ROBOCEK PROJECT DEVELOPERS

ACTUATOR, ROBOTICS WITH ARDUINO

ROBOCEK 2020-21

First printing, January 2022

There are an endless number of things to discover about robotics. A lot of it is just too fantastic for people to believe.

-Daniel H. Wilson

Acknowledgment

A great deal of time and effort have been put forward to compile this book with relevant topics. Thank God almighty for providing all the necessary resources and facility through our colleagues, friends, faculty members who have helped us in every possible way to compile this book. We thank ROBOCEK Execom 2020-21 for providing us the opportunity in writing this book. We would extent our sincere gratitude to ROBOCEK Execom members and ROBOCEK alumni for their support and suggestions. Last, but not the least, we thank our family members and friends for their support throughout this journey. We also thank all other authors and developers whose work have given us the inspiration and guidance to compile this book. We thank Arduino Community, tinkerCad, LastMinuteEngineering, stackOverflow, Overleaf and all other services, who have provided excellent articles, blogs and resources in building this book.

Preface

This is a compilation for the pioneers into the stream of Robotics. Robotics is believed to have the potential to positively transform lives and work practices, raise efficiency and safety levels and provide enhanced levels of service. Even more, robotics is set to become the driving technology nurturing a whole new generation of autonomous devices and cognitive artefacts that, through their learning capabilities, interact seamlessly with the world around them, and hence, provide the missing link between the digital and physical world. The era has advanced so much that the neural networks escalate their accuracy to prove that they replicate human thought processes. There it becomes necessary for an engineer to engineer robots.

It is undeniable that a good technical book is a great source of knowledge and an effective tool to foster the real aspects of engineering into the students. Bearing this in mind, in order to make the first step of beginners a little more easy and grounded we bring this compilations of ours to light. For most of the novels into robotics, Arduino might be the entry point. This material focuses on Arduino right from its base, Programming Arduino with C, and interfacing ideas of several basic modules. Traversing through each and every journal and books that might equip us to get this done has really added up to our knowledge. We were able to correct many of our notions and make a journey through many concepts really worth understanding being technical aspirants. Taking another perspective we have recognized the importance of dedication, patience and commitment that we should cultivate to improve ourselves relentlessly.

ACTUATOR : ROBOTICS WITH ARDUINO

With love

Edwin Jose George
edwinjosegeorge@gmail.com
ROBOCEK Execom member (2020-21)
Department of Computer Science and Engineering
Govt. College of Engineering Kannur (2018-22)

Meghana T.V
meghanathemmanamveedu@gmail.com
ROBOCEK Execom member (2020-21)
Department of Computer Science and Engineering
Govt. College of Engineering Kannur (2018-22)

For the pioneers into robotics with Arduino...

Contents

List of Figures

List of Tables

Acronyms

ADC	Analog to Digital Conversion
AI	Artificial Intelligence
DTMF	Dual Tone Multi Frequncy
EMR	Electro Magnetic Radiation
IC	Integrated Circuits
ICSP	In-Circuit Serial Programing
IDE	Integrated Development Environment
IoT	Internet of Things
IR	Infra-Red
LDR	Light Dependent Resistor
PIR	Passive Infrared
PWM	Pulse With Modulation
SBC	Single Board Computer
SoC	System on Chip

1 The ROBOCEK Family

1.1 Vision

To devise a passionate community of strong responsible engineers technically skilled enough to figure out the need of the hour and act wisely to crack the challenges emphasizing the universal values.

1.2 Mission

To foster young minds into the ever evolving realm of robotics fabricating an ample learning environment aiding in design, operation and robotics with the aid of hands-on projects and collaborations with the experienced.

1.3 Values

Strives to excel as a learning community by adapting, learning and relentlessly improving ourselves and the entire team by promoting initiatives devising healthy, ethical and social relations adhering to sustainable models of development.

1.4 Down the Memory Lane

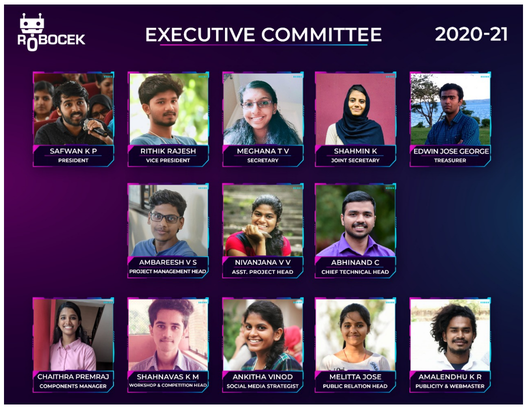
A vibrant team of young enthusiasts from Govt. College of Engineering Kannur inspired from a group of students from Chennai, fabricated a small team with one dream to excel in the field of robotics. They named it ***'Robotic Enthusiasts Of GCEK'***. They hanged on to their dream of a robotics club for GCEK till it attained its official recognition on 30th January, 2014.

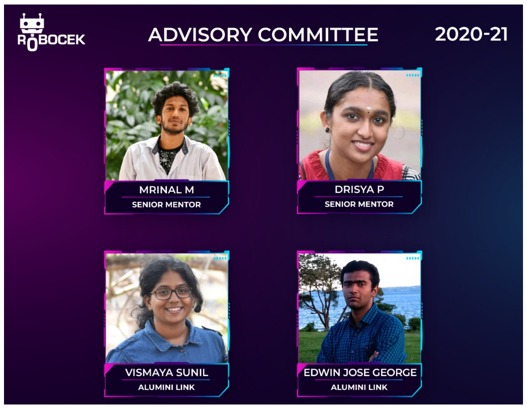
ROBOCEK gradually evolved from a handful of members with no background in robotics into one of the most highly acclaimed robotics club in Kerala with a dedicated team and unmatched performance. The

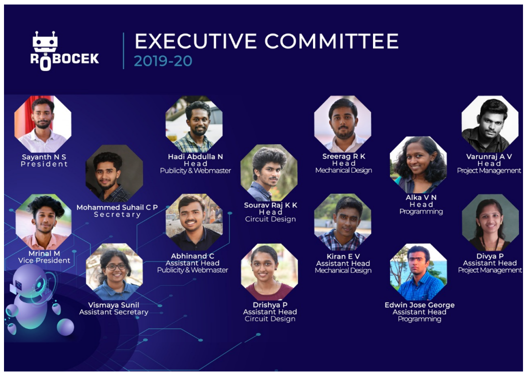
hardcore activities were accomplished by the students who organised themselves into an Executive Committee with prominent guidance from the Staff Advisors. They realised the dream of a room dedicated to ROBOCEK in GCEK. They conceived the idea of a basic workshop that would pave a path into the realm of robotics under ROBOCEK and finally converged to *'ACTUATOR',The Beginner's Workshop*. Bit by bit 'Actuator' turned to be the first and the most interesting workshop that waved among the freshers of GCEK and also any entry point into the ROBOCEK. The pioneers into the club slowly engaged themselves into learning in groups.

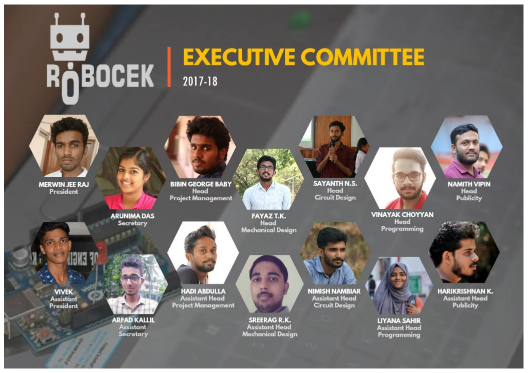
The dedicated cluster of students strived to explore the intangible sphere of robotics through workshops, competitions, collaborative learning and expeditions nurturing academic excellence striving to keep in pace with electronics and robotics. The team endeavoured various tech fests, including national level fest - Xplore'19 and social relevant projects like expo's,flood relief activities, workshops etc.

1.5 *ROBOCEK Execom*









It's always been a proud moment to see an enthusiastic team that shape the progress of our club. Starting from 2014, Dr T D John, then Principal GCE Kannur laid the foundation for the robotics club for the first time. Led by Sreepathi A (Batch 2k13-2k17) and Jayakrishnan M (Batch 2k13-2k17), striving through initial and difficult times, along with support from faculties especially Dr Abdul Nazar K P, Dr A Ranjith Ram, paved a better way to the future generation of the college to invest their interest and passion in innovative technology. With rising demand for betterment, an excellent team of enthusiastic students were chosen to drive ROBOCEK, *'The Robocek Execom'*, meeting the vision and mission of the club. Though led by a team, each and every innovation are always welcomed, helping the students to advance their career. Our courage to advance is from the everlasting support from ROBOCEK Alumni, providing examples for aspirants.

1.6 Reach out to us

- https://robocek.org
- robocek@gcek.ac.in
- https://github.com/robocek
- https://www.linkedin.com/company/robocek
- https://www.facebook.com/robocek
- https://www.instagram.com/robocek_official
- https://www.youtube.com/ROBOCEK

2
Introduction to Robotics

2.1 *What is Robotics?*

Robotics is an interdisciplinary field that integrates Computer Science and Engineering. It comprises design, construction, operation and use of robots. The field targets the creation of machines that can help and assist humans through blending aspects of mechanical engineering, electrical engineering, mechatronics, electronics etc

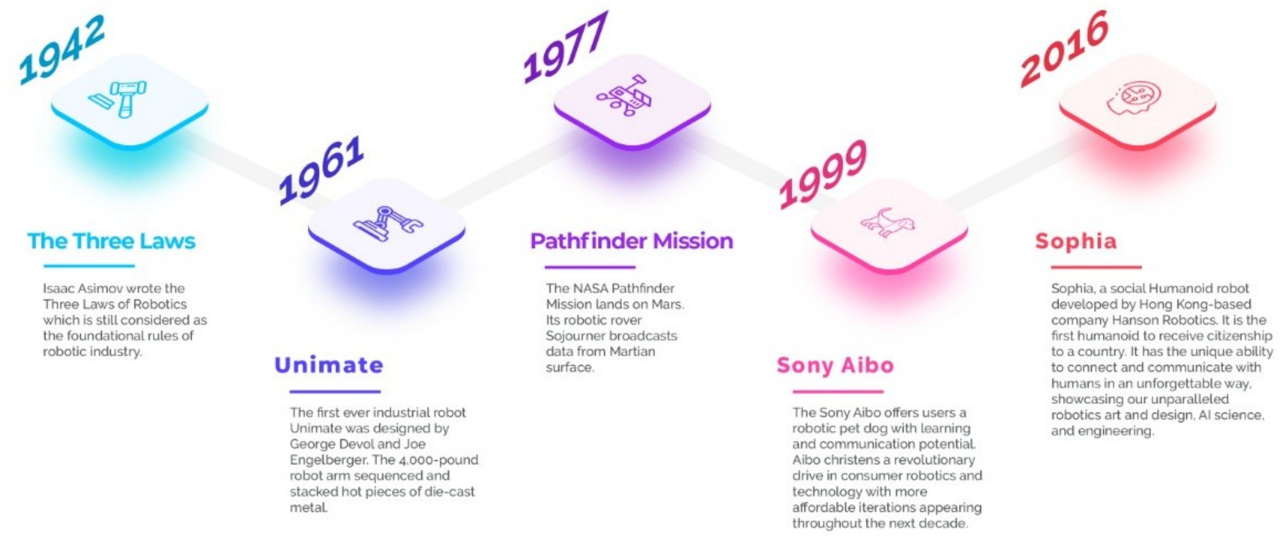
A Robot is an interactive machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer, to reduce human risk in hazardous works. Thus, robotics is an interdisciplinary branch of engineering and science that deals with the design, construction, operation, and use of robots, as well as computer systems for their control, sensory feedback, and information processing to create an efficient robot. A robot can be thought as an combination of sensors, controllers and actuators. Sensor provide collection of information from environment, that are processed by controllers who generate signal to actuators to interact with environment.

