INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Autonomous robotic systems require advanced obstacle detection and navigation mechanisms to operate efficiently in dynamic environments. Traditional navigation methods using fixed ultrasonic sensors often result in limited detection range and inaccurate path selection. This project proposes an omnidirectional navigation system that integrates an ultrasonic sensor, a servo motor, and an L293D motor driver to optimize robotic movement. The ultrasonic sensor is mounted on a servo motor, allowing it to scan a 180-degree area and improve obstacle detection accuracy. The L293D motor driver controls the movement of the robotic vehicle, ensuring smooth navigation based on real-time sensor data. The system employs a mean-based algorithm to minimize false positives in obstacle detection and optimize pathfinding. This approach significantly improves robotic efficiency, making it suitable for applications such as industrial automation, search-and-rescue operations, and autonomous vehicles.

Keywords – L293D, Ultrasonic Sensor, 16×2 LCD, Servo Motor

CHAPTER – 1

EMBEDDED SYTEMS

1.1 Introduction:

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, sometimes 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 load and peripherals to be connected.

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines, and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with a programming interface, and

embedded systems programming is a specialized occupation. Certain operating systems or language platforms are tailored for the embedded market, such as Embedded Java and Windows XP Embedded. However, some low-end consumer products use very inexpensive microprocessors and limited storage, with the application and operating system both part of a single program. The program is written permanently into the system's memory in this case, rather than being loaded into RAM (random access memory), as programs on a personal computer are

1.2 Characteristic Of Embedded System

- Speed (bytes/sec): Should be high speed
- Power (watts): Low power dissipation
- Size and weight: As far as possible small in size and low weight
- Accuracy (%error): Must be very accurate
- Adaptability: High adaptability and accessibility
- Reliability: Must be reliable over a long period of time

1.3 Applications Of Embedded System

We are living in the Embedded World. You are surrounded with many embedded products and your daily life largely depends on the proper functioning of these gadgets. Television, Radio, CD player of your living room, Washing Machine or Microwave Oven in your kitchen, Card readers, Access Controllers, Palm devices of your work space enable you to do many of your tasks very effectively. Apart from all these, many controllers embedded in your car take care of car operations between the bumpers and most of the times you tend to ignore all these controllers.

- **Robotics:** industrial robots, machine tools, Robocop soccer robots
- **Automotive:** cars, trucks, trains
- **Aviation:** airplanes, helicopters
- Home and Building Automation
- **Aerospace:** rockets, satellites

- **Energy systems:** windmills, nuclear plants
- **Medical systems:** prostheses, revalidation machine.

1.4 Microcontroller Versus Microprocessor

What is the difference between a Microprocessor and Microcontroller? By microprocessor is meant the general-purpose Microprocessors such as Intel's X86 family (8086, 80286, 80386, 80486, and the Pentium) or Motorola's 680X0 family (68000, 68010, 68020, 68030, 68040, etc). These microprocessors contain no RAM, no ROM, and no I/O ports on the chip itself. For this reason, they are commonly referred to as general-purpose Microprocessors.

A system designer using a general-purpose microprocessor such as the Pentium or the 68040 must add RAM, ROM, I/O ports, and timers externally to make them functional. Although the addition of external RAM, ROM, and I/O ports makes these systems bulkier and much more expensive, they have the advantage of versatility such that the designer can decide on the amount of RAM, ROM and I/O ports needed to fit the task at hand. This is not the case with Microcontrollers.

A Microcontroller has a CPU (a microprocessor) in addition to a fixed amount of RAM, ROM, I/O ports, and a timer all on a single chip. In other words, the processor, the RAM, ROM, I/O ports and the timer are all embedded together on one chip; therefore, the designer cannot add any external memory, I/O ports, or timer to it. The fixed amount of on-chip ROM, RAM, and number of I/O ports in Microcontrollers makes them ideal for many applications in which cost and space are critical.

In many applications, for example a TV remote control, there is no need for the computing power of a 486 or even an 8086 microprocessor. These applications most often require some I/O operations to read signals and turn on and off certain bits

1.5 Microcontrollers For Embedded Systems

In the Literature discussing microprocessors, we often see the term Embedded System. Microprocessors and Microcontrollers are widely used in embedded system products. An embedded system product uses a microprocessor (or Microcontroller) to do

one task only. A printer is an example of embedded system since the processor inside it performs one task only; namely getting the data and printing it. Contrast this with a Pentium based PC. A PC can be used for any number of applications such as word processor, print-server, bank teller terminal, Video game, network server, or Internet terminal. Software for a variety of applications can be loaded and run. Of course, the reason a pc can perform myriad tasks is that it has RAM memory and an operating system that loads the application software into RAM memory and lets the CPU run it.

In this robot as the fire sensor senses the fire, it senses the signal to microcontroller. In an Embedded system, there is only one application software that is typically burned into ROM. An x86 PC contains or is connected to various embedded products such as keyboard, printer, modem, disk controller, sound card, CD-ROM drives, mouse, and so on. Each one of these peripherals has a Microcontroller inside it that performs only one task.

CHAPTER 2

PROJECT OVERVIEW

2.1 Introduction to Project:

A. Background: In this modern world, pathfinding algorithms play a vital role in robotics and automation. However, commonly used pathfinding algorithms for ultrasonic sensors are limited by the monodirectional nature of the sensor. They often rely on simple binary directional decisions (left and right), which is inaccurate [1]. This paper aims to enhance the existing algorithms. These algorithms are crucial for finding paths and avoiding obstacles that come the car's way. They have various ap plications in the military to reach inaccessible or dangerous areas. Moreover, they have applications in obstacle avoidance systems on extraterrestrial surfaces with an atmosphere. Suresh Kumar Gawre Assistant Professor Department of Electrical engineering MANIT Bhopal sgawre28@gmail.com Fig. 1. Circuit Diagram

B. Motivation The motivation behind this project stems from the limita tions of the monodirectional nature of the ultrasonic sensor. This does not provide an accurate way of finding the ideal path as it gives false positives (detailed in the Algorithm section). While other wide-angle methods exist, they tend to be expensive and complicated. This motivated us to create a composite sensor using an ultrasonic sensor mounted on a servo motor. We formulated an algorithm for omnidirectional pathfinding and obstacle detection alongside this sensor.

C. Paper Organization In the following section, we'll give a basic overview of our project, Section II tells us about the design and hardware requirements. Subsequently, Section III tells us about the algorithm and its working principle. Section IV will describe the software requirement and control mechanism, emphasizing their important role. To conclude our discussion, Section V tells us about the advantages, real-world applications, and future potential of the project, meanwhile, Section VI will provide a detailed summary of our discoveries and inputs. Refer to Figures 12 and 13, showing the block diagram and the final model, respectively. Our overall methodology revolves around omni directional pathfinding using a mean-based algorithm, utilizing a single ultrasonic sensor

2.1.1 Existing System:

Conventional robotic navigation systems rely on static ultrasonic sensors that provide monodirectional obstacle detection, limiting their effectiveness in complex environments. These systems suffer from blind spots and inaccurate obstacle readings due to signal reflections, leading to frequent path recalculations and inefficient movement. Additionally, traditional robotic platforms lack servo motor-driven scanning, preventing adaptive obstacle detection. The absence of a motor control mechanism like the L293D driver results in imprecise movement, causing navigation inefficiencies and suboptimal path execution. The limitations of these traditional systems highlight the need for a more advanced robotic navigation approach that integrates dynamic obstacle detection and precise motor control.

2.1.2 Proposed System:

The proposed system represents a significant advancement in robotic navigation by integrating three critical components: an ultrasonic sensor, a servo motor, and an L293D motor driver. The ultrasonic sensor is intelligently mounted on a servo motor, enabling it to scan a wide **180-degree field of view**, which provides enhanced obstacle detection capabilities. This setup allows the system to gather detailed spatial information about its surroundings, ensuring a comprehensive understanding of potential obstacles.

The collected data is processed through a **mean-based algorithm**, a method designed to improve the efficiency and accuracy of path selection. This algorithm minimizes false positives, ensuring reliable and precise navigation even in complex or dynamic environments.

The L293D motor driver serves as the backbone for controlling the robotic vehicle's movement. It enables **precise and responsive control** of the motors, allowing smooth transitions and seamless execution of calculated paths. This level of control enhances the overall stability and efficiency of the robot, making it capable of performing intricate maneuvers with ease.

These features make the system highly suitable for a variety of applications, including **industrial automation, search-and-rescue missions, and autonomous robotic systems**. Its versatility and reliability ensure it can excel in both structured and unstructured environments, addressing a wide range of real-world challenges.

2.2 Block Diagram:

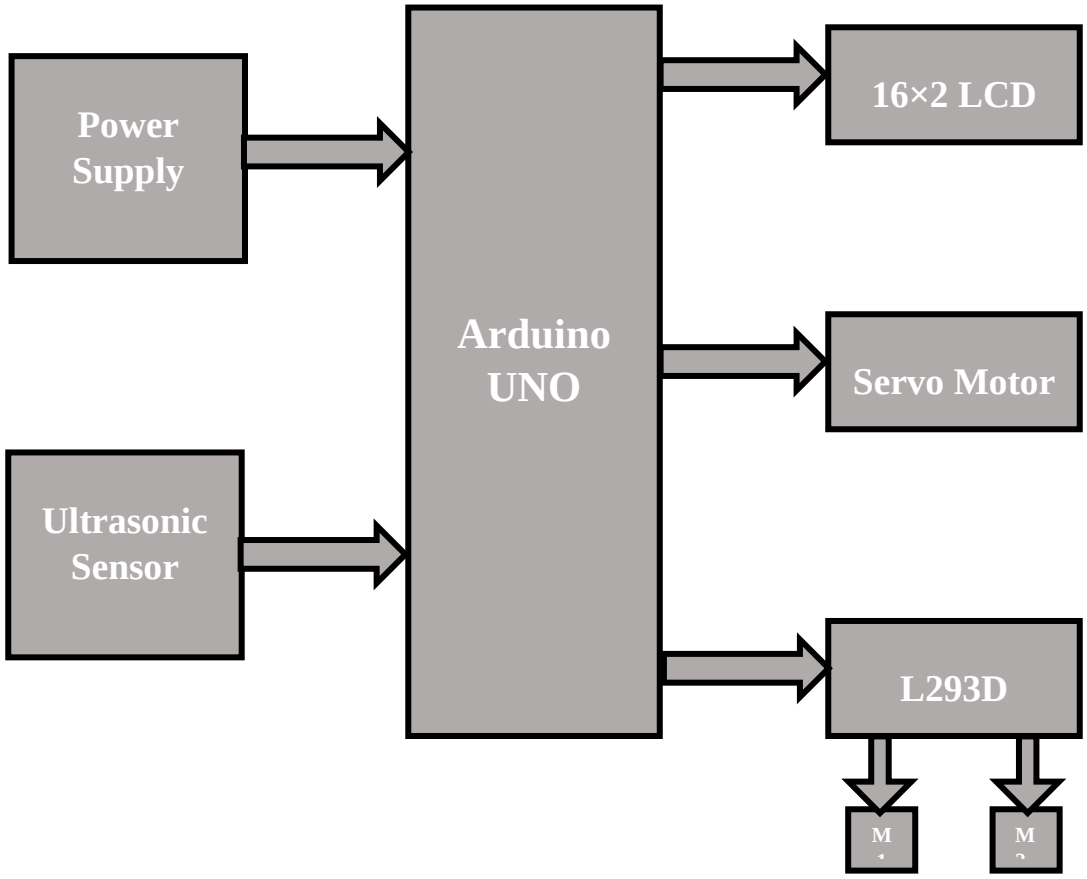

Fig 0.1: Block diagram of Omni-Directional Robo Using Single Ultrasonic Sensor

CHAPTER – 3

HARDWARE DESCRIPTION

3.1 Arduino UNO

3.1.1 Introduction to Microcontroller

Microcontroller as the name suggest, a small controller. They are like single chip computers that are often embedded into other systems to function as processing/controlling unit. For example, the control you are using probably has microcontrollers inside that do decoding and other controlling functions. They are also used in automobiles, washing machines, microwaves ovens, toys, etc., where automation is needed.

3.1.2 Arduino UNO Microcontroller:

The Arduino Uno is a microcontroller board based on the Atmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, 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 Atmega8U2 programmed as a USB-to-serial converter. “Uno” means “One” in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board’s power jack. Leads

from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5Vpin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.·
- **5V. The** regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3.3V.**A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins

3.1.2.1 Memory:

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library)..

3.1.2.2 Input and Output:

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the Atmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, arising or falling edge, or a change in value. See the attach Interrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which although provided by the underlying hardware, is not currently included in the Arduino language.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:
- **I2C: 4 (SDA) and 5 (SCL).** Support I2C (TWI) communication using the Wire library.

  There are a couple of other pins on the board:
- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

3.1.2.3 Communication:

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The Atmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An Atmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard

USBCOM drivers, and no external driver is needed. However, on Windows, an *.inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A Software Serial library allows for serial communication on any of the Uno's digital pins. The Atmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus

3.1.3 Arduino UNO Board:

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 converters

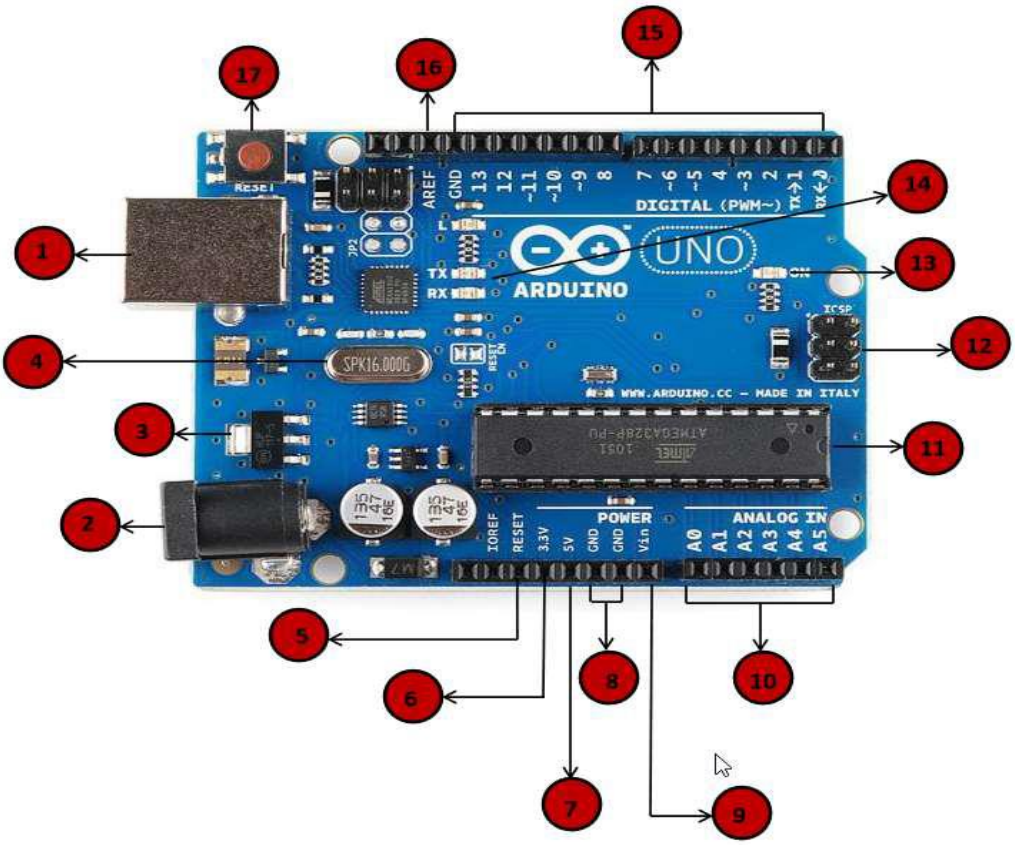

Fig 1.2: Arduino Uno board

3.1.3.1 Technical Specifications:

Table 1.1: Arduino UNO specifications

Features	Specifications
Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V

Digital I/O Pins	14(of which 6 provide PWM output)
Analog Input Pins	6
DC Current pere I/O pin	40 mA
DC Current for 3.3v pin	50 mA
Flash Memory	32 KB(ATmega328) of which 0.5KB used by boot loader
SRAM	2 KB(ATmega328)
EEPROM	1 KB(ATmega328)
Clock Speed	16 MHz

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

- **USB Interface:**

  Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection

- **External power supply:**

  Arduino boards can be powered directly from the AC mains power supply by connecting it to the power supply (Barrel Jack).

