write phone book entry |
| **AT+CSCS** | Select TE character set |

|  |  |
| --- | --- |
| **AT+CSQ** | Signal quality |

COMPUTER DATA INTERFACE

|  |  |
| --- | --- |
| **Command** | **Description** |
| **ATE** | Command Echo |
| **ATQ** | Result code suppression |
| **ATV** | Define response format |
| **ATX** | Response range selection |
| **AT&C** | Define DCD usage |
| **AT&D** | Define DTR usage |
| **AT&K** | Select flow control |
| **AT&Q** | Define communications mode  option |
| **AT&S** | Define DSR option |
| **AT+ICF** | DTE-DCE character framing |
| **AT+IFC** | DTE-DCE Local flow control |
| **AT+IPR** | Fixed DTE rate |

SERVICE

|  |  |
| --- | --- |
| **Command** | **Description** |
| **AT+CLIP** | Calling line identification  presentation |
| **AT+CR** | Service reporting control |
| **ATV+DR** | Data compression reporting |

NETWORK COMMUNICATION PARAMETER

|  |  |
| --- | --- |
| **Command** | **Description** |
| **ATB** | Communications standard option |
| **AT+CBST** | Select bearer service type |
| **AT+CEER** | Extended error report |
| **AT+CRLP** | Radio link protocol |
| **AT+DS** | Data compression |

MISCEELANEOUS

|  |  |
| --- | --- |
| **Command** | **Description** |
| **AT/** | Re-execute command line |
| **AT?** | Command help |
| **AT\*C** | Start SMS interpreter |
| **AT\*T** | Enter |
| **AT\*V** | Activate V.25bis mode |
| **AT\*NOKIA TEST** | Test command |
| **AT+CESP** | Enter SMS block mode protocol |

SMS TEXT MODE:

|  |  |
| --- | --- |
| **Command** | **Description** |
| **AT+CSMS** | Select message service |
| **AT+CPMS** | Preferred message storage |
| **AT+CMGF** | Message format |
| **AT+CSCA** | Service center address |

|  |  |
| --- | --- |
| **AT+CSMP** | Set text mode parameters |
| **AT+CSDH** | Show text mode parameters |
| **AT+CSCB** | Select cell broadcast message types |
| **AT+CSAS** | Save settings |
| **AT+CRES** | Restore settings |
| **AT+CNMI** | New message indications to TE |
| **AT+CMGL** | List messages |
| **AT+CMGR** | Read message |
| **AT+CMGS** | Send message |
| **AT+CMSS** | Send message from storage |
| **AT+CMGW** | Write message to memory |
| **AT+CMGD** | Delete message |

SMS PDU MODE

|  |  |
| --- | --- |
| **Command** | **Description** |
| **AT+CMGL** | List messages |
| **AT+CMGR** | Read message |
| **AT+CMGS** | Send message |
| **AT+CMGW** | Write message to memory |

**CHAPTER 4**

# SYSTEM MODELLING AND DESIGN

### FUNCTIONAL DESCRIPTION

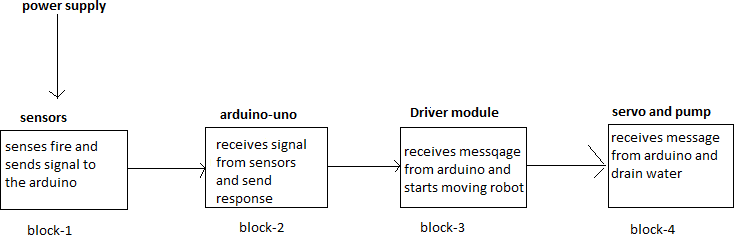


Fig 4.1: Functional description diagram The constituent parts involved in the process are

* + - sensors
    - Arduino with Atmel Atmega328 microcontroller
    - L293 driver module
    - Servo with pump

First block portrays to be sensors which receives, verifies and forwards the message to the Microcontroller. Micro is the second block. Micro processes the message and sends to the driver module. Driver module behaving as the third constituent part and servo pump acts as fourth part which extinguishes the fire

#### DATA FLOW DIAGRAM

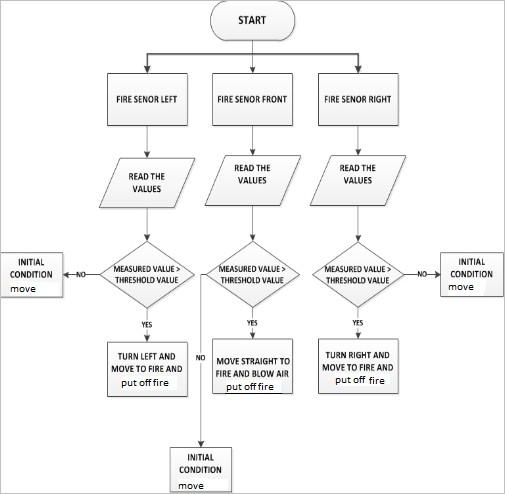


Fig 4.4: Flow chart

The flow starts by initializing the ports of components. First the power supply should be on to the circuit and three sensors are there one on middle and remaing two on right and left side of chassis whenever the fire is occurred the respective value is read by the sensors when fire is occurred the voltage becomes zero and chassis is moved to the respective and put off fire whenever there is no fire then there is no input is occurred occurred voltage is more than 0 volts and the initial condition is move to other direction