Robots are put to use in different applications. Their usage is determined by the field they concentrate on. Robotics is a vast field. They may be manufactured or processed in different forms, shapes and utility. Joseph Engelberger, godfather of robotics, once said *"I can't define a robot, but I know one when I see one."*

2.2 *Evolution of Robotics*

The question of evolution of robotics take us to the ancient world and era of industrial revolution. Engineers were trying to develop machines that can handle dangerous tasks for automotive industry and defence. Robots they developed were meant to be a replica of human actions.

Major Events in the Evolution of Robotics


1942
The Three Laws
Isaac Asimov wrote the Three Laws of Robotics which is still considered as the foundational rules of robotic industry.
1961
Unimate
The first ever industrial robot Unimate was designed by George Devol and Joe Engelberger. The 4,000-pound robot arm sequenced and stacked hot pieces of die-cast metal.
1977
Pathfinder Mission
The NASA Pathfinder Mission lands on Mars. Its robotic rover Sojourner broadcasts data from Martian surface.
1999
Sony Aibo
The Sony Aibo offers users a robotic pet dog with learning and communication potential. Aibo christens a revolutionary drive in consumer robotics and technology with more affordable iterations appearing throughout the next decade.
2016
Sophia
Sophia, a social Humanoid robot developed by Hong Kong-based company Hanson Robotics. It is the first humanoid to receive citizenship to a country. It has the unique ability to connect and communicate with humans in an unforgettable way, showcasing our unparalleled robotics art and design, AI science, and engineering.

Further advancements in the technical sphere since the 2000's have led to more advanced automation and innovation. The improvements in the fields of Machine Learning and Artificial Intelligence (AI) again paves great milestones in the evolution of robotics. Automated machines have turned to be common comrades in industries like manufacturing, maritime explorations, military, agri-technologies.AI is utilized to assess environments and act according to the programmed goals. Recent developments have also led to the use of these technologies in data and predictive analytics to improve the customer satisfaction. The developments in this realm is certainly going to rule the pulse of the impending technical world.

3
Introduction to Arduino

Arduino is it one of the most widely used micro-controller to develop various DIY projects. The low cost and easy to deploy has given rise to various Internet of Things (IoT) and embedded projects. Arduino is an open source electronics platform based on easy-to-use hardware and software. Usually the term "Arduino" can be used to refer the following.

- Open Source Electronics Platform : Free to design and implement various units, easy availability and high customization.
- Arduino Integrated Development Environment (IDE) : A software to program Arduino boards.
- Online Arduino Community : Fast growing community who maintains and supports Arduino Developments.
  `https://github.com/arduino/` `https://www.arduino.cc/`

The Arduino project was started in 2005 as a program for students at the Interaction Design Institute Ivrea in Ivrea, Italy , aiming to provide a low-cost and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. The name Arduino comes from a bar in Ivrea, Italy, where some of the founders of the project used to meet. The ease at which various units can be attached gained Arduino its popularity. Anyone can start with programming and robotics by just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

Following are few of the key points of Arduino :

- Easy to use and expensive
- Cross-platform and open source
- Simple and clear programming
- Extensible software/hardware

Figure 3.1: Arduino Open Source Community

3.1 *Comparing Arduino to its alternatives*

Arduino is not the only board available to build custom projects and applications. Raspberry Pi, BeagleBone, Sharks Cove, Minnowboard MAX, Nanode, Waspmote or LittleBits are some of the most interesting alternatives to Arduino. Arduino and Raspberry Pi are the ones receiving the most attention within the community of software developers.

Raspberry Pi is a low cost Single Board Computer (SBC) developed by the British Raspberry Pi Foundation. They are used in places which require higher and faster calculations. They are boards with micro processors. Raspberry Pi acts as a mini computer with additional I/O connection pins along with Wi-Fi, Bluetooth and ability to connect to external devices via HDMI, USB ports etc.. It have a complete Operating System (Raspberian) burned into a SD card.

Arduino, on the other hand is a low cost System on Chip (SoC) board. They are used where complex information need not be analyzed. They are heavily used in Embedded systems and IoT. They are boards with micro controller that control other appliances. There are no Operating System burned into Arduino. Arduino simply uses machine code to execute instructions. The machine code are created by compiling programs written in high level languages like C, into an executable binary file.

Figure 3.2: Development boards

(a) Arduino Uno R3

(b) Raspberry Pi 2 Model B

3.2 *Micro-controllers and Micro-processors*

Micro-controllers and Micro-processors are common terms used in IoT and embedded systems. It is worth a while to understand the difference between them, to choose which is better to our project need.

Micro-controllers and micro-processors are used to execute instructions and control various units interfaced with them. However their complexity and utility can vary greatly. Micro controllers are similar to a small computer fabricated into a single Integrated Circuits (IC). It contains a processor core, ROM, RAM, and I/O pins It does not need any external circuits to do its task. It can manage memory and other services its own. It does the job of managing units as well an performing calculations. Micro-processor, on the other hand, has only CPU inside them. It does not have RAM, ROM of its own. Processors are dedicated to perform calculations. They depend on external circuits for its peripheral like RAM, ROM to work. They are used where the task are complex and tricky. The features are summarized in table 3.1.

(a) Microprocessor

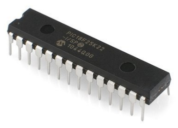

(b) Micro-controller

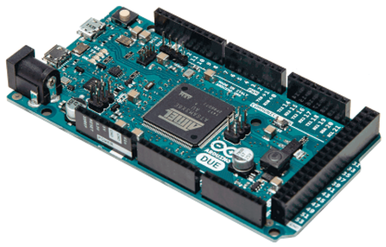
Figure 3.3: Processor and Micro-Controllers

Micro processor	*Micro controller*
RAM, ROM, EEPROM needs to be connected	RAM, ROM, EEPROM are present on single IC
Expensive	Cheap
High processing speed (>1Ghz)	Low processing speed (<50Mhz)
No power saving technology	Optimized power usage
Used in large applications	Used in small application
Process complex task	Process simple task
Dissipate high heat. Might need cooling	Does not dissipate high heats.
Usage of external storage (HardDisk - GB of spaces)	Usage of internal Storage (EEPROM - few KB space)
Eg: Intel Pentium 4, Intel Core i7, AMD Athlon	Eg: ATmega328, ESP8266, ESP32, ATMEGA32U4
Board: Raspberry Pi	Board: Arduino

Table 3.1: Comparison of Micro-processors and Micro-controllers

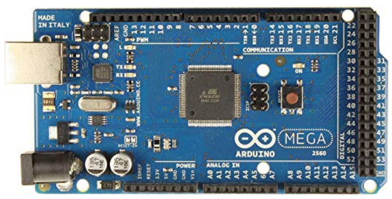
3.3 *Arduino Boards*

Arduino is a large community that develops various micro-controller boards. Depending on the project application and usage, various customized official boards are available. The most generally used Arduino board is the "Arduino Uno". We would be making use of Arduino Uno to develop various projects.


(a) Arduino Due

(b) Arduino Leonardo

(c) Arduino Uno R3


(d) Arduino Mega 2560

(e) Arduino Nano

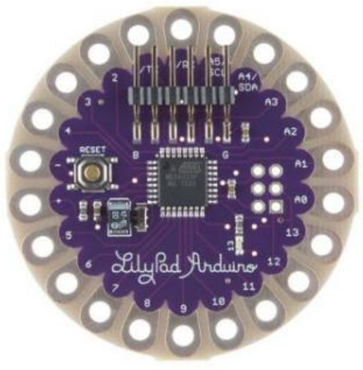

(f) LillyPad Arduino

Figure 3.4: Arduino Boards

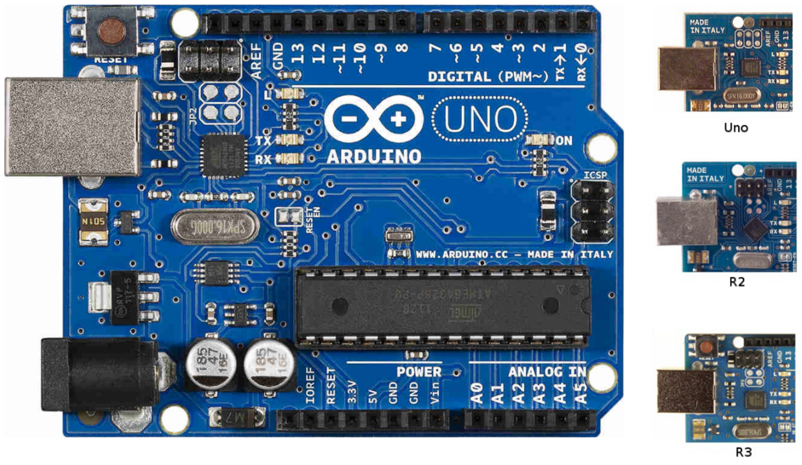


Figure 3.5: Arduino Uno versions. Notice the alignment and position of ATMega16U2 microcontroller