- **Voltage Regulator:**

  The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

- **Crystal Oscillator:**

The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz

- **Arduino Reset:**

  It can reset your Arduino board, i.e., start your program from the beginning. It can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).

- **Pins (3.3, 5, GND, Vin):**
  - 3.3V (6): Supply 3.3 output volt
  - 5V (7): Supply 5 output volt
  - Most of the components used with Arduino board works fine with 3.3 volt and 5 volt.
  - GND (8)(Ground): There are several GND pins on the Arduino, any of which can be used to ground your circuit.
  - Vin (9): This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.
- **Analog pins:**

  The Arduino UNO board has five analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.

- **Main microcontroller:**
- Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

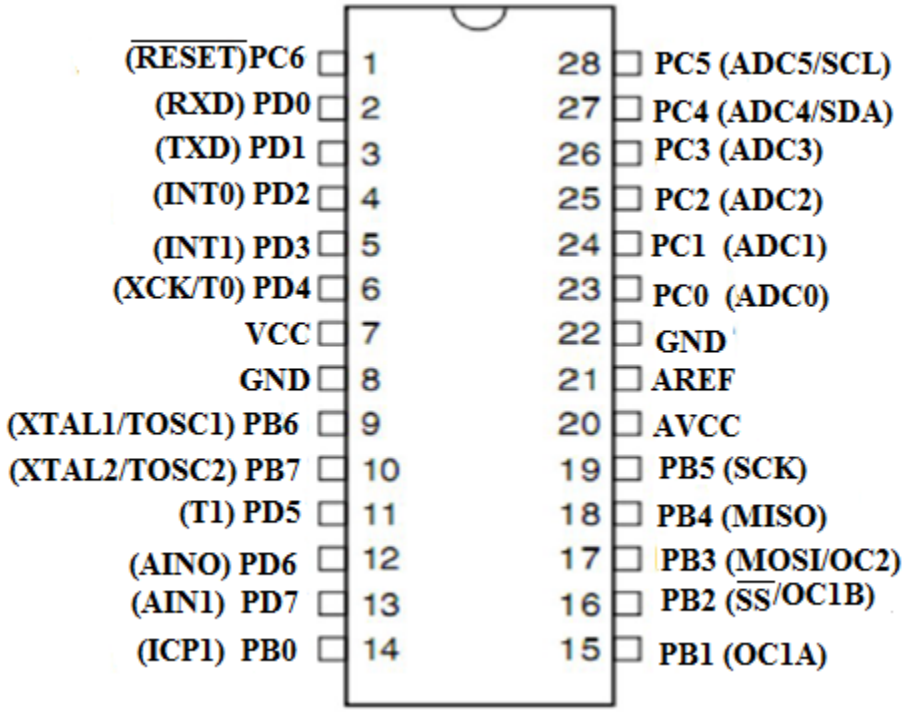

Fig 1.3: Pin diagram

The Atmega8U2 programmed as a USB-to-serial converter. “Uno” means “One” in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards

3.1.3.2 Pin Description:

- **VCC:** Digital Supply Voltage.
- **GND:** Ground.
- **Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2:** Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port

B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

- **Port C (PC[5:0]):** Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability.

- **PC6/RESET:** If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

- **Port D (PD[7:0]):** Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

- **AVCC:** AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC[6:4] use digital supply voltage, VCC.
- **ADC [7:6] (TQFP and VFQFN Package Only):** In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

- **12. ICSP pin:** Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus

- **15. Digital I / O:** The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled "~" can be used to generate PWM.

- **AREF:** AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins working.

3.2 Power Supply:

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

This power supply section is required to convert AC signal to DC signal and also to reduce the amplitude of the signal. The available voltage signal from the mains is 230V/50Hz which is an AC voltage, but the required is DC voltage (no frequency) with the amplitude of +5V and +12V for various applications.

In this section we have the Transformer, Bridge rectifier, are connected serially and voltage regulators for +5V and +12V (7805 and 7812) via a capacitor (1000uF) in parallel are connected parallel as shown in the circuit diagram below.

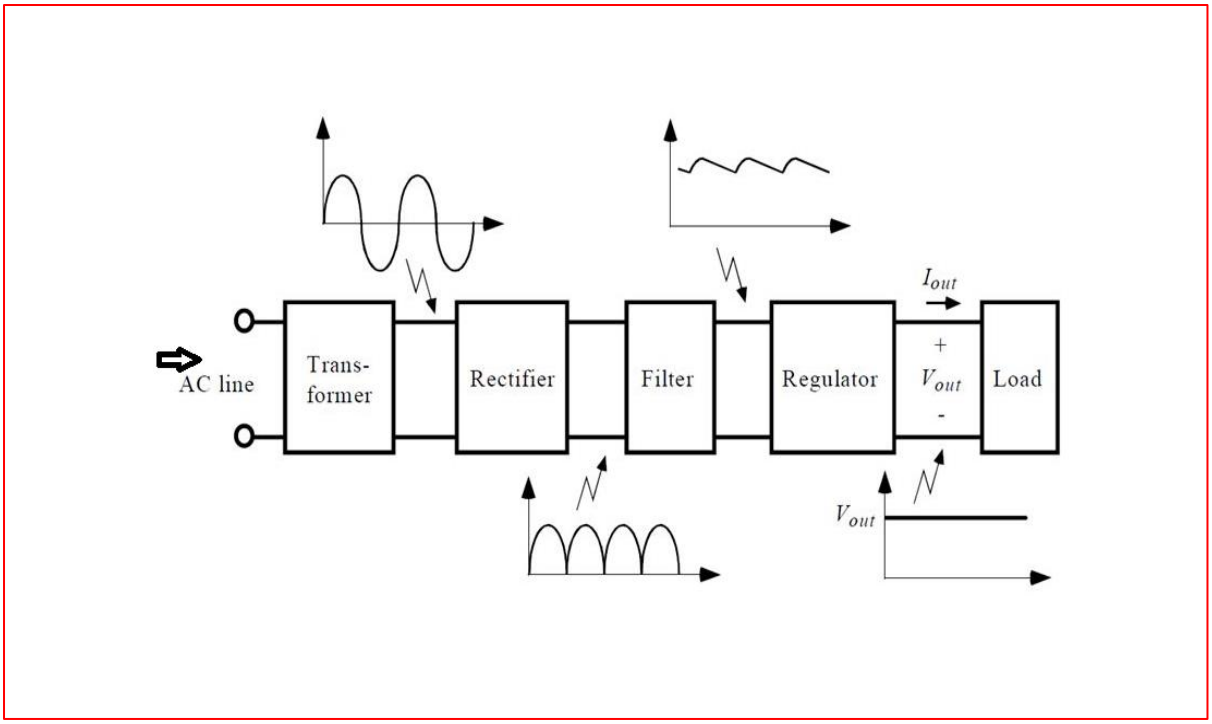

Fig 1.4: Power Supply

3.2.1 Transfomer:

Transformer is a static device used to convert the voltage from one level to another level without change its frequency. There are two types of transformers

1. Step-up transformer
2. Step-down transformer

Step-up transformer converts low voltage level into high voltage level without change its frequency.

Step-down transformer converts high voltage level into low voltage level without change its frequency.

In this project we using step-down transformer which converts 230V AC to 12V AC [or] 230V AC to 5V as shown below

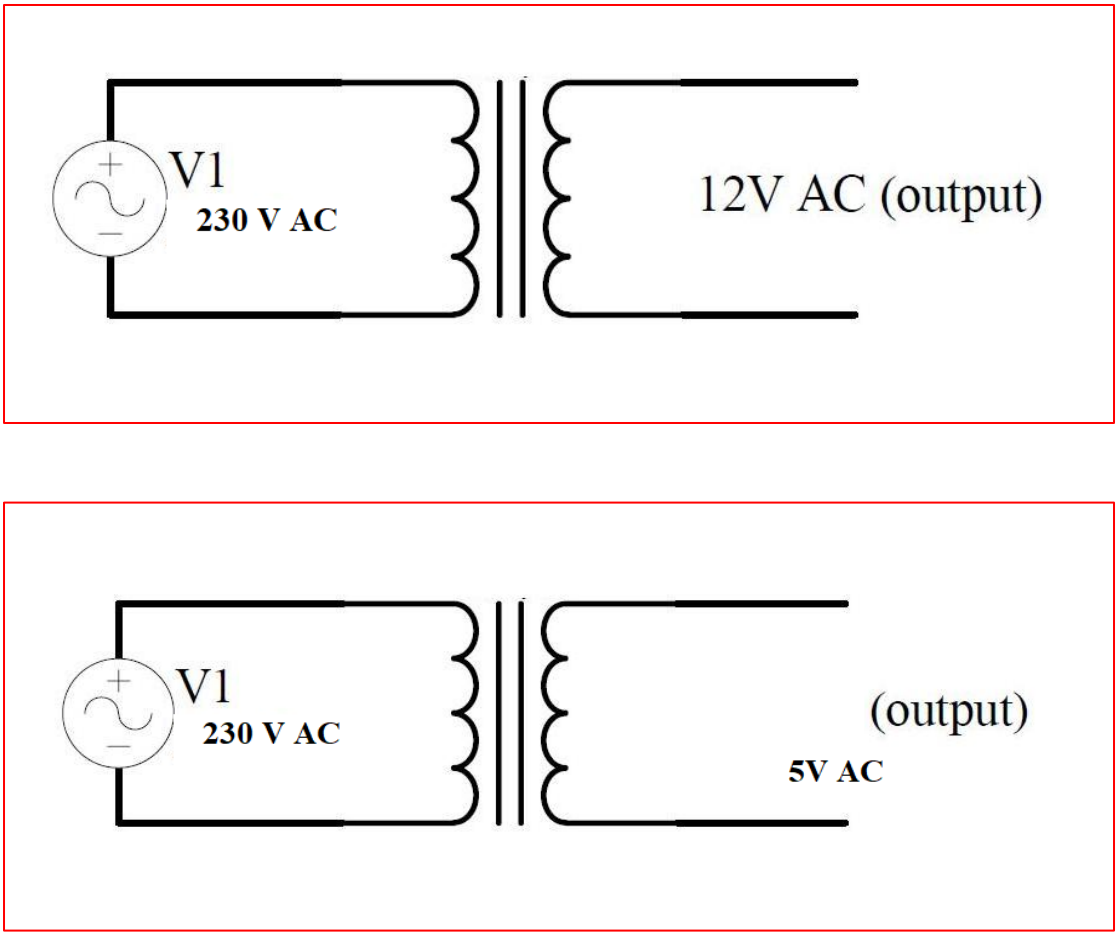

Fig 1.5: Transformers

3.2.2 Rectifier:

The purpose of a rectifier is to convert an AC waveform into a DC waveform (OR) Rectifier converts AC current or voltages into DC current or voltage. There are two different rectification circuits, known as **'half-wave'** and **'full-wave' rectifiers**. Both use components called **diodes** to convert **AC into DC**.

1. The Half-wave Rectifier

The half-wave rectifier is the simplest type of rectifier since it only uses one diode, as shown in figure.

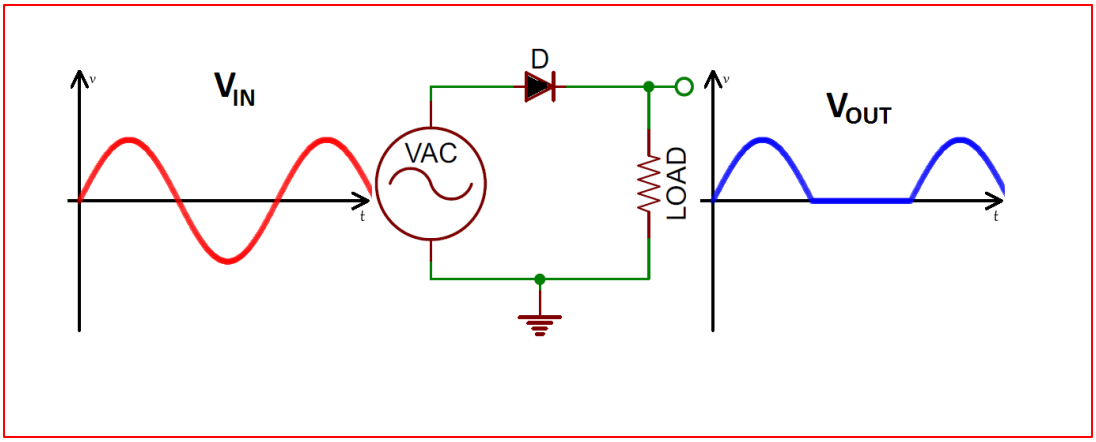

Fig 1.6: Half Wave Rectifier

Figure 3.6 shows the AC input waveform to this circuit and the resulting output. As you can see, when the AC input is positive, the diode is forward-biased and lets the current through. When the AC input is negative, the diode is reverse-biased and the diode does not let any current through, meaning the output is 0V. Because there is a 0.7V voltage loss across the diode, the peak output voltage will be 0.7V less than Vs.

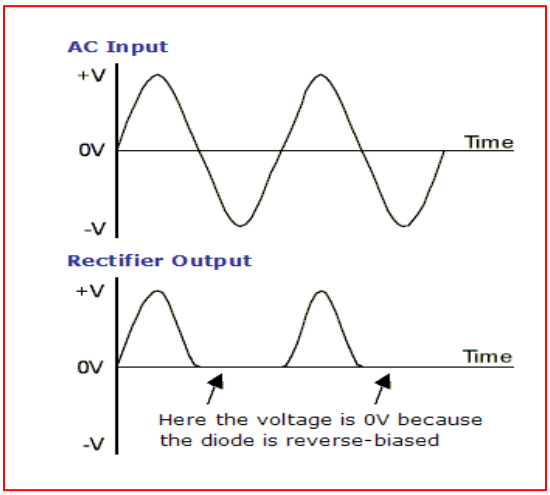

Fig 1.7: Half-Wave Rectification

While the output of the half-wave rectifier is DC (it is all positive), it would not be suitable as a power supply for a circuit. Firstly, the output voltage continually varies between 0V and Vs-0.7V, and secondly, for half the time there is no output at all.