# CHAPTER 5: IMPLEMENTATION AND TESTING

### MICROCONTROLLER – FLAME SENSOR INTERFACING

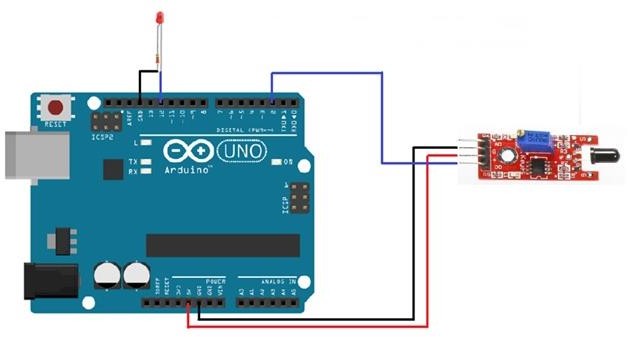


Fig 5.1: Microcontroller –flame sensor interfacing

Fig 5.1 The flame sensor is used to detect the fire or other [light sources](http://microcontrollerslab.com/photo-resistor-interfacing-arduino-light/) which are in the range of wavelength from 760nm to 1100nm. The module consists of an [IR sensor](http://microcontrollerslab.com/ir-receiver-transmitter-arduino/), potentiometer, [OP-Amp](http://microcontrollerslab.com/zero-crossing-detection-using-simple-op-amp/) circuitry and a [led](http://microcontrollerslab.com/led-blinking-arduino-uno-r3/) indicator. When a flame will be detected, the module will turn on its red led. This module is sensitive to flame but it can also detect ordinary light. The detection point is 60 degrees. The sensitivity of this sensor is adjustable and it also has a stable performance.

It has both outputs, analog and digital. The analog output gives us a real time voltage output signal on thermal resistance while the digital output allows us to set a threshold via a potentiometer. In our tutorial we are going to use both of these outputs one by one and see how the sensor works.We can use the flame sensor to make an alarm when detecting the fire, for safety purpose in many projects and in many more ways.

# CHAPTER 6: ADVANTAGES AND APPLICATIONS

### ADVANTAGES

* Prevention from dangerous incidents.
* Minimization of o ecological consequences
* financial loss can be prevented.
* a threat to a human life can be minimized.
* No supervision is required to control robot

### DISADVANTAGES

* It is applicable only for shorter distances
* Doesn’t predict nor interfere with operator’s thoughts.
* Cannot force directly the operator to work.

### APPLICATIONS

* + - Can be used in extinguishing fire where probability of explosion is high. For eg. Hotel kitchens, LPG/CNG gas stores, etc.
    - Can be used in Server rooms for immediate action in case of fire
    - Can be used in extinguishing fire where probability of explosion is high. For eg. Hotel kitchens, LPG/CNG gas stores, etc.
    - Every working environment requiring permanent operator's attention, At power plant control rooms
    - Can be used in serach and rescue operation
    - Can Used in domestic cold storage places

# CONCLUSION

The prototype of the fire fighter robot was efficiently designed. This prototype has facilities to be integrated with many sensors making it move forward. The toolkit detects the infrared light emitted by the fire with photo diode and sends signal to controller. We intend to extend this work to provide a keypad programmed to allow manipulation of robot to move desired direction with help of motor driver module and extinguish the flames using water tank which is rotated at 180 degress with help of servo in order for faster result. This future work will also explore to the use of a long distance sensor with suitable hardware to get more better and faster results addition to the characters.

## FUTURE SCOPE

The project has been motivated by the desire to design a system that can detect fires and take appropriate action, without any human intervention. The development of sensor networks and the maturity of robotics suggests that we can use mobile agents for tasks that involve perception of an external stimulus and reacting to the stimulus, even when the reaction involves a significant amount of mechanical actions. This provides us the opportunity to pass on to robots tasks that traditionally humans had to do but were inherently life- threatening. Fire-fighting is an obvious candidate for such automation. Given the number of lives lost regularly in fire- fighting, the system we envision is crying for adoption. Our experience suggests that designing a fire-fighting system with sensors and robots is within the reach of the current sensor network and mobile agent technologies. Furthermore, we believe that the techniques developed in this work will carry over to other areas involving sensing and reacting to stimulus, where we desire to replace the human with an automated mobile agent.