3.4 *Arduino Uno R3*

On-board units

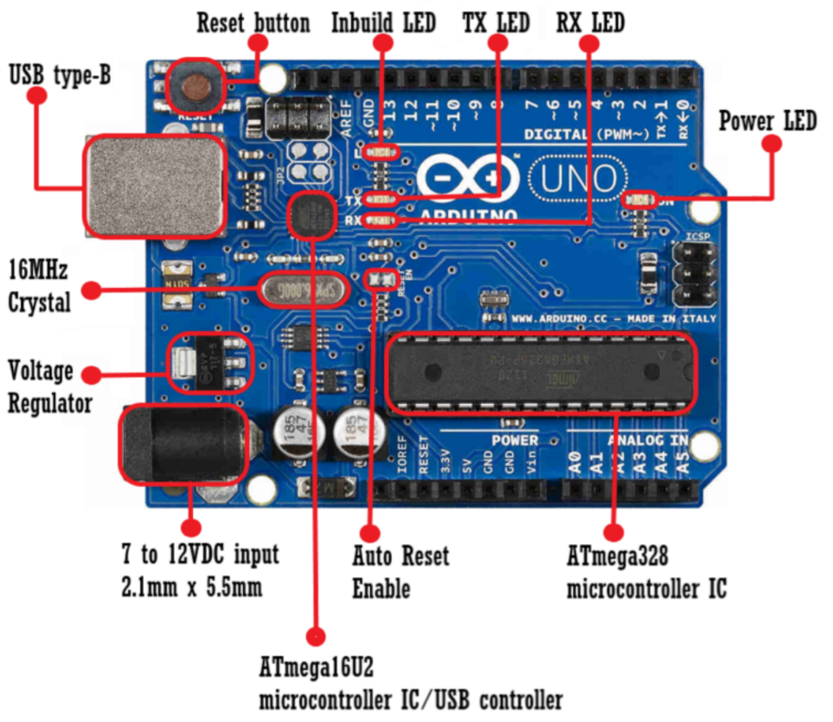

Reset button
Inbuild LED
TX LED
RX LED
USB type-B
Power LED
16MHz Crystal
Voltage Regulator
7 to 12VDC input 2.1mm x 5.5mm
Auto Reset Enable
ATmega328 microcontroller IC
ATmega16U2 microcontroller IC/USB controller

Figure 3.6: Arduino Uno R3 Pinout

- ATmega328 micro controller
  Heart of Arduino Uno R3. This micro controller unit executes instructions. Programs are stored inside this unit.

- ATmega16U2 micro controller
  Used to assist main micro controller. Placed near to USB port to decode USB information. Stands as a boot loader that write programs into main micro controller. The transmit LED (Tx LED) and receive (Rx LED) turns on whenever read or write operations are performed to micro-controllers.

- 16Mhz
  Act as heart beat of Arduino board. Serves as clock for timing various signals.

- Use type-B
  Used to interface Arduino to computer. Arduino board can be programmed via USB. Serial communication with board and serial monitor can also be achieved.

- 12V DC input
  The board can be powered via 12V DC adapter. The 12V is passed via capacitors to provide enough ampere and voltage across the board.

- Voltage regulator
  The digital circuits usually work at 5V. The 5V voltage regulator ensures that all the units gets proper voltage levels. If the board is successfully powered, the power LED glow brightly.

- Reset Button
  At time we might need to restart the board from the beginning. The reset button is used to reload the program from start. The reset can also be triggered inside the program.

Micro controller : ATmega328

Parameter	***Value***
CPU Type	8-bit AVR
Performance	20MIPS at 20MHz
Flash Memory	32 KB
SRAM	2 KB
EEPROM	1 KB
Pin Count	28 or 32 pins
Maximum operating frequency	20Mhz
Maximum I/O pins	23
External Interrupts	2
Board: Raspberry Pi	Board: Arduino

Table 3.2: ATmega328 spec

3.5 *Pin Layout*

Pins of Arduino can be broadly divided into two categories depending on their utility. For each pin, there is a marking on the board to denote the function of that pin. Pins are divided into:

1. General Purpose Input Output pins (total 20 pins)
   The functions of these pins can be programmed as per user need. They can either act as input pins or output pins. Input pins are those pins which Arduino would be listening for voltage variations. Output pins are those pins where Arduino controls the output voltage. These pins can also be classified into two categories. They are

   - Digital Pins
   - Analog Pins

2. Special Purpose pins (total 9 pins)
   The functions of these pins are predetermined and cannot be changed. They are reserved for special purpose. They include VCC (5v and 3.3v) , Vin, GND, RESET, IOREF, AREF, ICSP header.

   VCC pins provide a fixed output voltage, acting as a positive terminal of a cell. GND pin provide a fixed zero voltage. They act as the negative terminal of a cell. Any additional components that need to be interfaced with Arduino would surely be connected to GND pin. Vin pin is used to power up the Arduino. Depending on how the board is powered, Vin pin can also act as 5v VCC pin.

   RESET pin have the function similar to RESET push button. They can cause the program to restart from the beginning. IOREF and AREF stands for Input-output reference and Analog reference pin. They stand as a reference point to calculate voltages at digital and Analog pins. In-Circuit Serial Programing (ICSP) header pins are place close to micro-controllers. It is the ability of a micro-controller to be programmed without disconnecting from the circuitry.

3.6 Methods to power up Arduino

There are mainly three ways to power up the board.

The first method is to use UBS cable to power up the device. Just connect USB port to computer or a power bank. In this method the Vin pin can act as a 5V output pin. Make sure you don't draw much power so as the damage the port. The 12V DC jack should be kept free.

The second method is the power up via 12V DC jack. This will cause the board to have a bit higher current to handle components connect to it. The Vin pin act as a constant 5V DC output pin. In this method the USB port should to kept free. To make serial communication, make use of pins 0,1.

The third pin is to power 5V via Vin pin. The 12V jack and USB must be kept free. In this method, the board is likely to run on lower ampere.

3.7 Digital Pins

Digital pins are those pins that act on digital values. A positive 5V (above 2.5V) act as binary 1 or HIGH digital signal. A ground 0V (below 2.3V) act as binary 0 or LOW digital signal. Arduino being a digital system, all general purpose input-output pins can be used as digital pins. There are total of 20 pins that can be used for digital signals, ranging from pin 0 to pin13 and A0 to A5. To make use of digital signals we make use of functions like digitalRead() and digitalWrite().

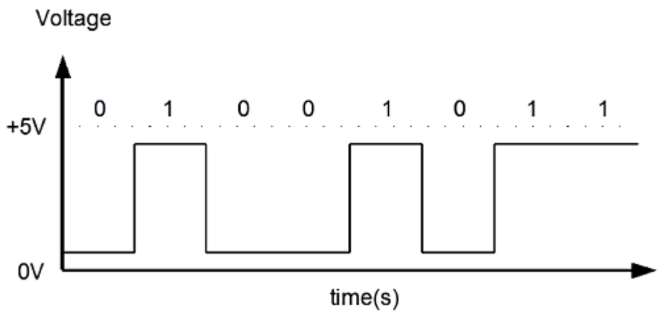


Figure 3.7: Digital signals

3.8 Analog Pins

The pins that support analog input are called analog pins. Analog signals are those signals that vary continuously. That is, they do not have a specific cutoff like digital signals. They can assume wide range of values. Temperature is an example of analog signals that vary continuously. Since Arduino is an digital circuit, the analog signals needs to be converted to digital signals for the micro controller to understand. This function is performed by Analog to Digital Conversion (ADC). ADC convert analog signal to 10bit digital value by sampling the signal and then mapping each sample to 2^{10} levels. There are a total of 6 pins in Arduino Uno that supports ADC, ranging from pins A0 to A5. To read analog value we make use of function analogRead().

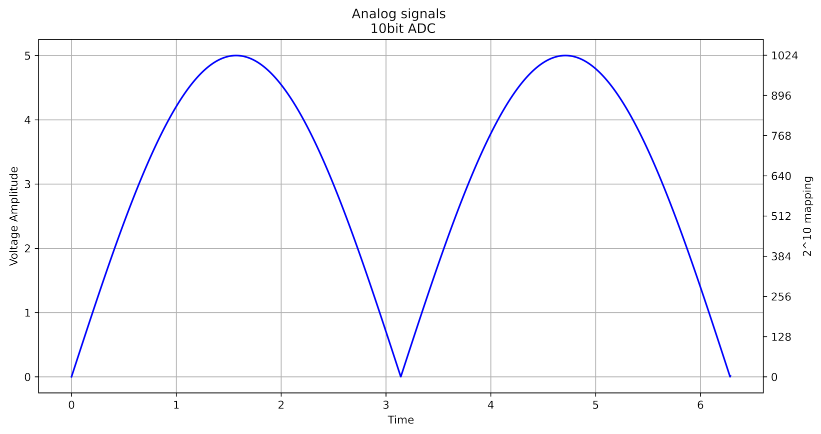


Figure 3.8: Analog signals

3.9 Pulse With Modulation (PWM)

PWM is the technique used to convert the digital signals to analog output results from Arduino. Arduino cannot directly produce various voltage levels. Being a digital circuit, it can only produce voltage levels

0V and 5V. To stimulate an analog effect, it creates square waves. The square wave have two key components: frequency and duty cycle. Frequency stays constant of about 500 Hz whereas the duty cycle is manipulated. Duty cycle refers to the amount of time signal stays high in a given time cycle. If Duty cycle is 100%, we would get an effect of 100% of 5V = 5V. If Duty cycle is 50%, we would get an effect of 50% of 5V = 2.5V.

Arduino Uno can support PWM on its 6 selected pins. They are pins 3, 5, 6, 9, 10 and 11. These pins have a tilde symbol (~) associated with their pin number on the board. The analog output function accepts 8 bit numbers, that are mapped to duty cycle. The 8 bit numbers can produce $2^8 = 256$ possible variations, spanning from zero to 255. The function analogWrite() is used to produce analog output from Arduino at PWM pins.