The Full-wave Bridge Rectifier

The circuit in figure 3 addresses the second of these problems since at no time is the output voltage 0V. This time four diodes are arranged so that both the positive and negative parts of the AC waveform are converted to DC. The resulting waveform is shown in figure 4.

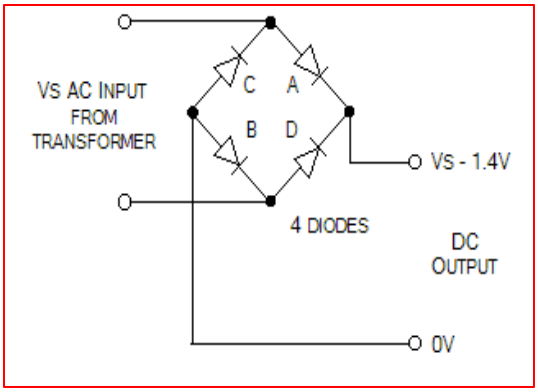

Fig 1.8: Full-Wave Rectifier

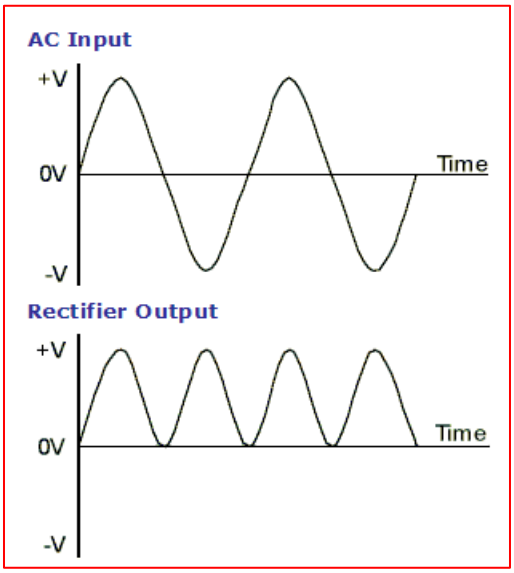

Fig 1.9: Full-Wave Rectification

When the AC input is positive, diodes A and B are forward-biased, while diodes C and D are reverse-biased. When the AC input is negative, the opposite is true – diodes C and D are forward-biased, while diodes A and B are reverse-biased.

While the full-wave rectifier is an improvement on the half-wave rectifier, its output still isn't suitable as a power supply for most circuits since the output voltage still varies between 0V and Vs-1.4V. So, if you put 12V AC in, you will 10.6V DC out.

3.2.3 Diodes

Diodes allow electricity to flow in only one direction. The arrow of the circuit symbol shows the direction in which the current can flow. Diodes are the electrical version of a valve and early diodes were actually called valves.

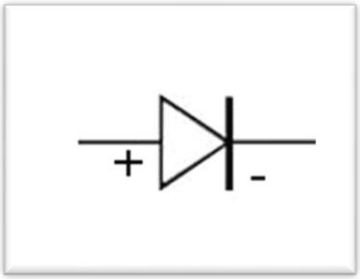

Fig 1.10: Diode Symbol

A **diode** is a device which only allows current to flow through it in one direction. In this direction, the diode is said to be 'forward-biased' and the only effect on the signal is that there will be a voltage loss of around 0.7V. In the opposite direction, the diode is said to be 'reverse-biased' and no current will flow through it.

3.2 LIQUID CRYSTAL DISPLAY

3.3.1 Introduction to LCD

A liquid crystal display (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. Each pixel consists of a column of liquid crystal molecules suspended between two transparent electrodes, and two polarizing filters, the axes of polarity of which are perpendicular to each other.

Without the liquid crystals between them, light passing through one would be blocked by the other. The liquid crystal twists the polarization of light entering one filter to allow it to pass through the other.

A program must interact with the outside world using input and output devices that communicate directly with a human being. One of the most common devices attached to an controller is an LCD display. Some of the most common LCDs connected to the controllers are 16X1, 16x2 and 20x2 displays. This means 16 characters per line by 1 line 16 characters per line by 2 lines and 20 characters per line by 2 lines, respectively.

Many microcontroller devices use 'smart LCD' displays to output visual information. LCD displays designed around LCD NT-C1611 module, are inexpensive, easy to use, and it is even possible to produce a readout using the 5X7 dots plus cursor of the display.

They have a standard ASCII set of characters and mathematical symbols. For an 8-bit data bus, the display requires a +5V supply plus 10 I/O lines (RS, RW, D7, D6, D5, D4, D3, D2, D1, D0).

For a 4-bit data bus it only requires the supply lines plus 6 extra lines (RS, RW, D7, D6, D5, D4). When the LCD display is not enabled, data lines are tri-state and they do not interfere with the operation of the microcontroller.

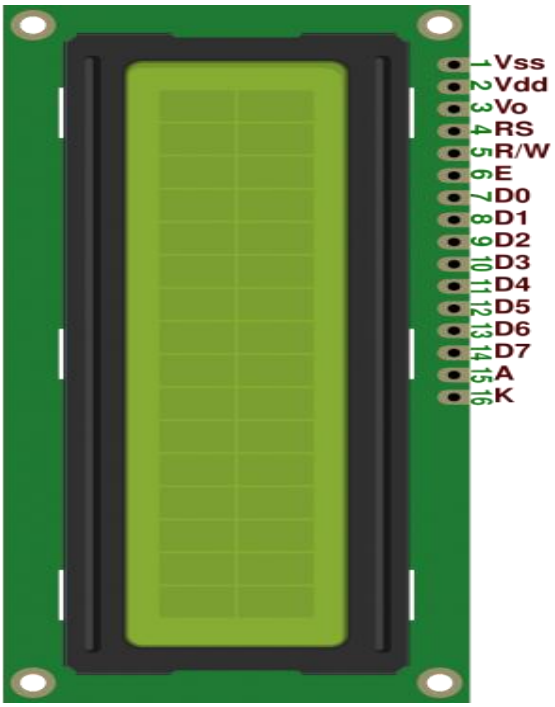

Fig 1.11: 16 X 2 LCD Display

3.3.2 General Description:

The LCD s used exclusively in watches, calculators and measuring instruments is the simple seven-segment displays, having a limited amount of numeric data. The recent advances in technology have resulted in better legibility, more information displaying capability and a wider temperature range. These have resulted in the LCD s being extensively used in telecommunications and entertainment electronics. The LCD s has even started replacing the cathode ray tubes (CRTs) used for the display of text and graphics, and also in small TV applications

3.3.3 LCD Pin Diagram:

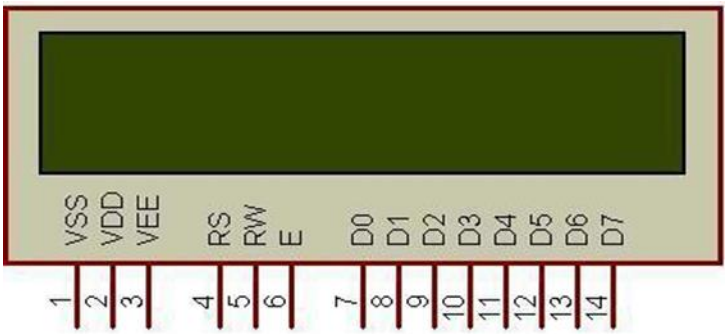

Fig 1.12: LCD Pin Diagram

Table 1.2: Pin Configuration & its Functions

PIN	SYMBOL	FUNCTION
1	VSS	Power Supply(GND)
2	VDD	Power Supply(+5V)
3	VO	Contrast Adjust
4	RS	Instruction/data register select
5	R/W	Data Bus Line
6	E	Enable Signal
7-14	DB0-DB7	Data Bus Line
15	A	Power supply for LED B/L(+)
16	K	Power Supply for LED B/L(-)

3.3.4 Control Lines:

EN:

Line is called “Enable.” This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring EN high (1) and wait for the minimum amount of time

required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

EN=0 => High Impedance

EN=1=> Low Impedance

RS:

Line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.).When RS is high (1), the data being sent is text data which should be displayed on the screen.

RS = 0 => Command

RS = 1 => Data

`

RW:

Line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands, so RW will almost always be low.

RW = 0 => 0 Writing data to LCD

RW = 1 => 1 Reading data from LCD

3.3.5 Contrast Control:

To have a clear view of the characters on the LCD, contrast should be adjusted. To adjust the contrast, the voltage should be varied. For this, a preset is used which can behave like a variable voltage device. As the voltage of this preset is varied, the contrast of the LCD can be adjusted.

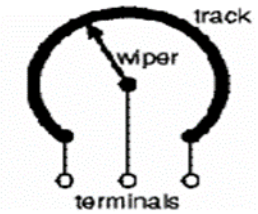

Fig 1.13: Variable Resistor

3.3.6 Potentiometer:

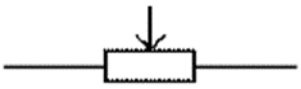

Fig 1.14: Potentiometer symbol

Variable resistors used as potentiometers have all **three terminals** connected. This arrangement is normally used to **vary voltage**, for example to set the switching point of a circuit with a sensor, or control the volume (loudness) in an amplifier circuit. If the terminals at the ends of the track are connected across the power supply, then the wiper terminal will provide a voltage which can be varied from zero up to the maximum of the supply.

3.3.7: Presets:

These are miniature versions of the standard variable resistor. They are designed to be mounted directly onto the circuit board and adjusted only when the circuit is built. For example, to set the frequency of an alarm tone or the sensitivity of a light-sensitive circuit, a small screwdriver or similar tool is required to adjust presets. Presets are much cheaper than standard variable resistors so they are sometimes used in projects where a standard variable resistor would normally be used.

Multiturn presets are used where very precise adjustments must be made. The screw must be turned many times (10+) to move the slider from one end of the track to the other, giving very fine control.

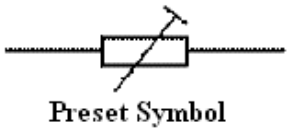

Fig 1.15: Preset symbol

3.4 Buzzer:

A buzzer is a small yet efficient component to add sound features to our project/system. It is very small and compact 2-pin structure hence can be easily used on breadboard, per board and even on PCBs which makes this a widely used component in most electronic applications. There are two types are buzzers that are commonly available. The one shown here is a simple buzzer which when powered will make a Continuous Beep sound, the other type is called a readymade buzzer which will look bulkier than this and will produce a Beep. Beep. Beep. Sound due tothe internal oscillating circuit present inside it. But, the one shown here is most widely used because it can be customized with help of other circuits to fit easily in our application. This buzzer can be used by simply powering it using a DC power supply ranging from 4V to 9V. A simple 9V battery can also be used, but it is recommended to use a regulated +5Vor +6V DC supply.

Magnetic Transducer

Magnetic transducers contain a magnetic circuit consisting of a iron core with a wound coil and a yoke plate, a permanent magnet and a vibrating diaphragm with a movable iron piece. The diaphragm is slightly pulled towards the top of the core by the magnet's magnetic field. When a positive AC signal is applied, the current flowing through the excitation coil produces a fluctuating magnetic field, which causes the diaphragm to vibrate up and down, thus vibrating air. Resonance amplifies vibration through resonator consisting of sound hole(s) and cavity and produces a loud sound.

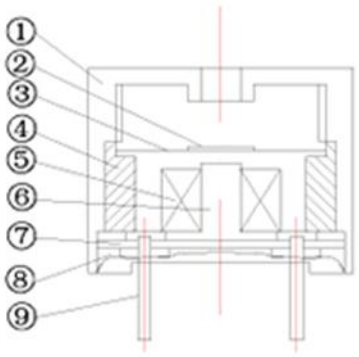

Fig 1.16: Magnetic Transducer

Magnetic Buzzer (Sounder)

Buzzers like the TMB-series are magnetic audible signal devices with built-in oscillating circuits. The construction combines an oscillation circuit unit with a detection coil, a drive coil and a magnetic transducer. Transistors, resistors, diodes and other small devices act as circuit devices for driving sound generators. With the application of voltage, current flows to the drive coil on primary side and to the detection coil on the secondary side. The amplification circuit, including the transistor and the feedback circuit, causes vibration. The oscillation current excites the coil and the unit generates an AC magnetic field corresponding to an oscillation frequency. This AC magnetic field magnetizes the yoke comprising the magnetic circuit. The oscillation from the intermittent magnetization prompts the vibration diaphragm to vibrate up and down, generating buzzer sounds through the resonator.

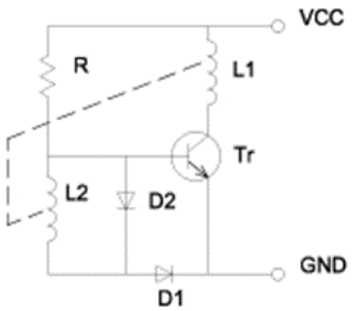

Fig 1.17; Recommended Driving Circuit for Magnetic Transducer

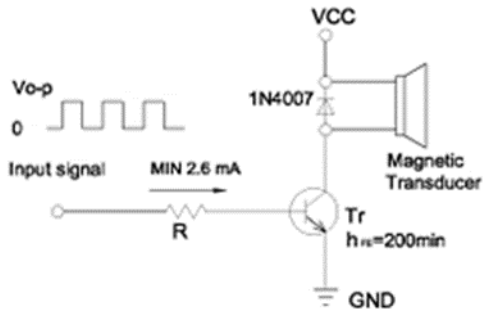

Fig 1.18: Introduction of Magnetic Buzzer (Transducer))

3.4.1 Pin Description:

- **Positive Pin**
  - Identified by (+) symbol or longer Terminal lead. Can be powered by 6 volts DC
- **Negative Pin**
  - Identified by (-) symbol or shorter Terminal lead. Typically connected to the ground of the circuit

3.4.2 Specification:

- **Rated Voltage:** 6V DC
- **Operating Voltage:** 4-8V DC
- **Rated Current**: <30mA.
- **Sound Type:** Continuous beep.
- **Resonant Frequency:** ~230 Hz
- Small and neat Sealed package
- Breadboard and Perf board friendly.

3.5 Relays:

A relay is an electrically controllable switch widely used in industrial controls, automobiles and appliances.

The relay allows the isolation of two separate sections of a system with two different voltage sources i.e., a small amount of voltage/current on one side can handle a

large amount of voltage/current on the other side but there is no chance that these two voltages mix up.

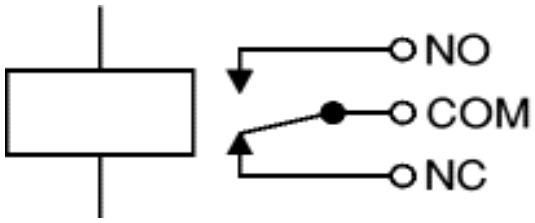

Fig 1.19: Circuit symbol of a relay

3.5.1 Operation:

When current flows through the coil, a magnetic field is created around the coil i.e., the coil is energized. This causes the armature to be attracted to the coil. The armature's contact acts like a switch and closes or opens the circuit. When the coil is not energized, a spring pulls the armature to its normal state of open or closed. There are all types of relays for all kinds of applications.