However, there has been research on many of these pieces in different contexts,

e.g. coordination among mobile agents, techniques for detecting and avoiding obsta cles, on-thefly communication between humans and mobile agents, etc. It will be both interesting and challenging to put all this together into a practical, autonomous fire-fighting service.

## REFERENCES

1. P. D. Minns, C Programming For the PC the MAC and the Arduino Microcontroller System. Author House, 2013
2. M. Banzi, Getting started with arduino. " O'Reilly Media, Inc.", 2009
3. A. M. Gibb, New media art, design, and the Arduino microcontroller: A malleable tool. PhD thesis, Pratt Institute, 2010
4. M. Margolis, Arduino cookbook. " O'Reilly Media, Inc.", 2011
5. D. Mellis, M. Banzi, D. Cuartielles, and T. Igoe, "Arduino: An open electronic prototyping platform, " in Proc. CHI, vol. 2007, 2007
6. C. K. Joo, Y. C. Kim, M. H. Choi, and Y. J. Ryoo, Self localization for intelligent mobile robot using multiple infrared range scanning system, In Control, Automation and Systems ICCAS'07, Seoul, Korea, 2007, 606-609.
7. <http://electronicsforu.com/electronics-projects/hardware-diy/arduino-ir-firefighter>robot
8. <http://maker.robotistan.com/arduino-dersleri>
9. J. Xu, W. Coombe, N. Boyson, A. Ohira, X. Gu, 143.472 Industrial Systems Design and Integration Fire Fighting Robot, 2006

## APPENDIX - A

#include <Servo.h>

Servo myservo;

int pos = 0;

boolean fire = false;

/\*-------defining Inputs------\*/

#define Left\_S A2 // left sensor

#define Right\_S A0 // right sensor

#define Forward\_S A1 //forward sensor

/\*-------defining Outputs------\*/

#define LM1 10 // left motor

#define LM2 11 // left motor

#define RM1 12 // right motor

#define RM2 13 // right motor

#define pump A5

void setup()

{

pinMode(Left\_S, INPUT);

pinMode(Right\_S, INPUT);

pinMode(Forward\_S, INPUT);

pinMode(LM1, OUTPUT);

pinMode(LM2, OUTPUT);

pinMode(RM1, OUTPUT);

pinMode(RM2, OUTPUT);

pinMode(pump, OUTPUT);

myservo.attach(9);

myservo.write(90);

}

void put\_off\_fire()

{

delay (500);

digitalWrite(LM1, HIGH);

digitalWrite(LM2, HIGH);

digitalWrite(RM1, HIGH);

digitalWrite(RM2, HIGH);

digitalWrite(pump, HIGH); delay(500);

for (pos = 0; pos <= 180; pos += 1)

{

myservo.write(pos);

delay(10);

}

for (pos = 180; pos >= 0; pos -= 1)

{

myservo.write(pos);

delay(10);

}

for (pos = 0; pos <= 180; pos += 1)

{

myservo.write(pos);

delay(10);

}

for (pos = 180; pos >= 0; pos -= 1)

{

myservo.write(pos);

delay(10);

}

digitalWrite(pump,LOW);

myservo.write(90);

fire=false;

}

void loop()

{

myservo.write(90); //Sweep\_Servo();

if (digitalRead(Left\_S) ==1 && digitalRead(Right\_S)==1 && digitalRead(Forward\_S) ==1) //If Fire not detected all sensors are zero

{

//Do not move the robot

digitalWrite(LM1, HIGH);

digitalWrite(LM2, HIGH);

digitalWrite(RM1, HIGH);

digitalWrite(RM2, HIGH);

}

else if (digitalRead(Forward\_S) ==0) //If Fire is straight ahead

{

//Move the robot forward

digitalWrite(LM1, HIGH);

digitalWrite(LM2, LOW);

digitalWrite(RM1, HIGH);

digitalWrite(RM2, LOW);

fire = true;

}

else if (digitalRead(Left\_S) ==0) //If Fire is to the left

{

//Move the robot left

digitalWrite(LM1, 1);

digitalWrite(LM2, 0);

digitalWrite(RM1, 0);

digitalWrite(RM2, 1);

}

else if (digitalRead(Right\_S) ==0) //If Fire is to the right

{

//Move the robot right

digitalWrite(LM1, 0);

digitalWrite(LM2, 1);

digitalWrite(RM1, 1);

digitalWrite(RM2, 0);

}

delay(300); //Slow down the speed of robot

while (fire == true)

{

put\_off\_fire();

}

}