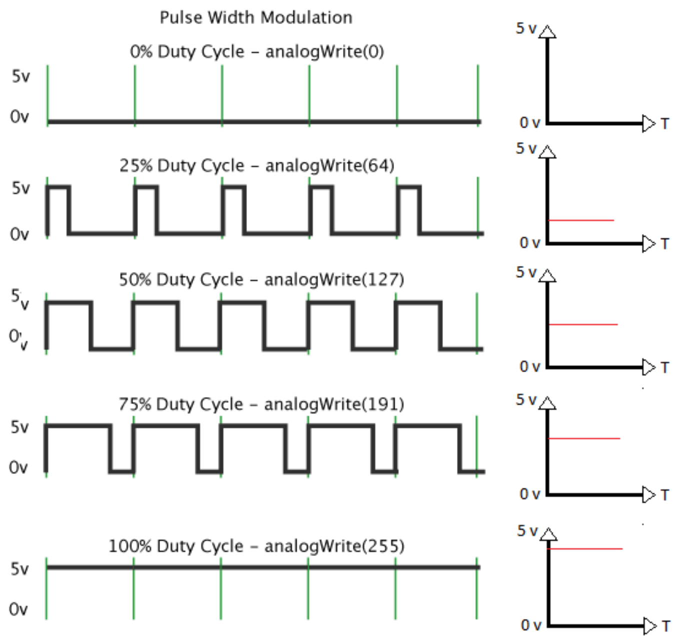

Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v
5 v
0 v
T

Figure 3.9: Pulse width Modulation

Table 3.3: ADC v/s PWM

ADC	*PWM*
Analog to Digital Converter	Pulse Width Modulation
Implements sampling on analog signals to convert to digital values	Uses width of the pulse (duty cycle) to convert digital to analog signals
10 bit resolution : input	8 bit resolution : output
Applicable only on A0 to A5	Applicable only on 3, 5, 6, 9, 10, 11
analogRead()	analogWrite()

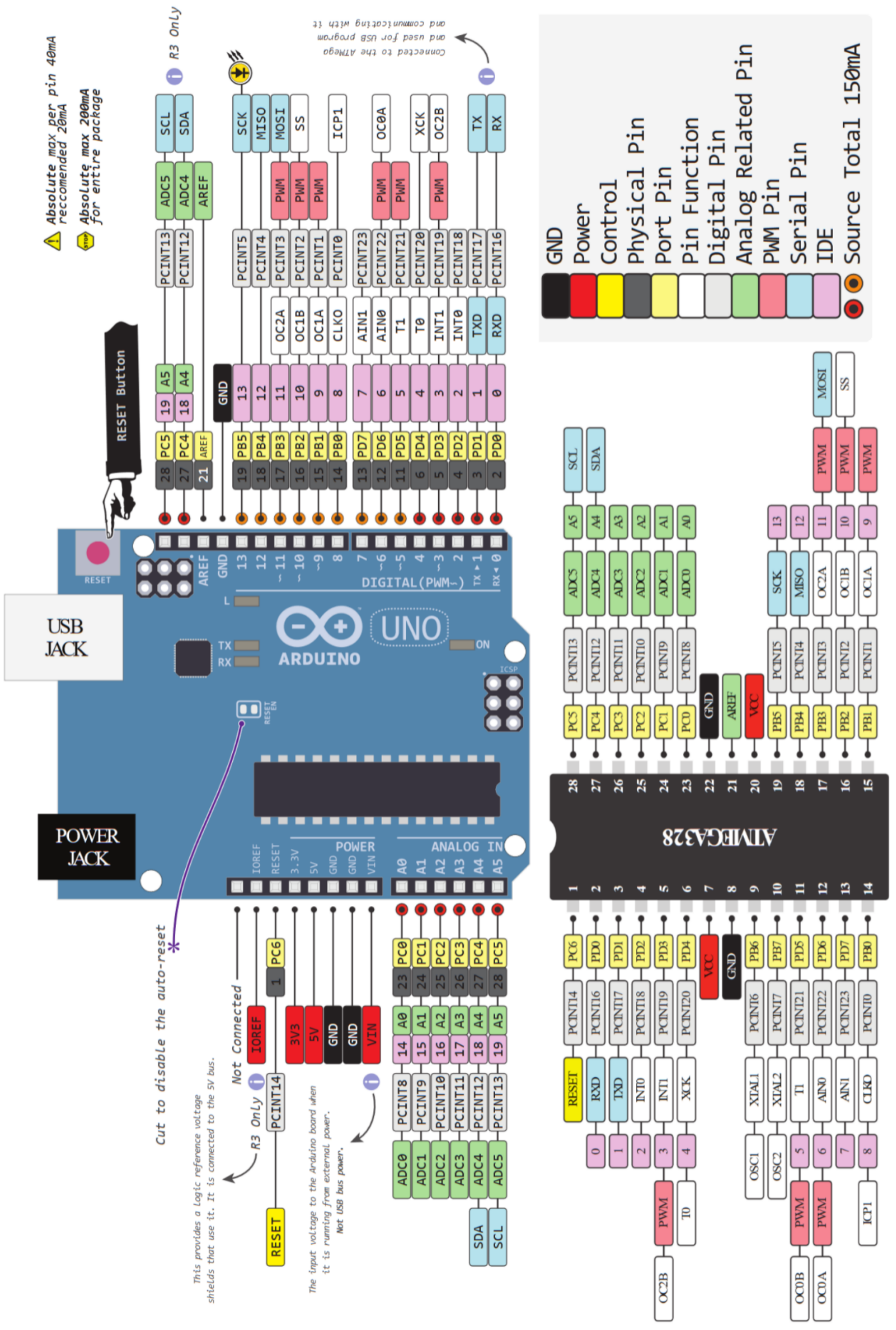

GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
RESET Button
USB JACK
POWER JACK
ARDUINO
UNO
ATMEGA328
R3 Only
Connected to the ATmega and used for USB program and communicating with it
Cut to disable the auto-reset
Not Connected
This provides a logic reference voltage shields that use it. It is connected to the 5V bus.
The input voltage to the Arduino board when it is running from external power. Not USB bus power.

Figure 3.10: Arduino Uno Datasheet

4
Programming Arduino with C

Arduino is it one of the most widely used micro-controller to develop various DIY projects. Developers design and assemble various sensors and other components to interact with environment. Being a controller, the board needs to be told and taught how, when, and where to communicate with other devices and environment. The process of instructing the controller what to do, step-by-step can be termed as programming. From being an art, programming is now an essential skill for any type of developer. To assist in developing various programs, developers have build various development tools to make programming easier and fun.

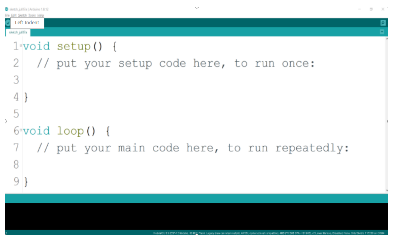
ARDUINO
Genuino
AN OPEN PROJECT WRITTEN, DEBUGGED, AND SUPPORTED BY ARDUINO.CC AND THE ARDUINO COMMUNITY WORLDWIDE
LEARN MORE ABOUT THE CONTRIBUTORS OF ARDUINO.CC on arduino.cc/credits

4.1 *Programming the voltages*

A programming language is a set of grammatical rules that is understood by a device. At the basic level, all digital systems work by making a large number of voltage level switches. These voltage switches occur between two levels, high volt and a low volt (possibly zero volt). These large numbers of switching are represented by an array of bi (two) level patterns called binary language. This is what the machine understands, called the first generation language. It is very tedious and error-sum to program in binary language (bunch of 1's and 0's). Hence small English like mnemonics were used to program, forming the second generation language - assembly language. They are converted to a binary pattern using an assembler. To make development easier, more English like languages were developed like C, C++. They form the high level language - third generation language. They are converted to binary patterns using compilers and interpreters. There are fourth generation and fifth generation languages developed, meeting the demands of complex calculation. We would be tinkering around third generation language - C programming language to program Arduino Uno.

4.2 *Arduino IDE*

IDE contains necessary tools required to write, compile and upload our code (sketch) to Arduino. It is a open source, cross-platform application that is written in Java. It supplies a software library which provides many common input and output procedures. There are various form of Arduino IDE - Arduino IDE software (windwos, linux, mac systems), Arduino Web IDE (runs on web browser), Arduinroid (Andriod App) that can be used to upload our sketch.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 4.1: Arduino IDE

Parts of Arduino IDE

1. Verify : Checks the C program for errors
2. Upload : Burns the compiled binary executable file into Arduino board
3. New Tab : Open a new workspace to code
4. Open : Opens new file
5. Save : save current works
6. Serial Monitor : Opens a new windows to conduct serial communications with the board
7. Sketch : The current program that is been edited
8. Code area : Areas to write code
9. Message area : View message of boards
10. Debug window : View status of boards
11. Connection Info : View current connected board configurations

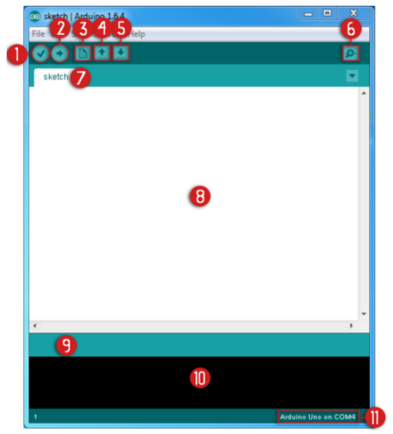
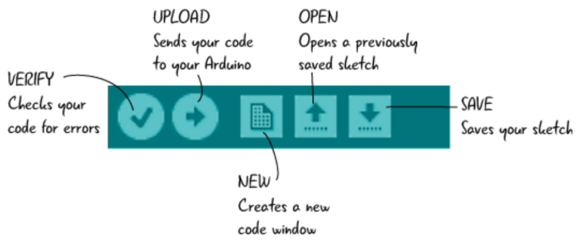


Figure 4.2: Parts of IDE

Arduino IDE : quick clicks

1. Install libraries
   They include code written by developers for certain sensors/usages.
   Tools -> Manage Libraries
   Shortcut : Ctrl + Shift + I
2. Verify / compile
   Sketch -> Verify/Compile
   Shortcut : Ctrl + R
3. Upload
   Sketch -> Upload
   Shortcut : Ctrl + U

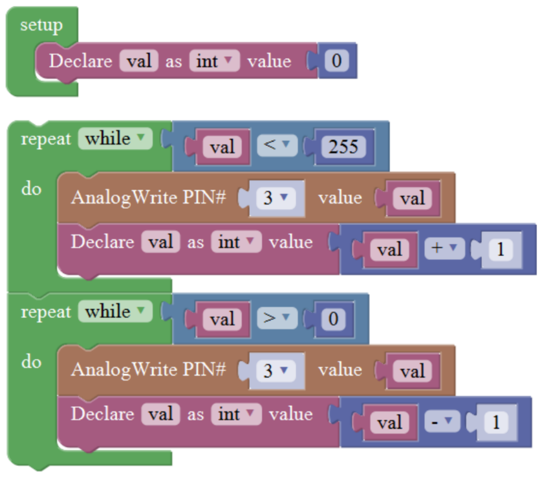

UPLOAD
Sends your code to your Arduino
OPEN
Opens a previously saved sketch
VERIFY
Checks your code for errors
SAVE
Saves your sketch
NEW
Creates a new code window

Steps to upload sketch to Arduino

1. Connect Arduino to PC
2. Select board port from Tools -> Port
3. Select board type from Tools -> Boards
4. Save your code
5. Verify your code
6. Upload your code
7. Wait until code uploads. Watch the message box for status information.