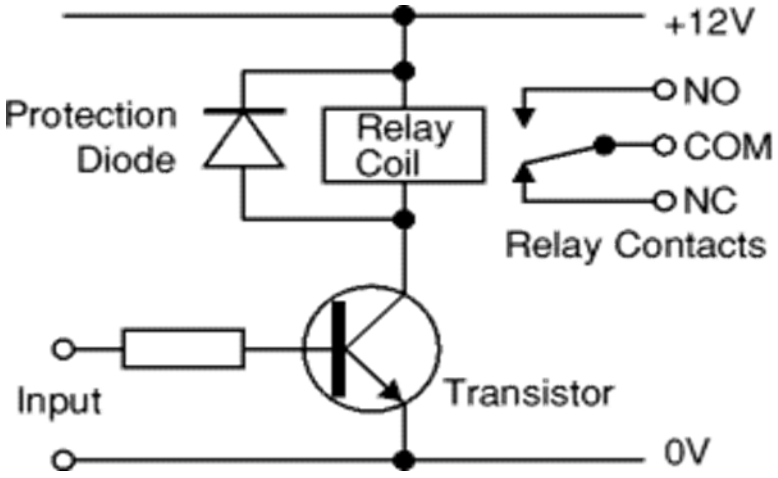

Fig 1.20: Relay Operation and use of Protection diodes

Transistors and ICs must be protected from the brief high voltage 'spike' produced when the relay coil is switched off. The above diagram shows how a signal diode (eg 1N4148) is connected across the relay coil to provide this protection. The diode is connected 'backwards' so that it will normally not conduct. Conduction occurs only when the relay coil is switched off, at this moment the current tries to flow continuously through the coil and it is safely diverted through the diode. Without the diode no current could flow

and the coil would produce a damaging high voltage 'spike' in its attempt to keep the current flowing.

In choosing a relay, the following characteristics need to be considered:

1. The contacts can be normally open (NO) or normally closed (NC). In the NC type, the contacts are closed when the coil is not energized. In the NO type, the contacts are closed when the coil is energized.

2. There can be one or more contacts. i.e., different types like SPST (single pole single throw), SPDT (single pole double throw) and DPDT (double pole double throw) relays.

3. The voltage and current required to energize the coil. The voltage can vary from a few volts to 50 volts, while the current can be from a few milliamps to 20milliamps. The relay has a minimum voltage, below which the coil will not be energized. This minimum voltage is called the “pull-in” voltage.

4. The minimum DC/AC voltage and current that can be handled by the contacts. This is in the range of a few volts to hundreds of volts, while the current can be from a few amps to 40A or more, depending on the relay.

3.5.2 Driving a Relay:

An SPDT relay consists of five pins, two for the magnetic coil, one as the common terminal and the last pins as normally connected pin and normally closed pin. When the current flows through this coil, the coil gets energized. Initially when the coil is not energized, there will be a connection between the common terminal and normally closed pin. But when the coil is energized, this connection breaks and a new connection between the common terminal and normally open pin will be established. Thus when there is an input from the microcontroller to the relay, the relay will be switched on. Thus when the relay is on, it can drive the loads connected between the common terminal and normally open pin. Therefore, the relay takes 5V from the microcontroller and drives the loads which consume high currents. Thus the relay acts as an isolation device.

Digital systems and microcontroller pins lack sufficient current to drive the relay. While the relay's coil needs around 10milli amps to be energized, the microcontroller's pin can provide a maximum of 1-2milli amps current. For this reason, a driver such as ULN2003 or a power transistor is placed in between the microcontroller and the relay. In order to operate more than one relay, ULN2003 can be connected between relay and microcontroller.

3.6 Ultrasonic Sensor:

The sensor is primarily intended to be used in security systems for detection of moving objects, but can be effectively involved in intelligent children's toys, automatic door opening devices, and sports training and contact-less-speed measurement equipment.

3.6.1 Introduction:

Modern security systems utilize various types of sensors to detect unauthorized object access attempts. The sensor collection includes infrared, microwave and ultrasound devices, which are intended to detect moving objects. Each type of sensor is characterized by its own advantages and drawbacks. Microwave sensors are effective in large apartments because microwaves pass through dielectric materials. But these sensors consist of expensive super-high frequency components and their radiation is unhealthy for living organisms.

Infrared sensors are characterized by high sensitivity, low cost and are widely used. But, these sensors can generate false alarm signals if heating systems are active or temperature change speed exceeds some threshold level. Moreover, infrared sensors appreciably lose sensitivity if small insects penetrate the sensor lens. Ultrasound motion detection sensors are characterized by small power consumption, suitable cost and high sensitivity. That it why this kind of sensor is commonly used in home, office and car security systems. Existing ultrasound sensors consist of multiple passive and active components and are relatively complicated for production and testing. Sensors often times require a laborious tuning process.

Fig 1.21: Ultrasonic Sensor

3.6.2 Basic Operation:

The ultrasound transmitter **TX** is emitting ultrasound waves into sensor ambient space continuously. These waves are reflecting from various objects and are reaching ultrasound receiver **RX**. There is a constant interference figure if no moving objects are in the placement

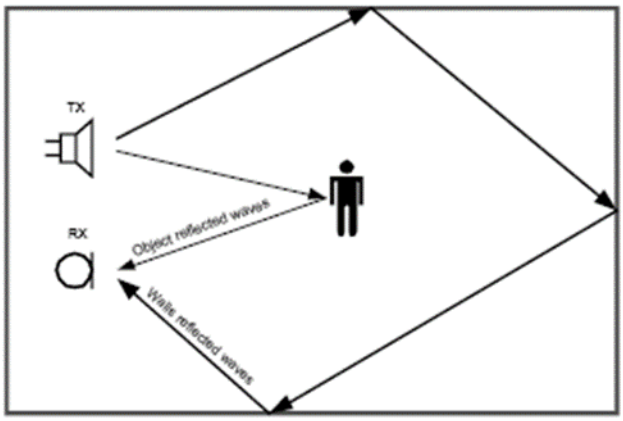

Fig 1.22: Basic Operation of Ultrasonic Sensor

Any moving object changes the level and phase of the reflected signal, which modifies the summed received signal level. Most low cost sensors (car security systems, for instance) perform reflected signal amplitude analysis to detect moving objects. In spite of implementation simplicity, this detection method is characterized by a high sensitivity to noise signals. For example, heterogeneous airflows, sensor vibrations, room window and door deformations, and gusts can change the interference figure and generate false alarm signals.

Better noise resistance may be obtained if the receive sensor is performing reflected signal frequency analysis instead of amplitude examination. The reflected signal spectrum emulates a Doppler Effect. Frequency components of the moving object speed vector have a component in the direction of ultrasound radiation propagation. Because ultrasound waves reflect from the windows, walls, furniture etc., the sensor can detect object movements in any direction. To implement this principle, the sensor must perform selection and processing of Doppler Effect frequency shift to detect moving objects.

The air condition systems, heat generators, and refrigerators typically include movable parts, which can cause device vibrations that generate high-frequency Doppler components in the reflected ultrasound signal. The heterogeneous variable temperature airflows are characterized by different ultrasound propagation speed that can raise low-frequency Doppler components in the reflected signal. That is why the noise resistant motion detection sensor should limit the Doppler signals' frequency range from lower and upper bounds to satisfactory false-alarm free operation.

3.7 Servo Motor

3.7.1 Introduction:

A **servo motor** is an electrical device which can push or rotate an object with great precision. If you want to rotate and object at some specific angles or distance, then you use servo motor. It is just made up of simple motor which run through **servo mechanism**. If motor is used is DC powered then it is called DC servo motor, and if it is AC powered motor then it is called AC servo motor. We can get a very high torque servo motor in a small and light weight packages. Doe to these features they are being used in many applications like toy car, RC helicopters and planes, Robotics, Machine etc.

Servo motors are rated in kg/cm (kilogram per centimeter) most hobby servo motors are rated at 3kg/cm or 6kg/cm or 12kg/cm. This kg/cm tells you how much weight your servo motor can lift at a particular distance. For example: A 6kg/cm Servo motor should be able to lift 6kg if the load is suspended 1cm away from the motors shaft, the greater the distance the lesser the weight carrying capacity.

The position of a servo motor is decided by electrical pulse and its circuitry is placed beside the motor.

Fig 1.23: Servo Motor

3.7.2 Servo Mechanism:

It consists of three parts:

1. Controlled device
2. Output sensor
3. Feedback system

It is a closed loop system where it uses positive feedback system to control motion and final position of the shaft. Here the device is controlled by a feedback signal generated by comparing output signal and reference input signal.

Here reference input signal is compared to reference output signal and the third signal is produces by feedback system. And this third signal acts as input signal to control device. This signal is present as long as feedback signal is generated or there is difference between reference input signal and reference output signal. So the main task of servomechanism is to maintain output of a system at desired value at presence of noises.

3.7.3 Working principle of Servo Motors

A servo consists of a Motor (DC or AC), a potentiometer, gear assembly and a controlling circuit. First of all we use gear assembly to reduce RPM and to increase torque of motor. Say at initial position of servo motor shaft, the position of the potentiometer knob is such that there is no electrical signal generated at the output port of the potentiometer. Now an electrical signal is given to another input terminal of the error detector amplifier. Now difference between these two signals, one comes from potentiometer and another comes from other source, will be processed in feedback mechanism and output will be provided in term of error signal. This error signal acts as the input for motor and motor starts rotating. Now motor shaft is connected with potentiometer and as motor rotates so the potentiometer and it will generate a signal. So as the potentiometer's angular position changes, its output feedback signal changes. After sometime the position of potentiometer reaches at a position that the output of potentiometer is same as external signal provided. At this condition, there will be no output signal from the amplifier to the motor input as there is no difference between external applied signal and the signal generated at potentiometer, and in this situation motor stops rotating.

3.7.4 Controls of Servo Motor:

All motors have three wires coming out of them. Out of which two will be used for Supply (positive and negative) and one will be used for the signal that is to be sent from the MCU.

Servo motor is controlled by PWM (Pulse with Modulation) which is provided by the control wires. There is a minimum pulse, a maximum pulse and a repetition rate. Servo motor can turn 90 degree from either direction form its neutral position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90° position, such as if pulse is shorter than 1.5ms shaft moves to 0° and if it is longer than 1.5ms than it will turn the servo to 180°.

Servo motor works on **PWM (Pulse width modulation)** principle, means its angle of rotation is controlled by the duration of applied pulse to its Control PIN. Basically servo

motor is made up of **DC motor which is controlled by a variable resistor (potentiometer) and some gears**. High speed force of DC motor is converted into torque by Gears. We know that WORK= FORCE X DISTANCE, in DC motor Force is less and distance (speed) is high and in Servo, force is High and distance is less. Potentiometer is connected to the output shaft of the Servo, to calculate the angle and stop the DC motor on required angle.

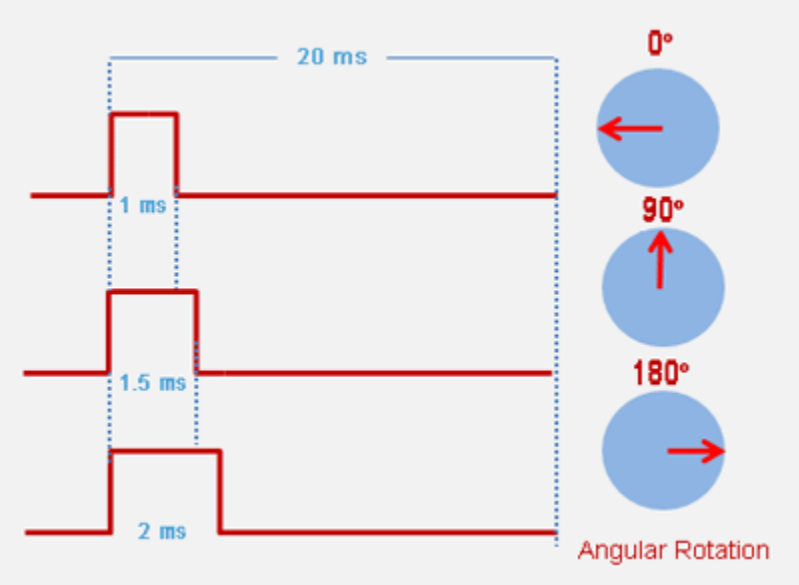

Fig 1.24: Servo Motor Position Control

Servo motor can be rotated from 0 to 180 degree, but it can go up to 210 degree, depending on the manufacturing. This degree of rotation can be controlled by applying the **Electrical Pulse** of proper width, to its Control pin. Servo checks the pulse in every 20 milliseconds. Pulse of 1 ms (1 millisecond) width can rotate servo to 0 degree, 1.5ms can rotate to 90 degree (neutral position) and 2 ms pulse can rotate it to 180 degree.

All servo motors work directly with your +5V supply rails but we have to be careful on the amount of current the motor would consume, if you are planning to use more than two servo motors a proper servo shield should be designed.

3.8 L293D Motor Driver:

3.8.1 Introduction:

The L293D motor driver is a popular integrated circuit used for controlling motors in electronic projects. It acts as a bridge between a microcontroller, such as an Arduino or Raspberry Pi, and DC motors, enabling the microcontroller to control the speed and direction of the motors. With its dual H-bridge configuration, the L293D allows for bidirectional control of up to two DC motors or unidirectional control of a single stepper motor. This feature makes it a versatile and essential component for robotics and automation applications.

One of the key features of the L293D is its ability to handle voltages between 4.5V and 36V, making it compatible with a wide range of motors. It also has built-in diodes to protect against back EMF (electromotive force), which occurs when the motor's magnetic field collapses, potentially damaging the circuit. The chip's compact design and ease of use have made it a favorite among hobbyists and engineers alike, providing a reliable solution for motor control in various DIY and professional projects.

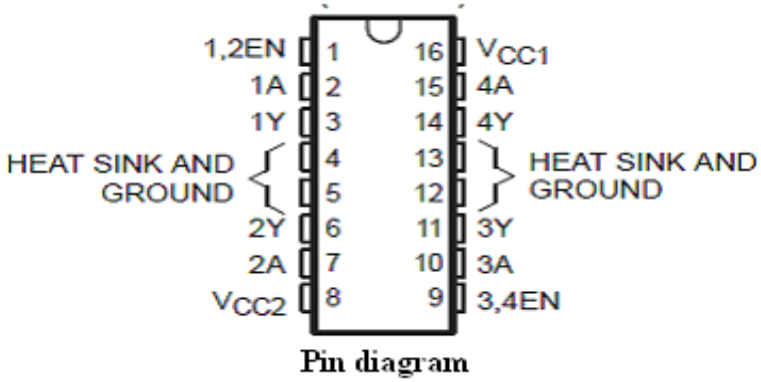

Fig 1.25: L293D Motor Driver

3.8.2 Features:

- Wide Supply-Voltage Range 4.5 V to 36V
- Separate Input-Logic Supply

- Internal ESD Protection
- Thermal Shutdown
- High-Noise-Immunity Inputs
- Functionally Similar to SGS L293 and SGS L293D
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

3.8.3 Description:

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo- Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications. On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.

A VCC1 terminal, separate from VCC2, is provided for the logic inputs to minimize device power dissipation. The L293 and L293D are characterized for operation from 0 to 70 degree Celsius.

Table 1.3: Function Table L293D Motor Driver

INPUT		OUTPUTS
A	EN	Y
H	H	H
L	H	L
X	L	Z

Block Diagram

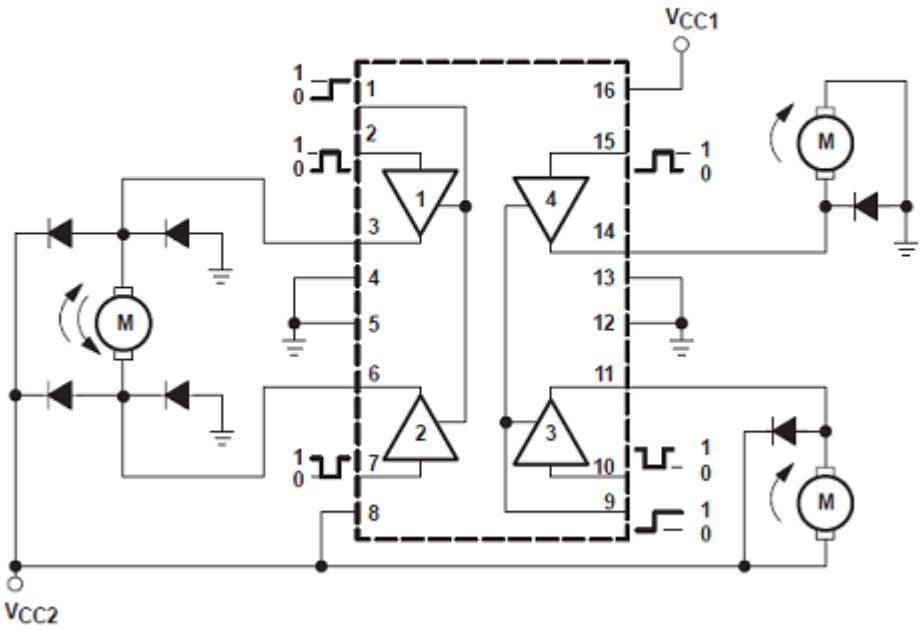

Fig 1.26: Block diagram of L293D Motor Driver

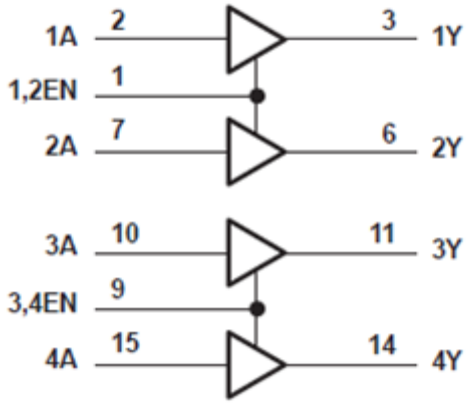

Fig 1.27: Logic Diagram

This chip contains 4 enable pins. Each enable pin corresponds to 2 inputs. Based on the input values given, the device connected to this IC works accordingly.

3.8.4 L293D Interfacing with 8051:

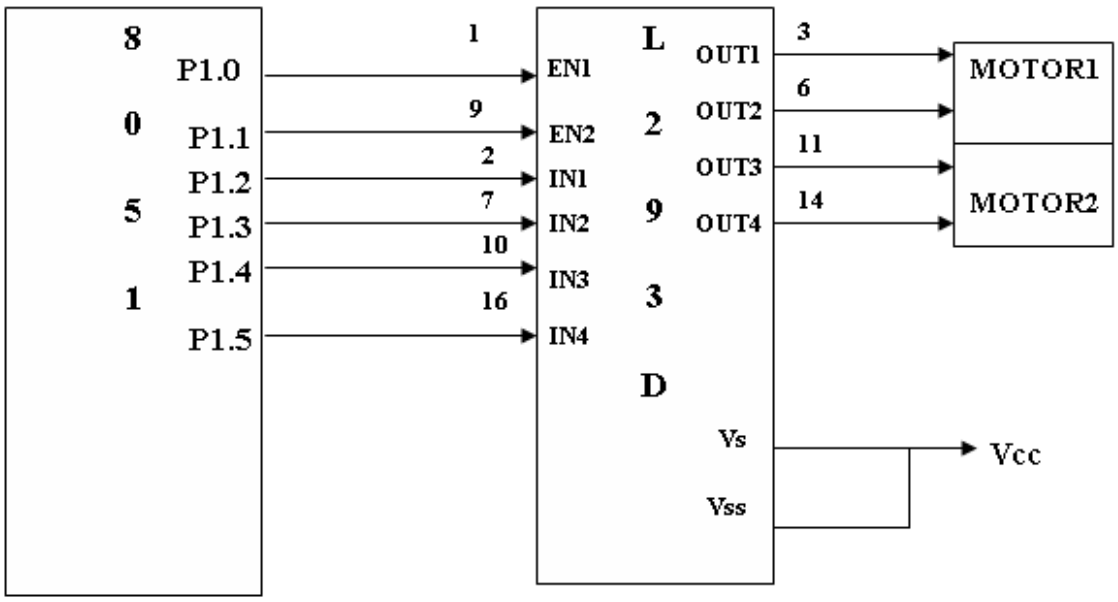

Fig 1.28: L293D Interfacing with 8051

This setup demonstrates how the L293D motor driver IC enables bidirectional control of two DC motors using a microcontroller. The enable pins (EN1 and EN2) activate the respective motor channels, while the input pins (IN1 to IN4) determine the direction of motor rotation. The L293D's built-in H-bridge configuration allows for seamless control of motor direction and speed, making it a versatile choice for robotics and automation projects. Its ability to handle back EMF through integrated diodes ensures circuit protection, further enhancing its reliability in motor control applications.

3.8.5 Control pins:

The DC motor description is carried out in the next section. The L293D output pins will be connected to the two motors of Robot. Thus, the output of L293D depends on the input provided from the microcontroller and the enable pins. It should be remembered that unless the enable pins are not high, whatever input values given to L293D IC will not be applied to the motors in any way

Left Wheel	Right Wheel	Movement
Forward	Forward	Forward
Backward	Backward	Backward
Forward	Stop	Right Turn
Stop	Forward	Left Turn
Forward	Backward	Sharp Right Turn
Backward	Forward	Sharp Left Turn

Table 1.4: Robot Controlling pin Truth table

CHAPTER – 4

SOFTWARES

4.1 Introduction to Arduino IDE

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

4.1.1 The key features are:

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

4.1.2 Arduino data types:

- Data types in C refers to an extensive system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.
- The following table provides all the data types that you will use during Arduino programming.

- **Void:**

  The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

  Example:

  ```
  Void Loop ()
  {
  // rest of the code
  }
  ```

- **Boolean:**

  A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte of memory.

  Example:

  Boolean state= false; // declaration of variable with type Boolean and initialize it with false.

  Boolean state = true; // declaration of variable with type Boolean and initialize it with false.

- **Char:**

  A data type that takes up one byte of memory that stores a character value. Character literals are written in single quotes like this: 'A' and for multiple

characters, strings use double quotes: "ABC". However, characters are stored as numbers. You can see the specific encoding in the ASCII chart.

This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used.

For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

Example:

- Char chr_a = ‘a’ ;//declaration of variable with type char and initialize it with character a.
- Char chr_c = 97 ;//declaration of variable with type char and initialize it with character 97

- **Unsigned char:**

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

Example:

Unsigned Char chr_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y.

- **Byte:**

A byte stores an 8-bit unsigned number, from 0 to 255.

Example:

byte m = 25 ;//declaration of variable with type byte and initialize it with 25

- **Int:**

Integers are the primary data-type for number storage. **int** stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

The **int** size varies from board to board. On the Arduino Due, for example, an **int** stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

Example:

int counter = 32 ;// declaration of variable with type int and initialize it with 32.

- **Unsigned Int:**

Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32 - 1).

Example:

Unsigned int counter= 60 ; // declaration of variable with type unsigned int and initialize it with 60.

- **Word:**

On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

Example:

word w = 1000 ;//declaration of variable with type word and initialize it with 1000.

- **Long:**

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from 2,147,483,648 to 2,147,483,647.

Example:

Long velocity= 102346 ;//declaration of variable with type Long and initialize it with 102346

- **Unsigned Long:**

  Unsigned long variables are extended size variables for number storage and store 32 bits (4 bytes). Unlike standard longs, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

  Example:

  Unsigned Long velocity = 101006 ;// declaration of variable with type Unsigned Long and initialize it with 101006.

- **Short:**

  A short is a 16-bit data-type. On all Arduinos (ATMega and ARM based), a short store a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

  Example:

  short val= 13 ;//declaration of variable with type short and initialize it with 13

- **Float:**

  Data type for floating-point number is a number that has a decimal point. Floating-point numbers are often used to approximate the analog and continuous values because they have greater resolution than integers.

  Floating-point numbers can be as large as 3.4028235E+38 and as low as 3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

  Example:

  float num = 1.352;//declaration of variable with type float and initialize it with 1.352.

- **Double:**

On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes. That is, the double implementation is exactly the same as the float, with no gain in precision. On the Arduino Due, doubles have 8-byte (64 bit) precision.

Example:

double num = 45.352 ;// declaration of variable with type double and initialize it with 45.352.

4.1.3 Arduino IDE and Board Setup

In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

- **Step 1: Board Selection & its Cable:**

  First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

Fig 1.29: USB Cable

- **Step 2: Download Arduino IDE Software**

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.

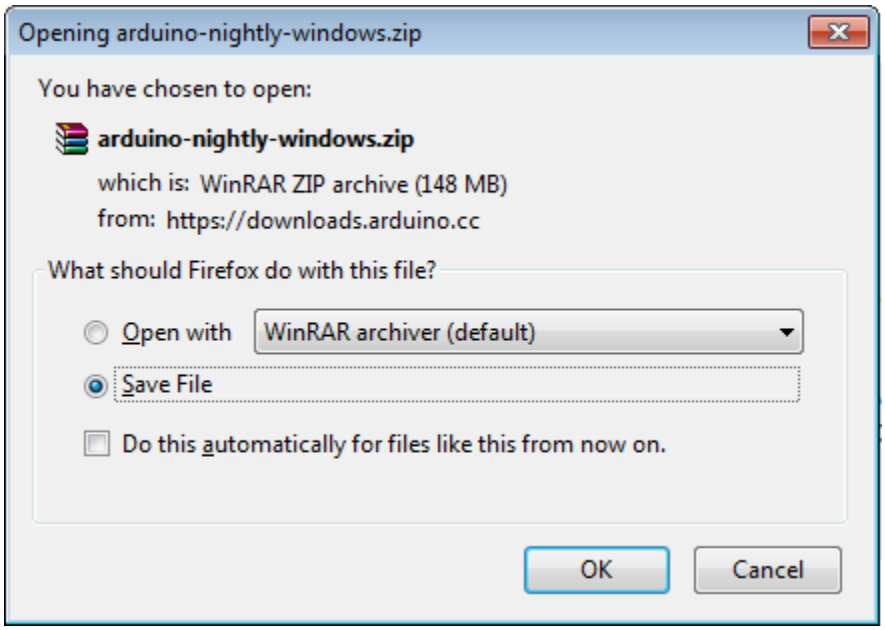

Fig 1.30: Downloading Arduino UNO Software

- **Step 3: Power up your board**

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port. Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

- **Step 4: Launch Arduino IDE.**

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double click the icon to start the IDE.

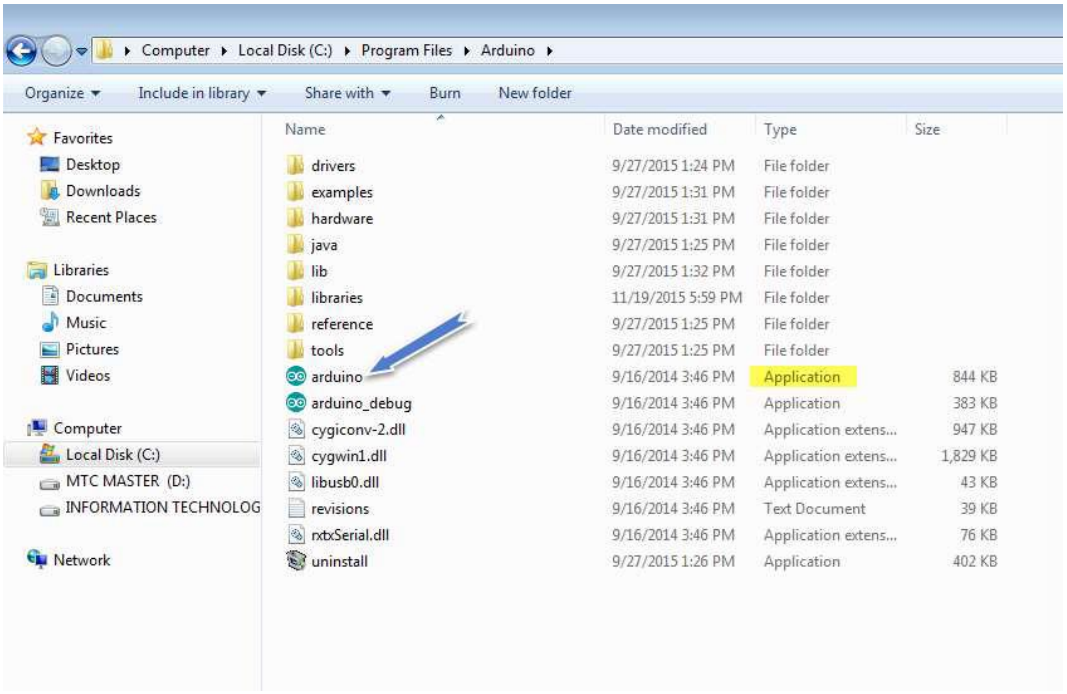

Fig 1.31: Launching Arduino IDE

- **Step 5: Open your first project.**

Once the software starts, you have two options:

- Create a new project.
- Open an existing project example.