4.3 *Programming language for Arduino*

The official language for Arduino development is using the C/C++ programming language. C language is best known for its speed and close relationship with memory. C language can effectively make the most out of a given hardware optimally. The support for other programming languages is also being developed. Python language is gaining its support in Arduino. Take a look at https://realpython.com/arduino-python/ to get started. Those who find difficulty in learning language, there are block codes available. It uses simple GUI to drag and drop various code block segments that are joined together to form a program. The automatically generated code can then be uploaded to Arduino. Take a look at https://www.postscapes.com/iot-visual-programming-tools/ to get started. It would be best to write programs in C to get a clear understanding of the hardware and the program.



```
setup
  Declare val as int value 0

repeat while val < 255
do
  AnalogWrite PIN# 3 value val
  Declare val as int value val + 1
repeat while val > 0
do
  AnalogWrite PIN# 3 value val
  Declare val as int value val - 1
```

Figure 4.3: block coding

4.4 *Basic concepts of C language*

- Data types
  Data types refers to the type of value we deal with. There are various types of data in C. They include characters, integers, decimal numbers etc. Each data occupy specific space in the memory to store and process those information Table 4.1 shows few commonly used C data types and their memory consumption.

Data type	***What value does it hold***	***Example***	***Memory Space***	
char	Single characters	'a', 'b', 'Q', '1'	1 byte	8 bit
int	Integer numbers with no decimal parts	-32768 to 32767	2 byte	16 bit
long int	Same as int type, with longer precision	2147483, 647885425	4 byte	32 bit
float	Number with decimal point	2.312, 2.2258 etc	4 byte	32 bit
double	Same as float, with longer precision	2.59874, 69.88524589	8 byte	64 bit
long double	Same as double, with longer precision	658.8885421, 9986.22104	10 byte	80 bit

Table 4.1: Datatypes in C

- Variables
  Data are stored in specific memory location. These memory location names are difficult to handle. Hence we name those locations for easy access. These names are called as variables. They simply refers to a named storage location in the memory.
  General syntax :

```
	<data type> <variable name>;

```

  Example

```
	int number = 5;
	float pie = 3.14;

```

- Statements
  It refers to an instruction that instructs to do a specific task. In C language, statements are ended with semicolon(;) There are different types of statements. Some of them are conditional statement, iteration statements, jump statements.

- Functions
  They refers to a group/set of instructions referenced under one name. They increase program readability and is easy to extend a feature or

find errors. We can pass value to a function - called parameters. We are free to define our own functions. However, certain functions are defined by the system and are reserved.

- Conditional statements
  These statements directs the control of the program execution based on some conditions.
  General syntax:

```
        if( condition )
        {
            Statements to execute if condition is true
        }
        else
        {
            Statements to execute if condition is false
        }

```

- Looping statements
  These statements executes a set of statements until some condition holds true. There are three types of loops, namely - for loops, while loops and do while loops
  General syntax - for loop :

```
        for(initialization ; condition ; updation)
        {
            Statements to be executed
        }

```

  General syntax - while loop:

```
        while( condition )
        {
            Statements to be executed
        }

```

  General syntax - do while loop:

```
    do{
        Statements to be executed
    }while (condition);

```

- Arithmetic - Comparison - Logical operators
  Operators gives us the power to manipulate various data. Table 4.2 shows the available operators that can be used with numbers. Table 4.3 shows various comparison operators that can be used make decisions. At times, we would require more than one conditions to be satisfied. Table 4.4 show how we can club different conditions to create single decision.

Symbol	*Meaning*
+	Addition
-	Subtraction
/	Division
*	Multiplication
%	Modulus ->returns the remainder
++	Unary addition ->adds by one
–	Unary subtraction ->subtracts by one

Table 4.2: Arithmetic Operators

Symbol	*Meaning*
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Both are equal?

Table 4.3: Comparison Operators

Symbol	*Name*	*Meaning*
<condition>&& <condition>	AND	True if both conditions are True
<condition>| | <condition>	OR	True if any of the conditions are True
! <condition>	NOT	True if condition is false

Table 4.4: Logical Operators

Arduino specific functions and usages

Arduino have a bunch of functions that are used by the Arduino board for its functioning. Below are few of the commonly used functions. Additional function and their usages can be found at https://www.arduino.cc/reference/en/

- void setup()
  This is an unavoidable function. The statements written in this function are executed exactly once. Whenever Arduino starts (or reset button is pressed) the program execution starts from this function. Usually the declaration part and other configuration are written here, that needs to be executed only once. After its execution, it automatically calls the void loop() function.

- void loop()
  This is an unavoidable function. The statements written in this function are executed again and again. This part consist of the core part of the program. Statements like monitoring a sensor, reading pins etc are written here. The Arduino never stops its execution. On reaching the end of loop(), it would execute the same function again.

- pinMode()
  This function determines how the pins of the board needs to be interfaced. It configures general purpose input output pins as input mode or output mode. Prior to using the pin, every pin must have its mode defined using this function. It is usually written in void setup() function.
  General syntax:

```
        pinMode(<pin number>, INPUT | OUTPUT | INPUT_PULLUP );

```

  Example:

```
        //pin 13 is configured as output mode
        pinMode(13, OUTPUT);

```

- Serial monitor
  Serial monitor is a window that can be found in Arduino IDE, that facilitates serial communication with Arduino. Streams of data can flow from the monitor to Arduino and from Arduino to the system. For Arduino to listen for serial communication, Serial service must be initiated at a specific baud rate. Baud rate is the rate at which Arduino speaks to the system. Usually 9600 bits per second is set as the baud rate.
  Example : Printing to Serial monitor

```
int value = 55;

void setup(){
    //starting serial communication
    Serial.begin(9600);

    //Print a line to monitor and go to new line
    Serial.println("Device started");

    //Print a line to monitor and remain in same line
    Serial.print("value = ");

    //Print a data inside a variable
    Serial.println(value);
}
void loop(){
    //do nothing
}

```

- digitalRead()
  This function is used to read the digital state of a pin configured as INPUT mode. If the pin have a voltage above 2.5V, the function returns HIGH (= 1). If the pin have a voltage below 2.3V, the function returns LOW (= 0).
  General syntax:

```
        state = digitalRead(<pin number>);

```

  Example:

```
        //reads status of INPUT pin 12
        int state = digitalRead(12);

```

- digitalWrite()
  This function is used to set the voltage of an OUPUT mode pin to 5V or 0V. If the pin is set to HIGH, we would get 5V at the pin. If the pin is set to LOW, we would get 0V at the pin.
  General syntax:

```
        digitalWrite(<pin number>, HIGH | LOW);

```

  Example:

```
        //turn on in-build LED of Arduino Uno : set to 5V
        digitalWrite(13,HIGH);

        //sets the voltage level to 0V
        digitalWrite(12,LOW);

```

- analogRead()
  This function is used to read the analog values analog pin configured

as INPUT mode. Analog pin varies from A0 to A5. The analog to digital converter (ADC) maps the voltage at the input pin into a 10 bit number. Table 4.5 summarizes the voltage to value mapping of the function.

Voltage at analog input pin	*10bit value returned*
0V	0
1V	205
2.5V	512
5V	1023

Table 4.5: Analog to Digital mapping

General syntax:

```
        value = analogRead(<pin number>);

```

Example:

```
        //read 10bit mapped voltage of A2 analog input pin
        int state = analogRead(A2);

```

- analogWrite()
  This function is used to produce analog output values at the PWM pins configured as OUTPUT mode. PWM supported pins in Arduino Uno are 3, 5, 6, 9, 10, 11. The PWM technique maps the 8bit numbers to 5V voltage output. Table 4.6 summarizes the functions.

Value passed	*Duty cycle*	*Voltage experienced*
0	0 %	0V
50	19.92%	0.99V
127	50%	2.5V
255	100%	5V

Table 4.6: Digital to Analog Mapping

General syntax:

```
        analogWrite(<pin number>, <8bit number>);

```

Example:

```
        //Set the voltage at pin 5 as 2.5V
        analogWrite(5,127);

```

- delay()
  This function is used pause the execution of program for a defined time. It accepts an integer denoting the number of milliseconds to be paused.
  General syntax:

```
        delay(<milliseconds>);

```

Example:

```
        delay(1000);  //pauses for 1 second

```

- delayMicroseconds()
  This function is used pause the execution of program for a defined time. It accepts an integer denoting the number of microseconds to be paused.
  General syntax:

```
        delayMicroseconds(<microseconds>);

```

  Example:

```
        delayMicroseconds(10);  //pauses for 10 microsecond

```

There are a lot of additional functions defined for Arduino Development. Furthermore, developers around the world have contributed various additional methods to facilitate easy development. These works are compiled into small library, whose codes are made public. These libraries can be easily downloaded and made available via library manager in Arduino IDE. Additional methods are also written for various types of Arduino boards.

4.5 *Example Programs*

Blinking an LED

```
int LED=13;

void setup(){
    pinMode(LED,OUTPUT);
    Serial.begin(9600);
}

void loop(){
    digitalWrite(LED,HIGH);
    Serial.println("LED turned ON");
    delay(1000);

    digitalWrite(LED,LOW);
    Serial.println("LED turned OFF");
    delay(1000);
}

```

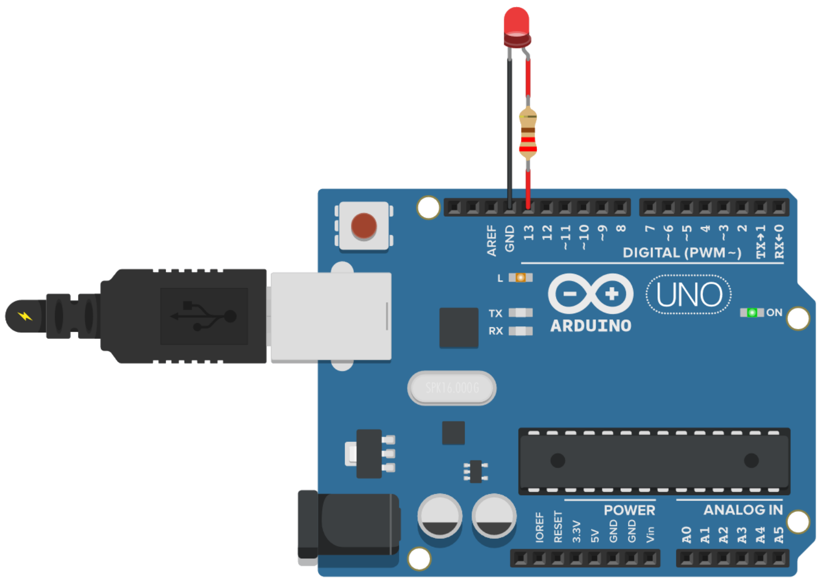


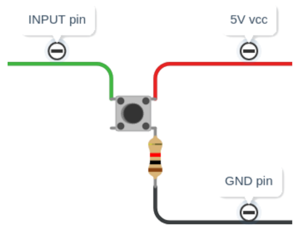
Figure 4.4: Led Blinking circuit

Interfacing Push Button

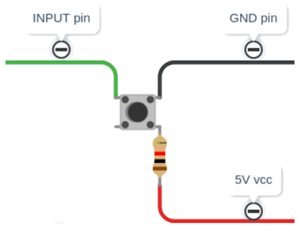
Buttons physically isolates the circuit. The circuit is closed only when the button is pressed. When the button is not pressed, the INPUT pin associated with that button feels that nothing is connected and in-circuit effects can affect the voltage. If the voltage falls below 2.3V, we will read low signal. If the voltage falls above 2.5V, we will read high signal. If the voltage stands between 2.3V and 2.5V, we cannot predict the output.