To create a new project, select File --> New. To open

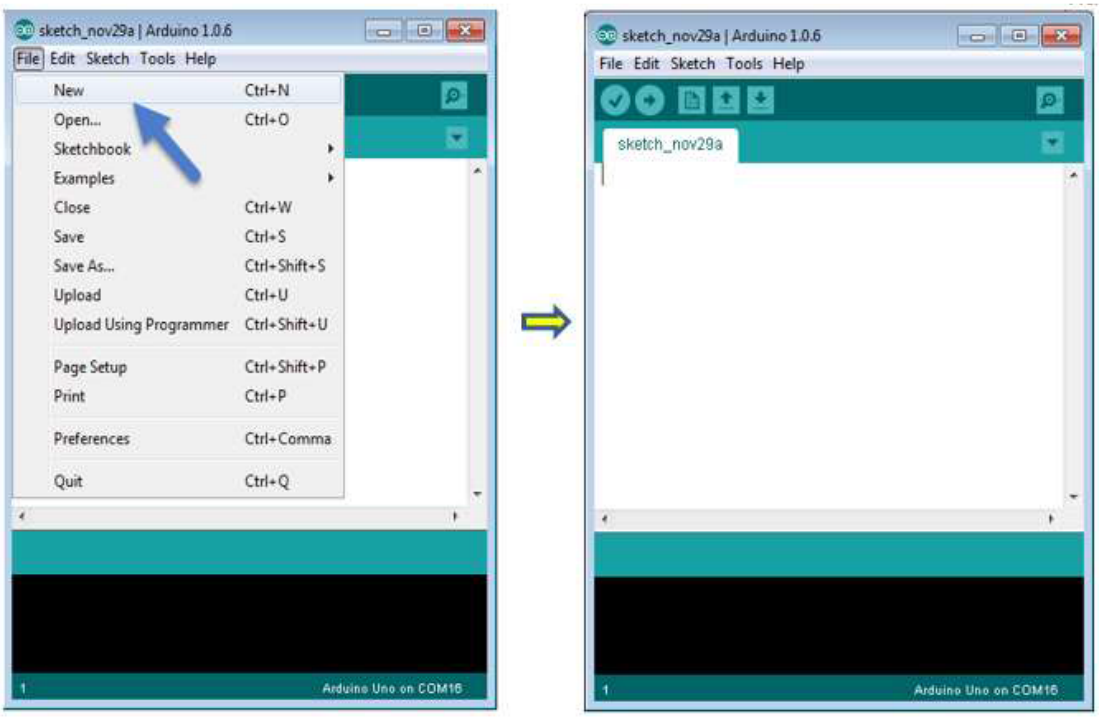

Fig 1.32: Opening New File

To open an existing project example, select File -> Example -> Basics -> Blink.

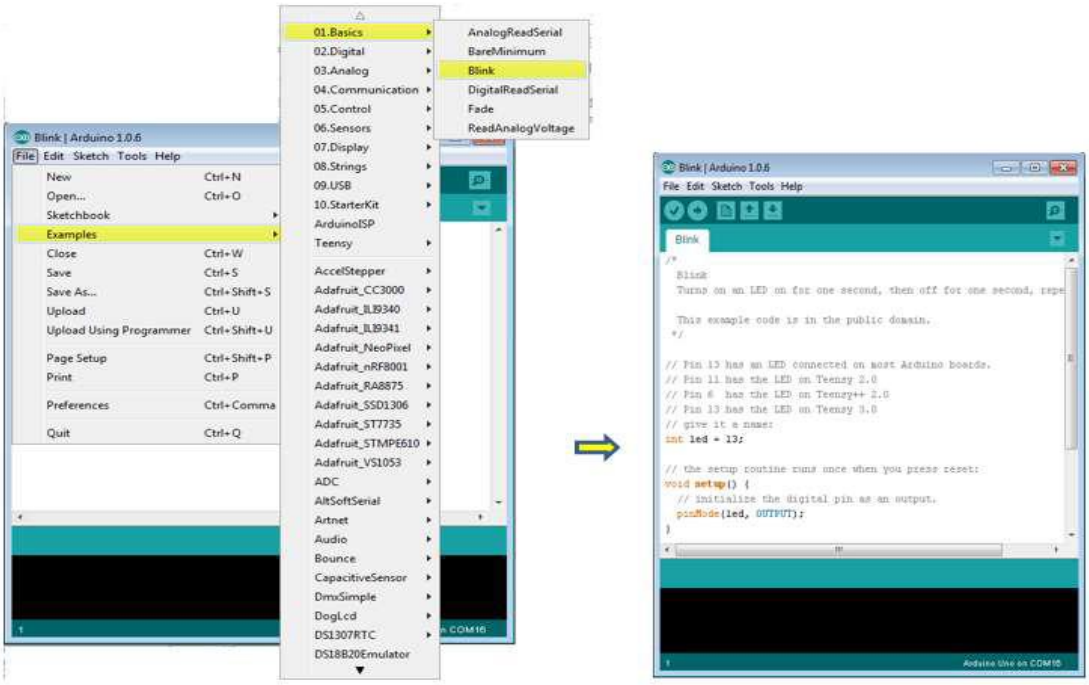

Fig 1.33: Opening Existing File

Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

- **Step 6: Select your Arduino board.**

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

Go to Tools -> Board and select your board

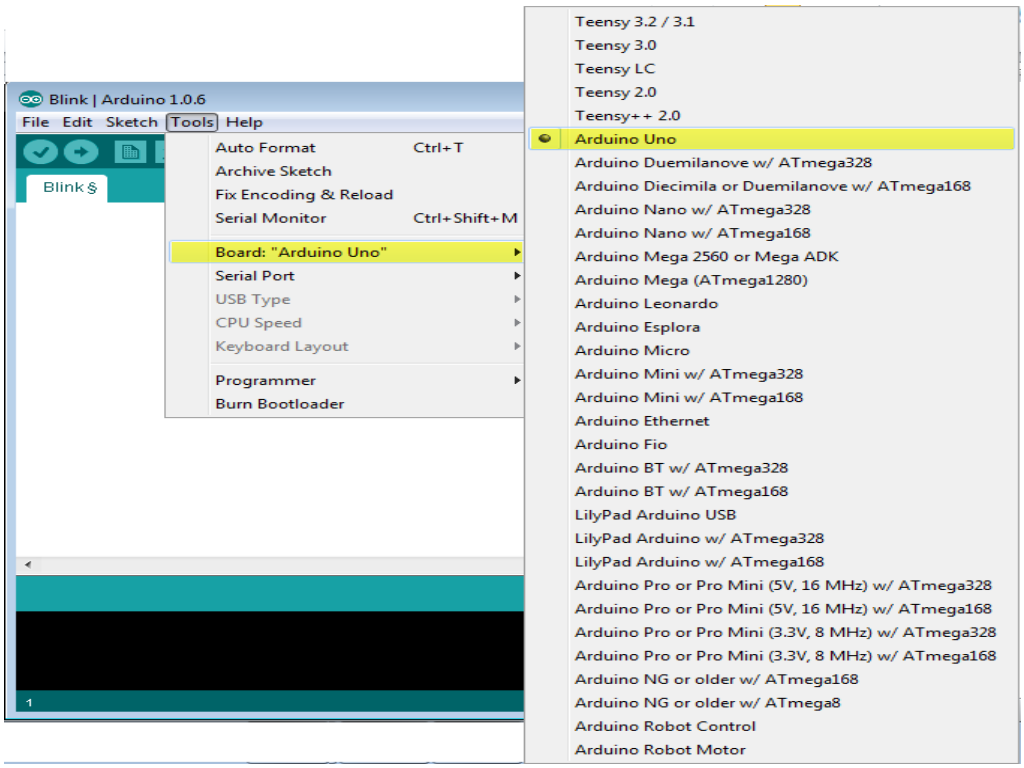

Fig 1.34: Selecting Arduino Board

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using

- **Step 7: Select your Serial board**

Select the serial device of the Arduino board. Go to **Tools** ->**Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.

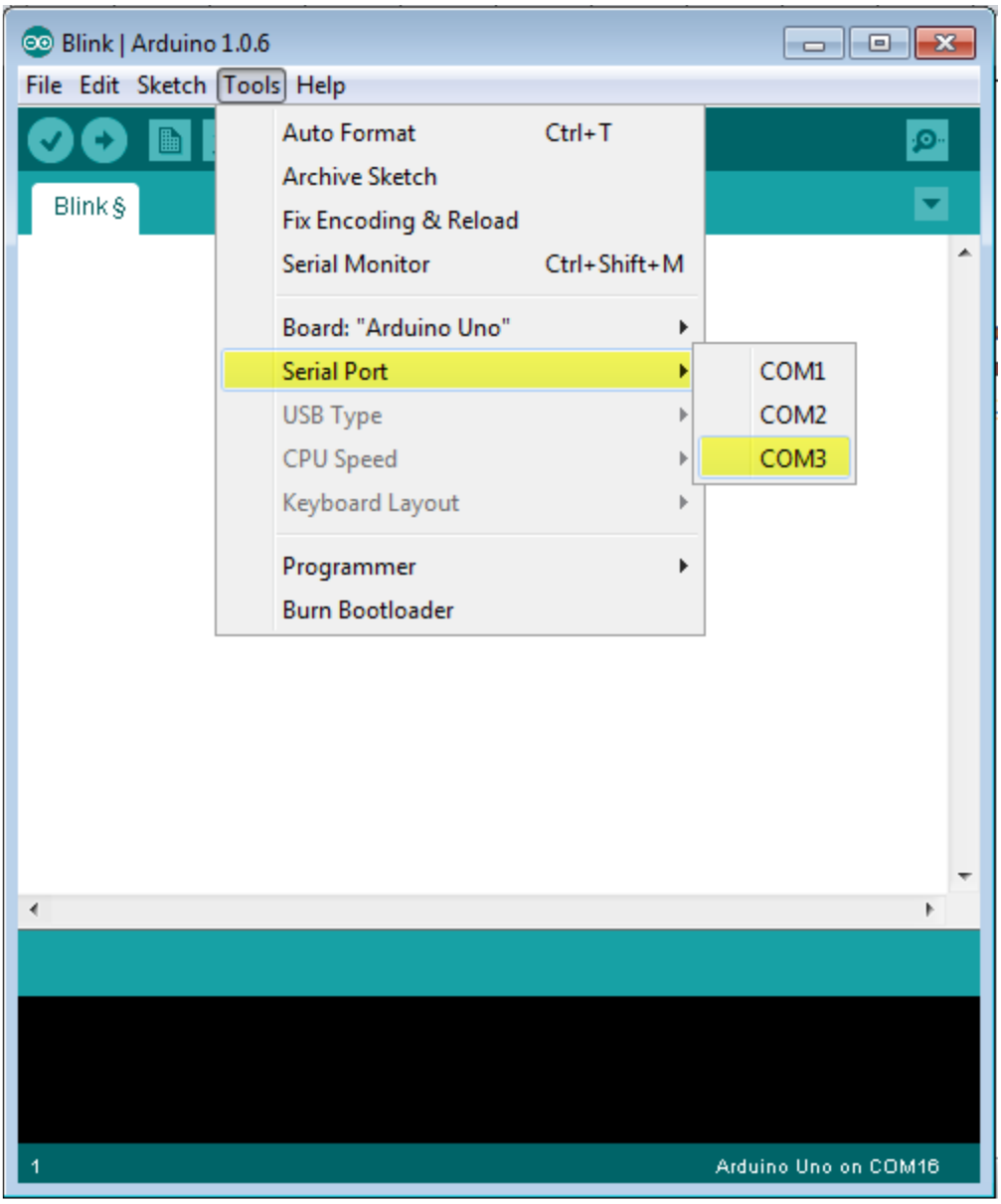

Fig 1.35: Selecting Serial Port

- **Step 8: Upload the program to your board.**

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.

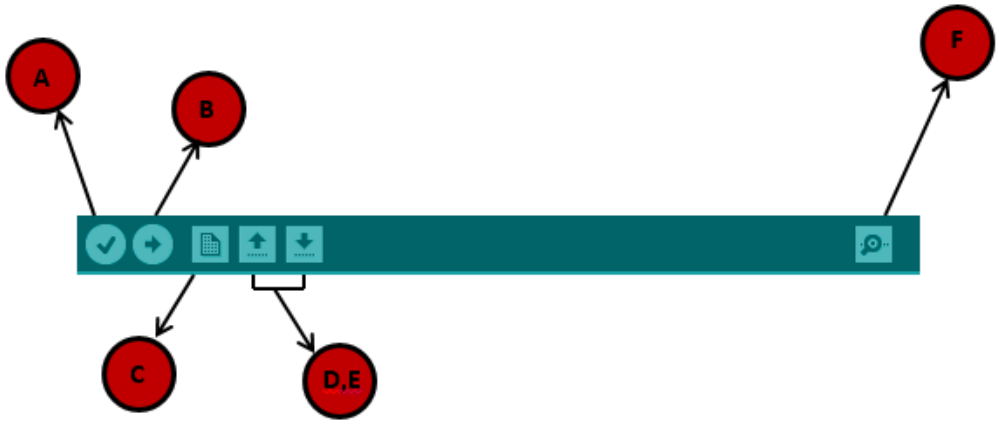

Fig 1.36: Uploading Program to the Board

A. Used to check if there is any compilation error.

B. Used to upload a program to the Arduino board.

C. Shortcut used to create a new sketch.

D. Used to directly open one of the example sketches.

E. Used to save your sketch.

F. Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

Note: If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

4.1.4 Arduino Programming Structure

In this chapter, we will study in depth, the Arduino program structure and we will learn more new terminologies used in the Arduino world. The Arduino software is open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

Sketch: The first new terminology is the Arduino program called "**sketch**".

Structure:

Arduino programs can be divided in three main parts: **Structure**, **Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions:

- Setup( ) function
- Loop( ) function

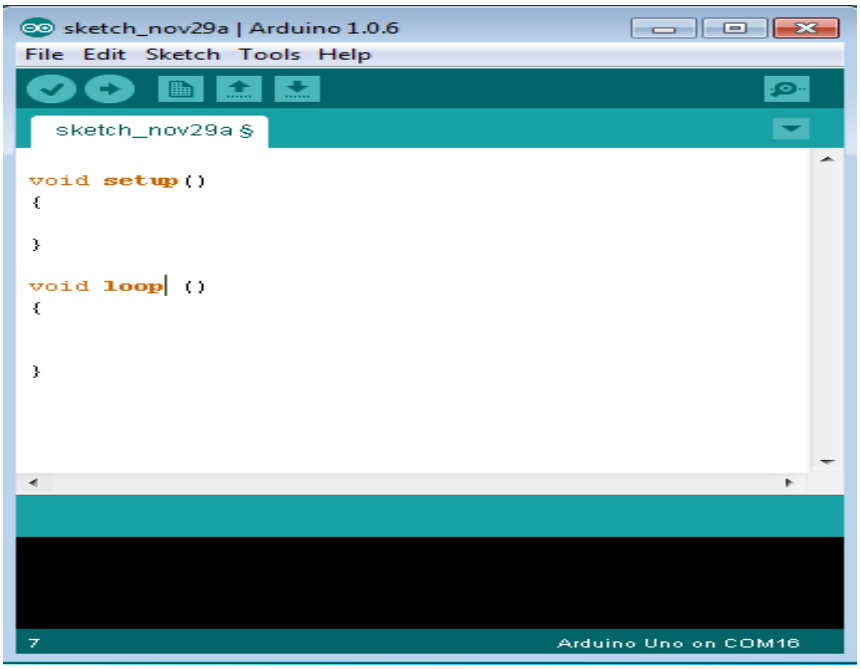

Fig 1.37: Sketch

Void setup ( )

{

}

- **PURPOSE:**

  The **setup()** function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.:

  - INPUT
  - OUTPUT
  - RETURN

Void loop() { }:

- **PURPOSE:**

After creating a **setup()** function, which initializes and sets the initial values, the **loop()** function does precisely what its name suggests, and loops secutively, allowing your program to change and respond. Use it to actively control the Arduino board.

- INPUT
- OUTPUT
- RETURN

CHAPTER – 5

SOURCE CODE

```
#include <SoftwareSerial.h>
 #include <Servo.h>
#include <LiquidCrystal.h>
// Pin Definitions
const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8;
const int m1 = 7, m2 = 6, m3 = 5, m4 = 4;
const int echo = A4, trig = A5;
const int buzzer = A1;
const int servoPin = 3;
// Global Variables
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
Servo myservo;
String data;
const float SAFE_DISTANCE = 20.0;  // Safe distance in cm
const float CRITICAL_DISTANCE = 10.0;  // Critical proximity distance
unsigned long lastMoveTime = 0;
const unsigned long MOVE_TIMEOUT = 5000;  // 5 seconds max movement time
// Enhanced Obstacle Detection and Avoidance
float measureDistance() {
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  float duration = pulseIn(echo, HIGH);
```

```
  return duration / 58.2;  // Convert to cm
}

void signalAlert(float distance) {
  lcd.clear();
  lcd.print("Dist: ");
  lcd.print(distance);
  lcd.print(" cm");
  lcd.setCursor(0, 1);

  // Different alerts based on proximity
  if (distance < CRITICAL_DISTANCE) {
    lcd.print("CRITICAL STOP!");
    digitalWrite(buzzer, HIGH);
    stop1();
    delay(500);
    digitalWrite(buzzer, LOW);
  } else {
    lcd.print("OBSTACLE AHEAD!");
    digitalWrite(buzzer, HIGH);
    delay(200);
    digitalWrite(buzzer, LOW);
  }
}

void obstacleAvoidance() {
  // Scan for alternative paths
  myservo.write(0);   // Center
```

```
  delay(500);
  float centerDistance = measureDistance();
  myservo.write(45);  // Right scan
  delay(500);
  float rightDistance = measureDistance();
  myservo.write(135); // Left scan
  delay(500);
  float leftDistance = measureDistance();
  myservo.write(90);  // Servo middle position
  // Decision making for path selection
  if (centerDistance > SAFE_DISTANCE) {
    front();  // Primary path clear
  } else if (rightDistance > leftDistance) {
    right();  // More space on right
  } else {
    left();   // More space on left
  }
}

void setup() {
  Serial.begin(9600);
  // LCD Initialization
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("OMNIDIRECTIONAL");
  lcd.setCursor(0, 1);
  lcd.print("PATHFINDING ROBOT");
  delay(1500);
```

```
  // Motor Pin Setup
  pinMode(m1, OUTPUT);
  pinMode(m2, OUTPUT);
  pinMode(m3, OUTPUT);
  pinMode(m4, OUTPUT);

  // Ultrasonic Sensor Pins
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);

  // Buzzer and Servo Setup
  pinMode(buzzer, OUTPUT);
  digitalWrite(buzzer, LOW);

  myservo.attach(servoPin);
  myservo.write(90);  // Center position

  // Initial Motor State
  stop1();

  // Startup Indication
  digitalWrite(buzzer, HIGH);
  delay(300);
  digitalWrite(buzzer, LOW);
}

void front() {
```

```
  Serial.println("MOVING FORWARD");
  lcd.clear();
  lcd.print("FORWARD");

  digitalWrite(m1, HIGH);
  digitalWrite(m2, LOW);
  digitalWrite(m3, HIGH);
  digitalWrite(m4, LOW);

  lastMoveTime = millis();
}

void back() {
  Serial.println("MOVING BACKWARD");
  lcd.clear();
  lcd.print("BACKWARD");

  digitalWrite(m1, LOW);
  digitalWrite(m2, HIGH);
  digitalWrite(m3, LOW);
  digitalWrite(m4, HIGH);

  lastMoveTime = millis();
}

void left() {
  Serial.println("TURNING LEFT");
  lcd.clear();
```

```
  lcd.print("LEFT TURN");

  digitalWrite(m1, HIGH);
  digitalWrite(m2, LOW);
  digitalWrite(m3, LOW);
  digitalWrite(m4, HIGH);

  lastMoveTime = millis();
}

void right() {
  Serial.println("TURNING RIGHT");
  lcd.clear();
  lcd.print("RIGHT TURN");

  digitalWrite(m1, LOW);
  digitalWrite(m2, HIGH);
  digitalWrite(m3, HIGH);
  digitalWrite(m4, LOW);

  lastMoveTime = millis();
}

void stop1() {
  Serial.println("STOPPED");
  lcd.clear();
  lcd.print("STOPPED");
```

```
  digitalWrite(m1, LOW);
  digitalWrite(m2, LOW);
  digitalWrite(m3, LOW);
  digitalWrite(m4, LOW);
}

void loop() {
  // Check for timeout to prevent prolonged movement
  if (millis() - lastMoveTime > MOVE_TIMEOUT) {
    stop1();
  }

  // Obstacle Detection
  float distance = measureDistance();

  if (distance < CRITICAL_DISTANCE) {
    signalAlert(distance);
    obstacleAvoidance();
  }

  // Serial Command Processing
  while (Serial.available()) {
    data = Serial.readString();
    delay(100);

    lcd.clear();
    lcd.print("CMD: " + data);
```

```
  // Command Processing
  switch(data[0]) {
    case 'f': front(); break;
    case 'b': back(); break;
    case 'l': left(); break;
    case 'r': right(); break;
    case 's': stop1(); break;
    default:
      Serial.println("Invalid Command");
      lcd.clear();
      lcd.print("INVALID CMD");
      break;
  }
 }

 delay(100);  // Small delay for stability
}
```

CHAPTER – 6:

ADVANTAGES & FUTURE SCOPE

6.1 Advantages:

- **Cost Effective:**
  Using a single ultrasonic sensor significantly reduces the hardware cost compared to setups with multiple sensors or more complex navigation systems.

- **Simplicity:**
  A mean-based approach simplifies the computation required for determining obstacles and paths, making it ideal for embedded systems with limited processing power.

- **Real-Time performance**
  Calculating mean values for distance measurements is computationally light, which allows for quicker responses and better real-time navigation.

- **Noise Reduction**
  Averaging multiple readings helps minimize the impact of sensor noise and anomalies, ensuring smoother pathfinding and fewer false obstacle detections.

- **Power Efficiency**
  With fewer sensors involved, the overall energy consumption is lower, making it suitable for battery-operated robots.

6.2 Future Scope:

- **Enhanced Sensor Integration:**
  Incorporate additional sensors (e.g., infrared or LiDAR) alongside the ultrasonic sensor to improve obstacle detection, precision, and environmental mapping. This would allow the robot to handle more complex terrains.

- **AI Integration:**
  Use machine learning algorithms to improve the mean-based approach by training the robot on different datasets. This could help the robot predict and adapt to obstacles more effectively.

- **Application Expansion:**
  Adapt the robot for specialized tasks, like warehouse automation, search-and-rescue operations, or agricultural monitoring, making the design more versatile and scalable.

- **Global Positioning & Mapping:**
  Integrate GPS or SLAM (Simultaneous Localization and Mapping) technologies to allow the robot to map and navigate large areas autonomously.

- **Dynamic Path Planning:**
  Implement real-time dynamic path planning that accounts for moving obstacles, enabling the robot to navigate crowded or unpredictable spaces.

CHAPTER – 7

RESULT AND CONCLUSION

7.1 Result:

7.1.1 OFF State

The below figure indicates the result of the system when it's in off state which means there is no power supplied. It contains Arduino UNO, Power Supply, Buzzer, Transformer, L293D motor driver, Arduino base board, LCD display, Ultrasonic Sensor, Servo motor, External power supply board.

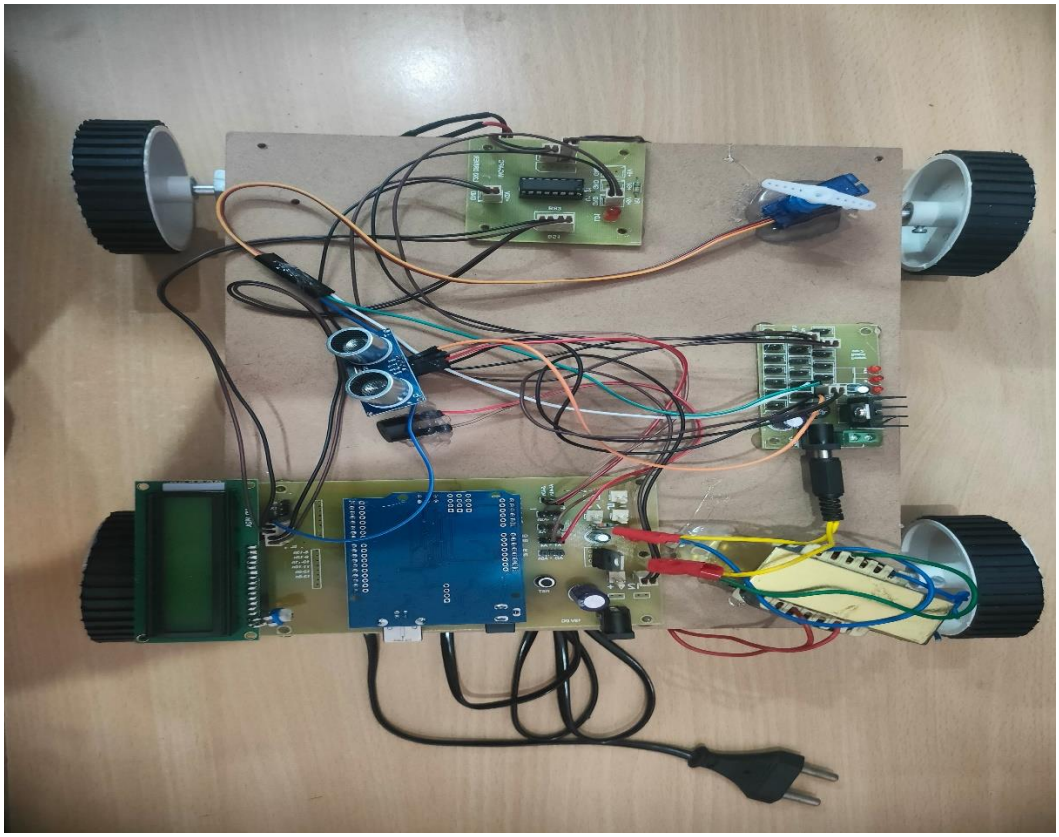

Fig 7.38: OFF State

7.1.2 ON State

The below figure indicates the result of the system when it's in **ON State** means power is supplied to the Arduino board. It contains Arduino UNO, Power Supply, Buzzer, Transformer, L293D motor driver, Arduino base board, LCD display, Ultrasonic Sensor, Servo motor, External power supply board.

The board makes the buzzer sound when its turned ON and the LCD displays the title of the project which is Omni-directional pathfinding robot.

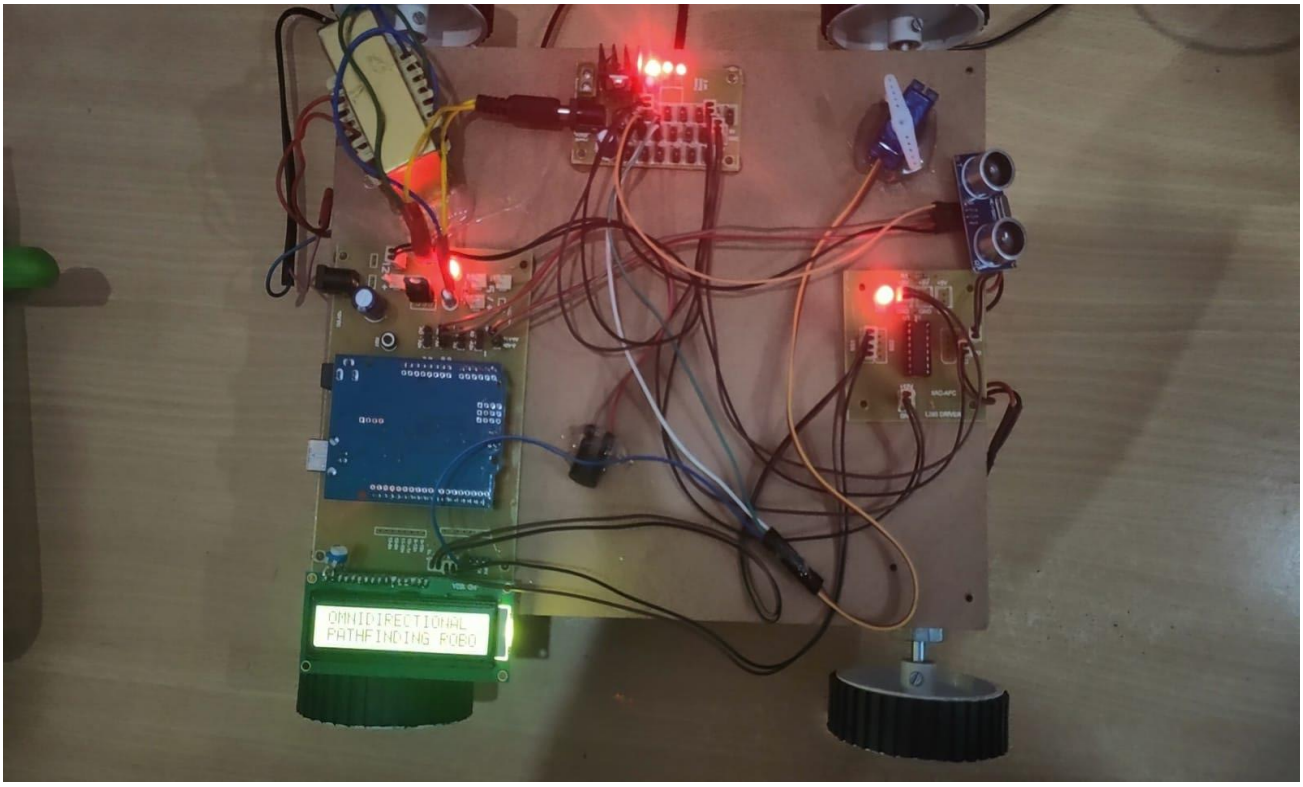

Fig 7.39: ON State

7.1.3 Different Directions and functions

7.1.3.1 Forward direction

The below figure indicates the result of the when its moving in the forward direction. Both the motor wheels rotate in the same direction.