Hence push buttons are interfaced by building pullup circuits or pulldown circuit configuration as shown in figure 4.5 . In pullup configuration, if the button is not pressed, the 5V is connected to the pin. When the button is pressed, the pin will be connected to the ground (0V). The opposite happens in pulldown configuration. However, instead of building these circuits, we can implement the pullup circuit via the small internal resistor of the board. To avail such capability, we need to declare the pin as **INPUT_PULLUP** mode.

```
int button = 12;
int state;

void setup(){
    pinMode(button,INPUT_PULLUP);
    Serial.begin(9600);
    }
void loop(){
    state = digitalRead(button);
    if( state == LOW ){
        Serial.println("Button pressed");
    }
    else{
        Serial.println("Button not pressed");
    }
    delay(500);
}

```



(a) Pullup config



(b) Pulldown config

Figure 4.5: Button configuration

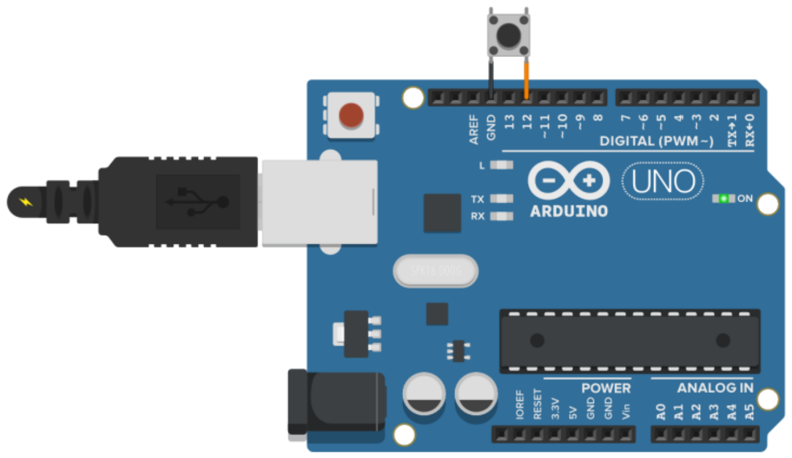


Figure 4.6: Push button circuit

Reading analog values : Interfacing potentiometer

```
int pot_pin = A0;

void setup(){
    pinMode(pot_pin, INPUT);
    Serial.begin(9600);
}

void loop(){
    int range = analogRead(pot_pin);
    Serial.print("Analog value: ");
    Serial.println(range);
}

```

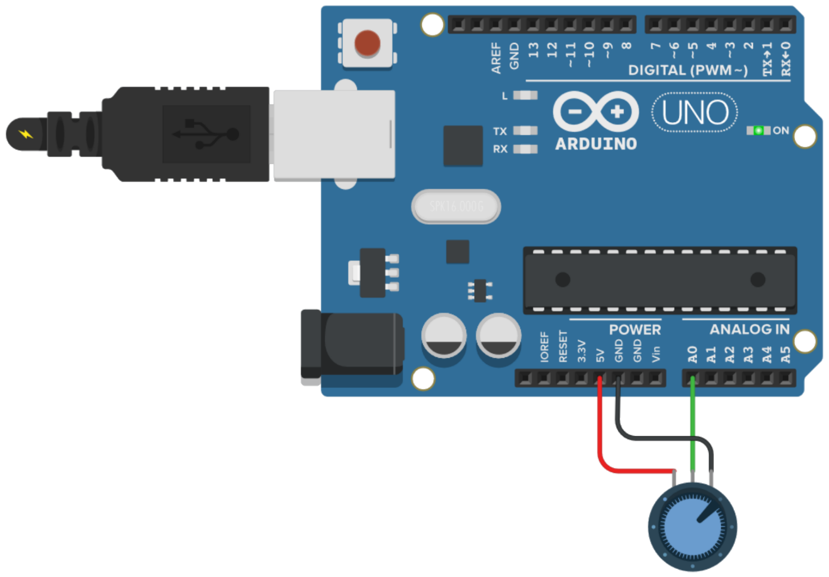


Figure 4.7: Potentiometer circuit

Writing analog values : Fading LED

```
int LED=11;
int bright;

void setup(){
    pinMode(LED,OUTPUT);
    Serial.begin(9600);
}

void loop(){
    Serial.println("Turning ON");
    for(bright = 0; bright<= 255; bright++){
        analogWrite(LED,bright);
        delay(500);
    }

    Serial.println("Turning OFF");
    for(bright = 255; bright>= 0; bright--){
        analogWrite(LED,bright);
        delay(500);
    }
}

```

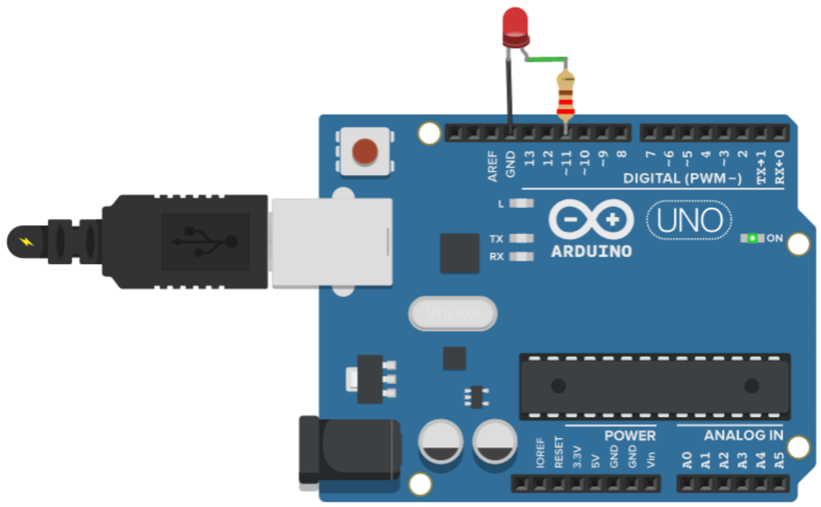
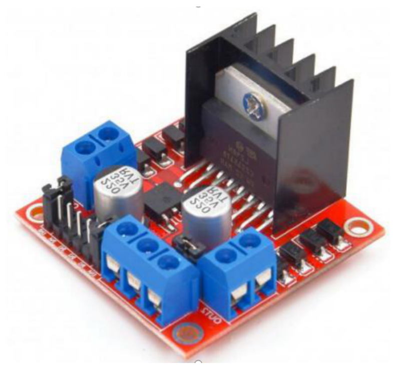


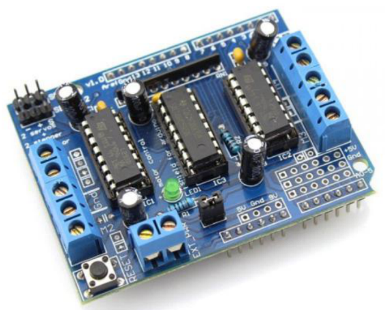
Figure 4.8: LED fading circuit

5
Motor Driver

As the name suggests, motor drivers are used to drive the motors. Motors consume higher current, more that what Arduino can supply. Hence they are interfaced with Arduino board using a motor driver. The motor driver consumes the power from external source and direct them to motors, controlled by Arduino. There are various motor drivers available. You might also able to find Arduino shields for motor drivers. A motor driver shield is much more powerful motor driver that can handle various DC motors, stepper motors, servo motors. In this session we would be talking about a simple motor driver - L298N



(a) L298N board



(b) 4 channel L298N Shield

Figure 5.1: L298N Motor driver

5.1 L289N motor driver

L298N motor driver is one of the many motor drivers available in the market. This driver have 4 out lines grouped into two as channel A and channel B. The working of the board is simple. Connect the external power source (12V) to the +12V pin and the negative terminal of the source to the GND pin. Since we need to interface it with Arduino, we need connect the GND pin of Arduino to the GND pin of the board. The logical signals are supplied from Arduino to the board on the pins input1 to input4. When any of the logical pins are high, corresponding

OUT pin turns on and gives an output of 10V. For example, if input3 is turned to HIGH, OUT3 would provide 10V and if input 2 is LOW, OUT2 would be at zero volt.

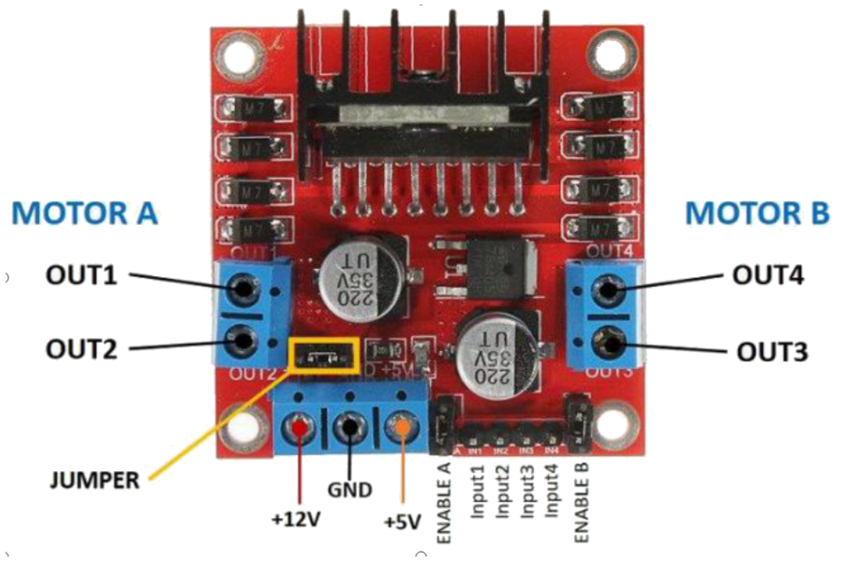


Figure 5.2: L298N PinOut

We can also find three jumper settings on the board. The jumper just above the power source (12V) connects the 5V pin via 5V regulator. The 5V pin is used to power IC on board. If the external power source is about 10 to 12 volts, we can keep this jumper on to power the board and motors using the 12V input. At this time, the 5V pin can act as an 5V output pin and can be used to power up the Arduino board. If we are providing external power supply greater than 12V, remove the jumper settings. The 5V regulator can get damaged. At this time, we would require additional 5V supply from Arduino to power the IC. L298N can handle up-to 35volts of external supply. Keep in mind that the board draws higher current and can exhaust your battery. Providing higher voltages can also cause the IC's to heat up rapidly. Make sure to cool them if they crosses the threshold.

The second jumper you can find are at either sides of the logical input pins. On the left side we have jumper for channel A and on right side for channel B. These pins actually control the speed of the motors. To adjust the speed of motor in each channel, remove the jumpers and connect PWM pins of Arduino to control the speeds. If you want to function the motors at full capacity, keep the jumper ON.

5.2 *Interfacing DC motor - motor driver - Arduino*

Let us connect the motor driver and Arduino. The driver would be powered up via external supply and Arduino will be powered via USB cable. In this sample connection , lets connect and 10V DC motor to each channels. The next question is, how to control the direction of the motors? Lets consider the channel A. The terminal of the motor is connected to the OUT pins.

INPUT 1	*INPUT2*	*OUT1*	*OUT2*	*Potential difference = OUT1-OUT2*	*Motor direction*
LOW	LOW	0v	0v	0V	NULL
LOW	HIGH	0v	10v	-10V	Anti-clockwise
HIGH	LOW	10v	0v	10V	Clockwise
HIGH	HIGH	10v	10v	0V	NULL

Table 5.1: The voltage difference experienced at each terminal can effect the rotation of motor

From the table 5.1, it is clear that by inverting the logical signals, we can change the direction of motor. The same can be applied at the channel B for direction control.

Now lets make the connections!

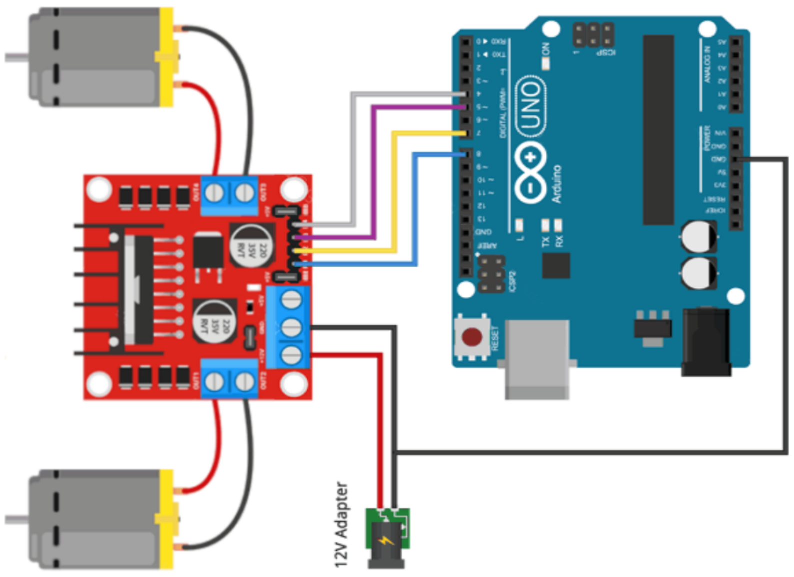


Figure 5.3: Interfacing Motor driver with Arduino

As shown in the figure 5.3, we have connected pins 8, 7, 5, 4 to input1, input2, input3 and input4 respectively. Both the boards shares the common GND and the driver is powered via 12V supply. Let's write a program that turns the motors clockwise and anti clockwise to control the bot movement. The bot movement are summarized in the table 5.2.

```
int m1 = 8, m2 = 7;  //left  motor pins
int m3 = 5, m4 = 4;  //right motor pins

void setup(){
    //set motor pins as output for Arduino.
    pinMode(m1,OUTPUT); pinMode(m2,OUTPUT);
    pinMode(m3,OUTPUT); pinMode(m4,OUTPUT);

    Serial.begin(9600);
}

//A function to control motor movement
void turn_motor(int input1, int input2, char dir){
    if( dir == 'F'){
        //clockwise rotation
        digitalWrite(input1,HIGH);
        digitalWrite(input2,LOW);
    }
    else if( dir == 'B'){
        //anti-clockwise rotation
        digitalWrite(input1,LOW);
        digitalWrite(input2,HIGH);
    }
    else if( dir == 'S'){
        //no rotation
        digitalWrite(input1,LOW);
        digitalWrite(input2,LOW);
    }
}

void loop(){

    turn_motor(m1, m2, 'F');  //left  wheel : clockwise
    turn_motor(m3, m4, 'F');  //right wheel : clockwise
    Serial.println("Bot moving forward");
    delay(1000);

    turn_motor(m1, m2, 'B');  //left  wheel : anti-clockwise
    turn_motor(m3, m4, 'F');  //right wheel : clockwise
    Serial.println("Bot turning left (rapidly)");
    delay(500);

    turn_motor(m1, m2, 'F');  //left  wheel : clockwise
    turn_motor(m3, m4, 'B');  //right wheel : anti-clockwise
    Serial.println("Bot turning right (rapidly)");
    delay(500);

    turn_motor(m1, m2, 'B');  //left  wheel : anti-clockwise
    turn_motor(m3, m4, 'B');  //right wheel : anti-clockwise
    Serial.println("Bot moving backward");
    delay(1000);

    turn_motor(m1, m2, 'S');  //left  wheel : stop
    turn_motor(m3, m4, 'S');  //right wheel : stop
    Serial.println("Bot stopped");
    delay(5000);
}
```

Table 5.2: Bot movement

INPUT1	INPUT2	INPUT3	INPUT4	Left Wheel	Right Wheel	Bot movement
HIGH	LOW	HIGH	LOW	Clockwise	Clockwise	Forward
LOW	HIGH	LOW	HIGH	Anti-clockwise	Anti-clockwise	Backward
LOW	LOW	HIGH	LOW	Stop	Clockwise	Left (slowly)
LOW	HIGH	HIGH	LOW	Anti-clockwise	Clockwise	Left (rapidly)
LOW	HIGH	LOW	LOW	Anti-clockwise	Stop	Left (slowly)
HIGH	LOW	LOW	LOW	Clockwise	Stop	Right (slowly)
HIGH	LOW	LOW	HIGH	Clockwise	Anti-clockwise	Right (rapidly)
LOW	LOW	LOW	HIGH	Stop	Anti-clockwise	Right (slowly)
LOW	LOW	LOW	LOW	Stop	Stop	Paused

5.3 *Speed controlled Bot*

You might have noticed that the bot drifts or becomes unstable when it changes it direction suddenly. This is because the wheels are rotating with higher speed than our bot can handle. The solution to such problems is the control the speed of the wheels. This can be easily achieved by making use of PWM pins in the motor driver. Lets try out our new speed controlled bot. It is easy to modify the above circuit and code to build the new bot quickly.

Re-configure the circuit by removing the jumper pins of channel A and B. Connect PWM pins 9 and 3 of Arduino to channel A and channel B respectively. You can identify PWM pins in Arduino by noticing the tilde symbol (~) on the board. The new circuit looks like figure 5.4. Modify the program for the new connection.

```
int pwmL= 9, m1= 8, m2= 7;  //left  motor pins
int pwmR= 3, m3= 5, m4= 4;  //right motor pins

void setup(){
    //set motor pins as output for Arduino.
    pinMode(m1,OUTPUT); pinMode(m2,OUTPUT);
    pinMode(m3,OUTPUT); pinMode(m4,OUTPUT);

    pinMode(pwmL,OUTPUT); pinMode(pwmR,OUTPUT);

    Serial.begin(9600);
}

//A function to control motor movement with speed regulation
void turn_motor(int in1, int in2, int PWM, int speed, char dir){
    //Setting the speed
    analogWrite(PWM, speed);

    if( dir == 'F'){        //clockwise rotation
        digitalWrite(in1,HIGH);
        digitalWrite(in2,LOW);
    }
    else if( dir == 'B'){ //anti-clockwise rotation
        digitalWrite(in1,LOW);
        digitalWrite(in2,HIGH);
    }
    else if( dir == 'S'){ //no rotation
        digitalWrite(in1,LOW);
        digitalWrite(in2,LOW);
    }
}

void loop(){

    //left  wheel : clockwise, speed=100%
    //right wheel : clockwise, speed=50%
    turn_motor(m1, m2, pwmL, 255, 'F');
    turn_motor(m3, m4, pwmR, 127, 'F');
    Serial.println("Bot moving forward with small right curve");
    delay(1000);

    //left  wheel : clockwise, speed=0%
    //right wheel : clockwise, speed=50%
    turn_motor(m1, m2, pwmL, 0, 'F');
    turn_motor(m3, m4, pwmR, 127, 'F');
    Serial.println("Bot moving left with low speed");
    delay(1000);
}
```