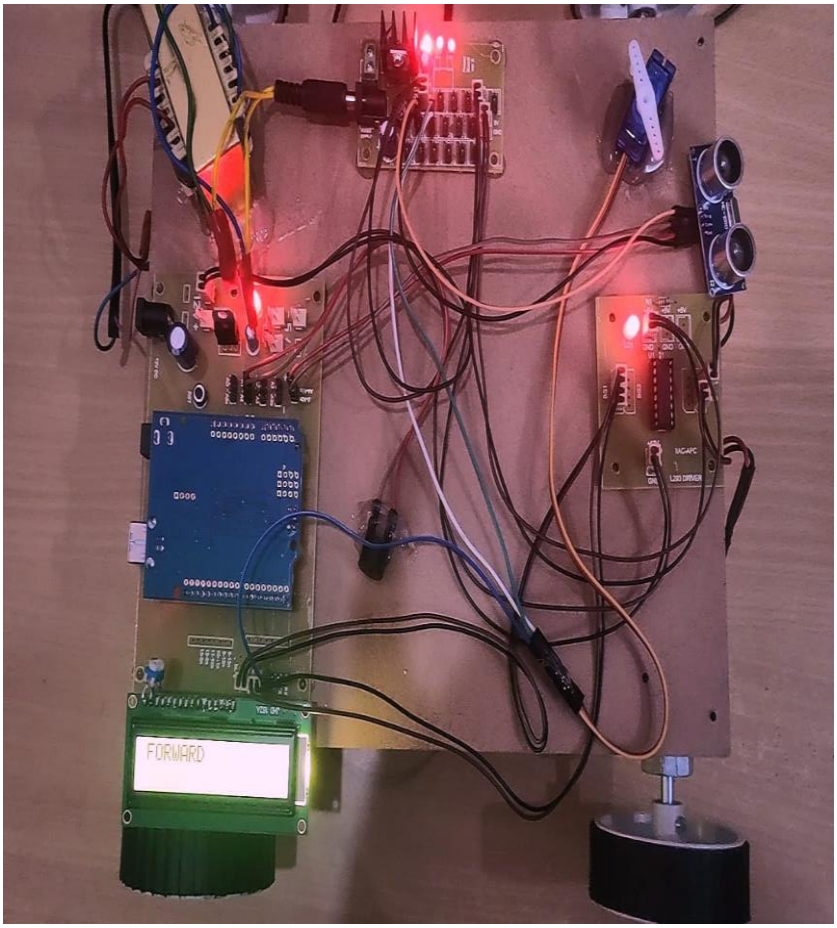

Fig 7.40: Forward direction

7.1.3.2 Stop Condition

The below figure indicates the result of the system when its in Stop position due to obstacle in the front. Once it's in Stop Position it scans for 180 deg and checks for free path avoiding the obstacle.

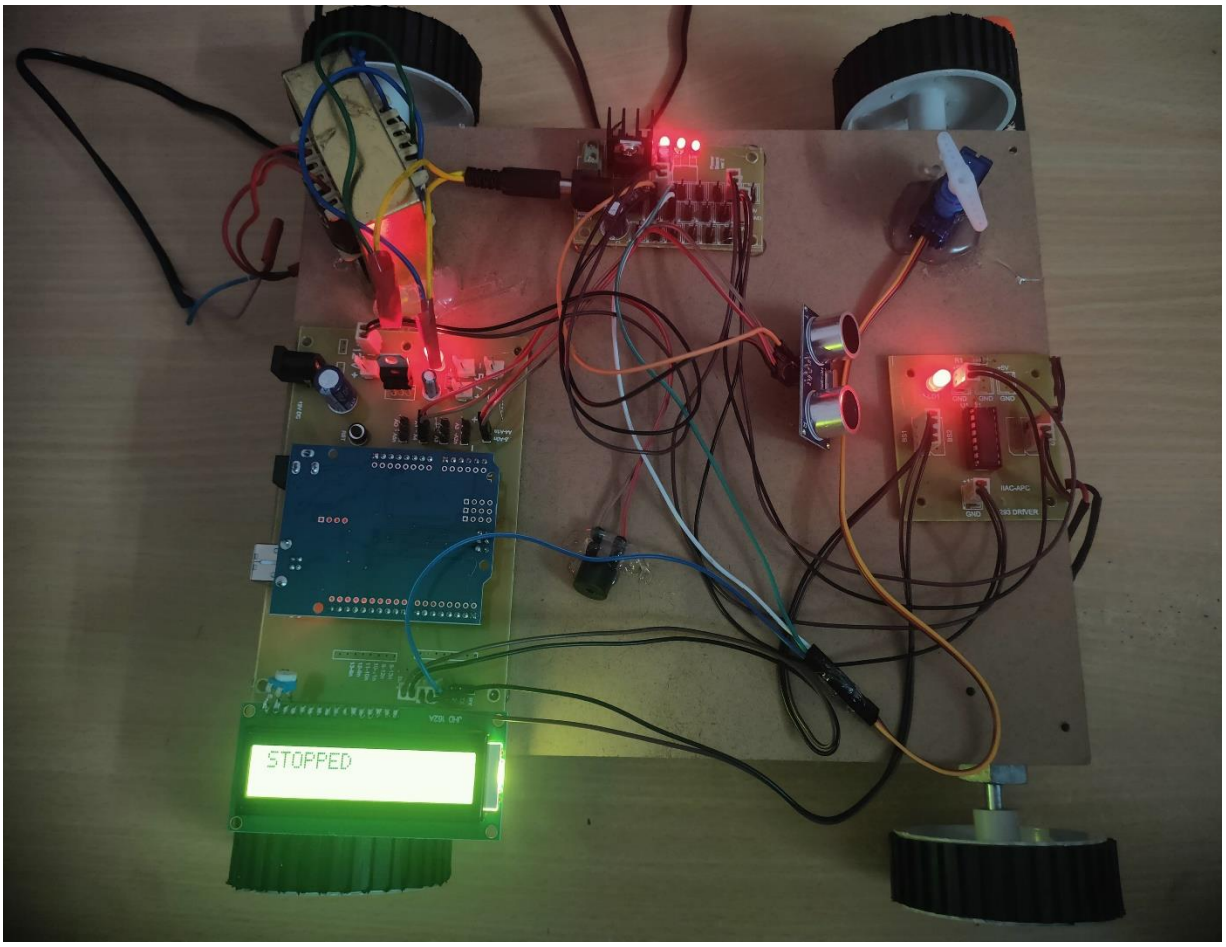

Fig 7.41: Stop Condition

7.1.3.3 Left turn Condition

The below figure indicates the result of the system when it's in turning position to the left direction due to obstacle in the front. Once it's in rotating motion it moves in that direction and then it scans for 180 deg and checks for free path avoiding the obstacle.

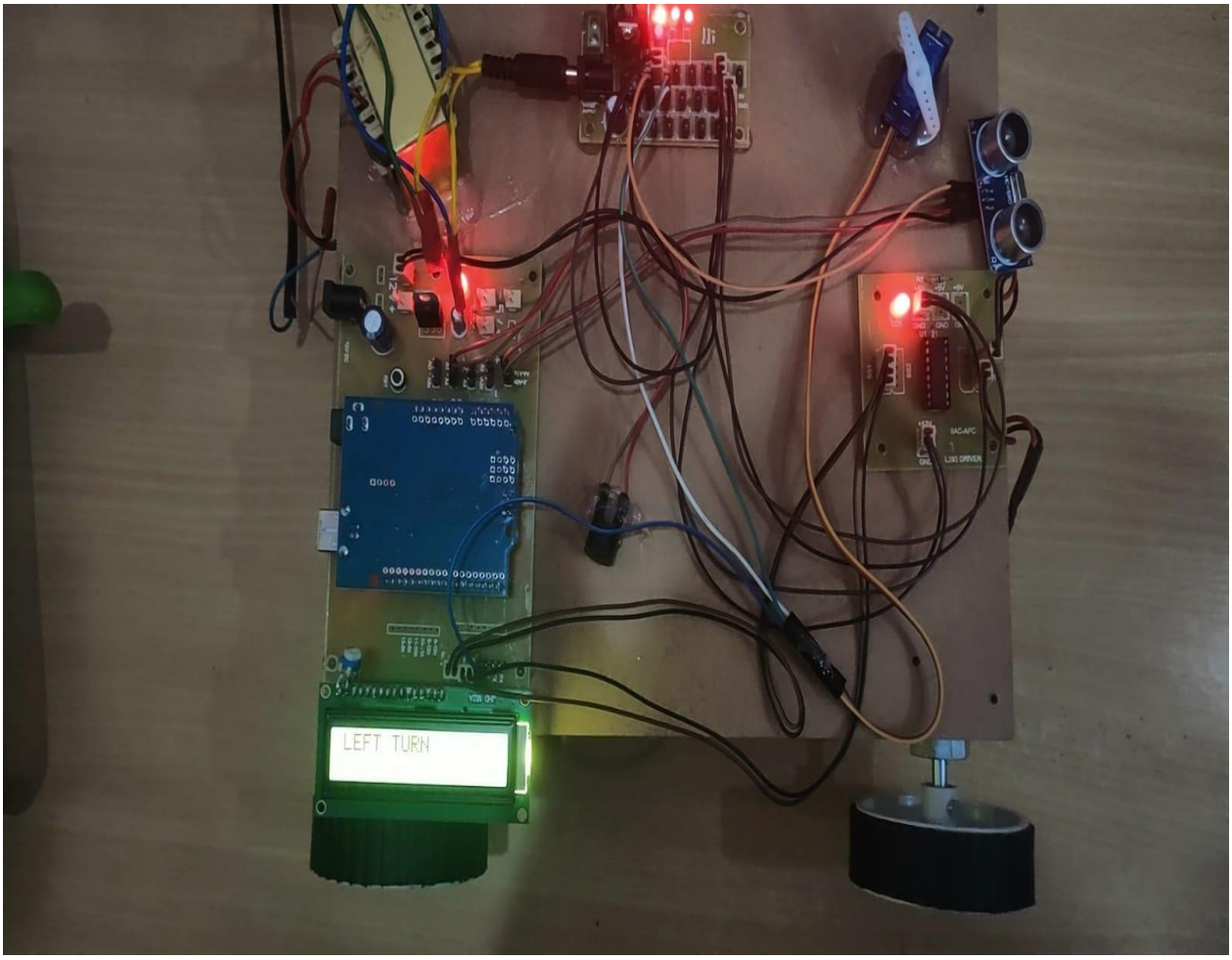

Fig 7.42: Left turn

7.1.3.4 Right Turn Condition

The below figure indicates the result of the system when it's in turning position to the Right direction due to obstacle in the front. Once it's in rotating motion it moves in that direction and then it scans for 180 deg and checks for free path avoiding the obstacle.

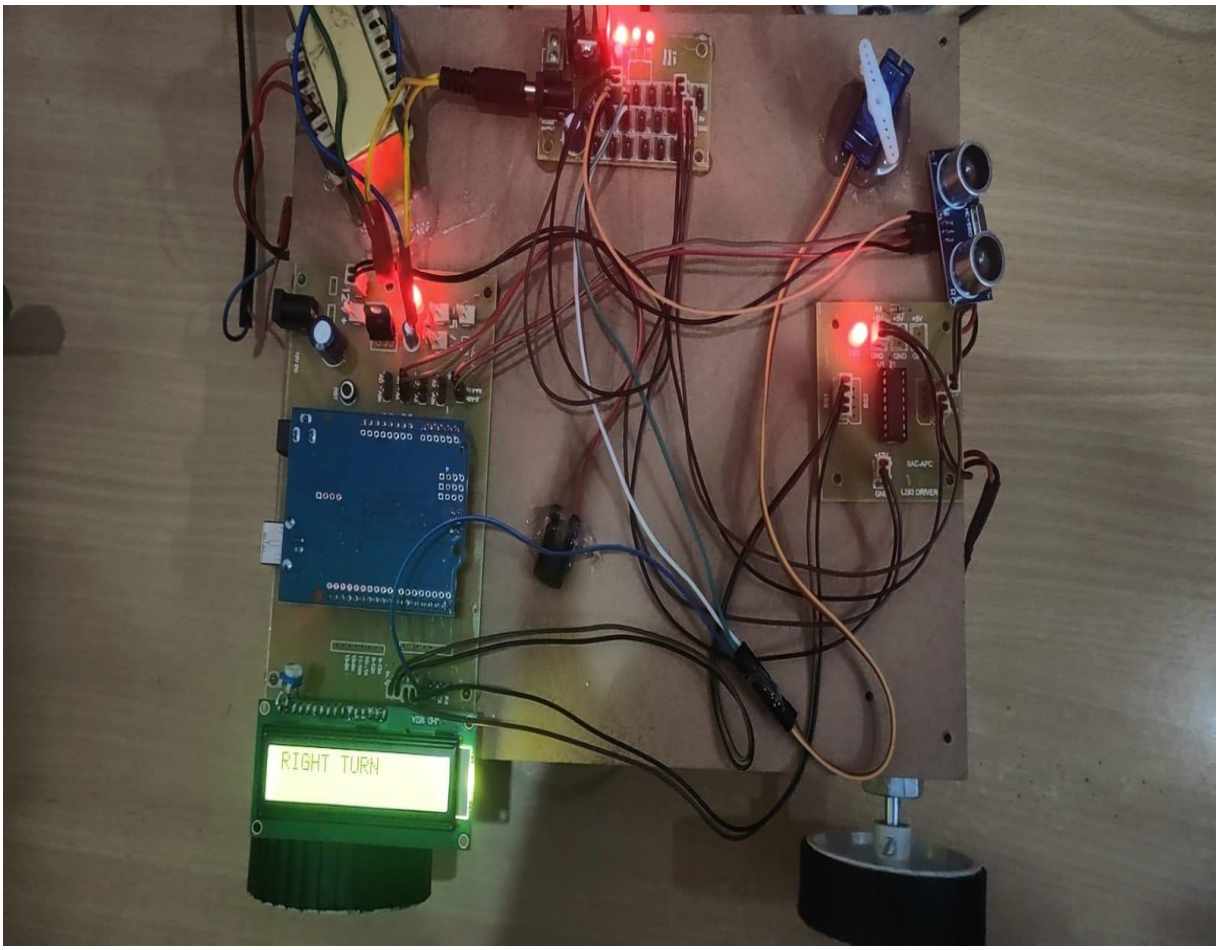

Fig 7.43: Right Turn

7.1 CONCLUSION

The integration of an ultrasonic sensor, servo motor, and L293D motor driver presents an advanced solution for autonomous robotic navigation. Unlike traditional systems, this approach significantly enhances obstacle detection coverage, minimizes navigation errors, and ensures precise movement control. The use of a mean-based algorithm ensures accurate path selection, reducing false positives and improving overall efficiency. Future enhancements may include AI-driven obstacle recognition, adaptive machine learning-based navigation, and additional sensor integration for increased robustness. This innovative and cost-effective solution has wide-ranging applications in industrial automation, autonomous vehicle navigation, and exploration in hazardous environments.

REFERENCES

- T. Mahmud et al., "Design and Implementation of an Ultrasonic Sensor-Based Obstacle Avoidance System for Arduino Robots," 2023 Int. Conf. Information and Communication Technology for Sustainable Development.
- L. Chen et al., "Wireless Car Control System Based on ARDUINO UNO R3," 2018 2nd IEEE Advanced Information Management Conference.
- R. Sissodia et al., "Arduino-Based Bluetooth Voice-Controlled Robot Car and Obstacle Detector," 2023 IEEE Int. Students' Conf. on Electrical, Electronics, and Computer Science (SCEECS).
- T. Akilan et al., "Surveillance Robot in Hazardous Places Using IoT Technology," 2020 Int. Conf. Advances in Computing, Communication, and Networking.
- "Control DC, Stepper & Servo with L293D Motor Driver Shield & Arduino," Last Minute Engineers, 2018.