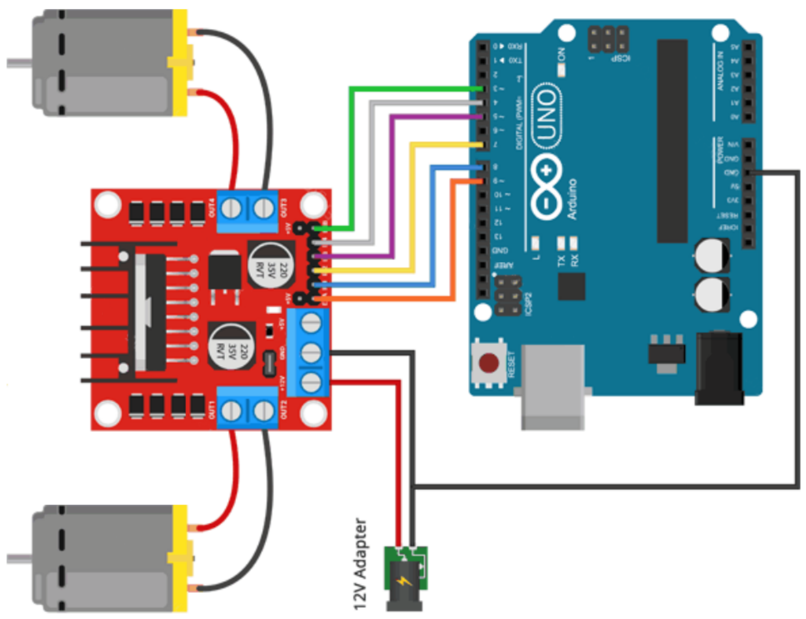


Figure 5.4: PWM supported wheel control

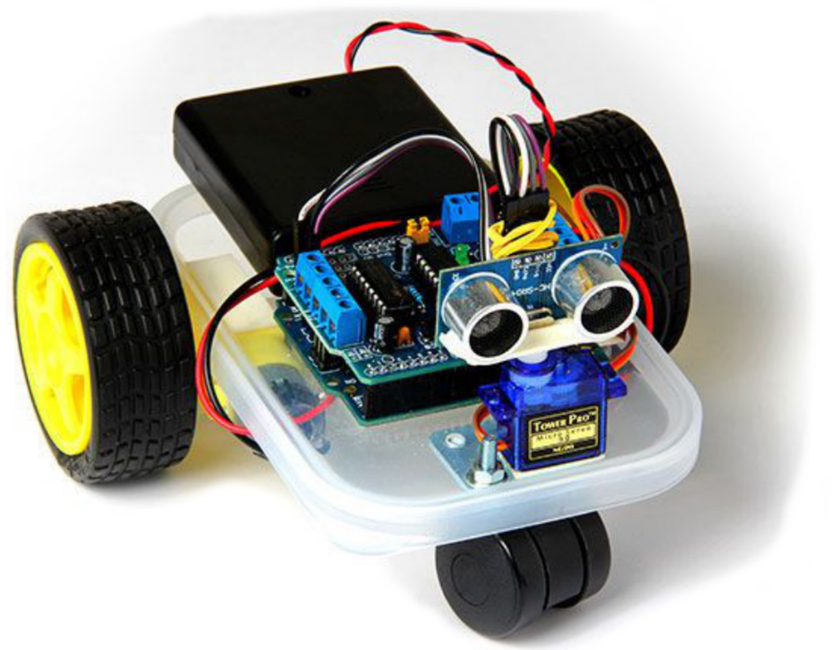
Keep in mind that PWM pins are 8-bit support pins. The Maximum analog value we can send is 255 i.e., $2^8 - 1$ (counting starts from zero).

If PWM value is set to 255, the motors will run at full capacity and would decrease as the value decreases. Do note that you are controlling the effective voltage sent to the motor and not the actual wheel rotation. To control the rotation of the wheel precisely, we would have to make use of stepper motors. Table 5.3 show the effective voltage felt at the motors.

PWM value	**Pulse width %**	**Effective voltage at 10V motor**
255	255/255 = 100.0%	100.0% * 10V = 10.0V
150	150/255 = 58.82%	58.82% * 10V = 5.88V
127	127/255 = 49.80%	49.80% * 10V = 4.98V
75	75/255 = 29.41%	29.41% * 10V = 2.94V

Table 5.3: Voltage experienced by the motors is in accordance with the effective voltage given by PWM pins

Try connecting few sensors like Ultrasonic sensors, Infra-Red (IR) sensor, Light Dependent Resistor (LDR), Accelerometer etc to make your bot autonomous and attractive!



6
Interfacing IR sensors

IR stands for Infrared Radiation, is an Electro Magnetic Radiation (EMR) with wavelengths longer than those of visible light. Discovered in 1800 by astronomer Sir William Herschel, IR is a region of the EMR spectrum where wavelengths range from about 700 nanometers (nm) to 1 millimeter (mm). They are longer than those of visible light, but shorter than those of radio waves. As they do not fall under the visible spectrum, they remain invisible to human eye. Certain animals and cameras can pick up those radiation and perceive them as images.

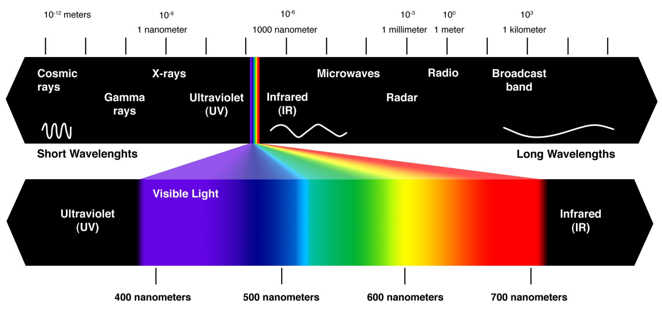


Figure 6.1: Infrared Spectrum

Based on their range of wave lengths, IR can be further classified into three regions. The Near Infrared regions spans from 700nm to 1400nm and is widely used in most of the IR sensors and fibre optics. The Mid infrared region spans from 1400nm to 300nm and is mainly used in heat sensing applications. The Far infrared region that spans from 300nm to 1mm is majorly used in thermal imaging. These different region are effectively used to build various application like night vision devices, infrared astronomy, infrared missile tracking etc.

6.1 *Types of IR sensors*

There are various IR sensors available in the market. Based on their configuration, we can classify them as Active IR sensors and Passive

Infrared (PIR) sensor . Active IR sensors are those sensor capable of both producing and sensing IR signals while PIR sensors mainly consist of detectors. Most of the motion detectors make use of PIR. The PIR detects the IR signals caused due to the heat energy transmitted by any object. In this chapter we would focus on Active IR sensor.

6.2 *Active IR Sensors*

An active IR sensor consist of both IR transmitter and IR receiver. The transmitter transmits the IR signals which would strike on an object and would bounce back to the receiver. However not all signals are bounced from the surface of the object. The bouncing of signals depends on the colour and material of the object. The dark colors have the ability to absorb more energy and transmit only a small portion of the received light. Light colors on the other hand, reflects most of the received light signals. It is this change of deflection of light that gives us the perception for colors. Since IR sensor is only capable of detecting the presence/absence of IR signals, they are employed to detect bright/dark surfaces. An IR sensor is not capable of differentiating various colors.

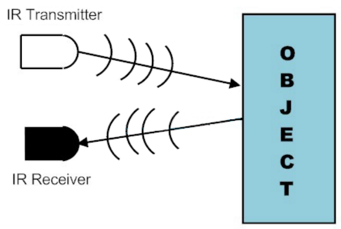
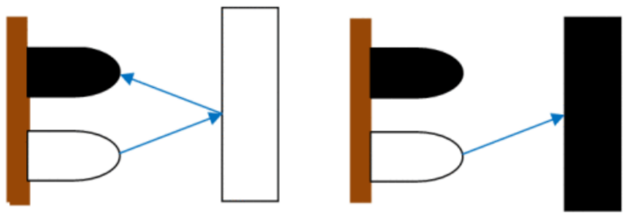


Figure 6.2: Behaviour of IR Sensor



(a) White surface (b) Black surface

Figure 6.3: Working of IR sensor

6.3 *IR sensor boards*

Now lets talk about the component required for a complete Active IR sensor. An IR sensor have two major LED that does the purpose of transmission and detection of signals. A transmission LED looks just like an normal LED diode. Upon applying sufficient voltage across the terminals, the LED produces signals which are transmitted in a straight line. A receiver LED act as an photo-diode that excite electrons upon receiving an IR signal. The receiver changes its resistance, which is used to detect the presence of IR signal. It is quite easy to identify both the LED. The IR receiver would be in dark color to prevent detection of surrounding IR signals. Note that the Sun emits a wide range of EMR, so it possible for IR receiver to detect the IR signals from the sunlight. Although it appears just like two LED, we would require proper circuitry to get a calibrated reading. Keep in mind that the receiver and transmitter LED need not always be on the same board, they can have separate circuits to function properly.

6.4 *Detailing IR sensor FC-51*

There are various IR sensor boards available in the market. You may choose any IR sensors suitable for your application. However the underlying principle of IR sensor are the same. Here we would make use of IR sensor board FC-51 to interface with Arduino.

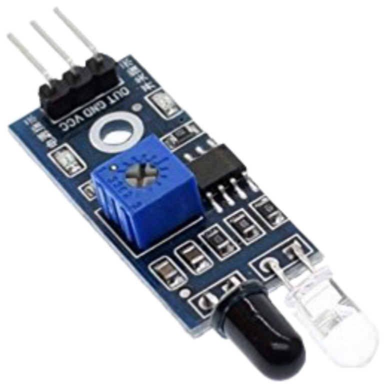


Figure 6.4: FC-51 IR Sensor board

A typical IR sensor board consist of both the transmitter and receiver diodes along with supporting circuitry which includes a potentiometer, an IC and a couple of resistors and LEDs. The potentiometer is used to adjust the sensitivity of the board. The more sensitive the board is, greater will be the amplification of weak signal detected. In other words, it would be able to detect IR signal from greater distance. The IC would amplify the change in the resistance of IR receiver LED and trigger corresponding voltage variations. We can also find two additional indicative LED on the board. One of the LED glows if the board is powered and the other LED glows when the board detects IR signals.

Now lets talk about the pins on the board. FC-51 have three legs for interfacing with Arduino. Each legs have there associated marking on the board to indicate what that leg is used for. The VCC pin indicates

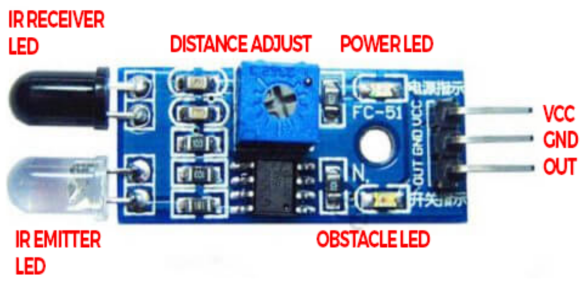


Figure 6.5: IR sensor board description

the power in for the board. The 5v supply from micro-controller is connected to the VCC and the ground (GND) of the board is connected to the GND pin of micro-controller. The OUT pin of the board would act as the input for the Arduino. The OUT pin gives out 5V upon detecting a bright surface and 0V upon detecting a dark surface. There do exist inverted boards that just detects the opposite! (see table 6.1) So keep in mind to check board you have before you start coding.

Type of board	Color of surface	OUT signal
Non inverted	Bright	5V(high)
	Dark	0V(low)
Inverted	Bright	0V(low)
	Dark	5V(high)

Table 6.1: Inverted and Non-inverted IR boards

Usually in an digital circuit, voltage below 2.3v is regarded as a low signal (0v | binary zero) and those above 2.5v as high signal (5v | binary 1). There do exist IR sensor boards that provide analog output reading. Make sure to understand the configuration of the board before interfacing with Arduino. With that, let start coding. Since FC-51 gives digital output (HIGH | LOW) we would be using digital pin of Arduino to interface.

6.5 *Code example 1*

Objective: Program to print the status of IR sensor to Serial monitor

```
int IR_pin = 2;    //connect OUT pin of IR to 2nd pin of Arduino

void setup(){

    pinMode(IR_pin,INPUT);     //IR as an input signal
    Serial.begin(9600);        //Baud rate
}

void loop(){

    //Reading the digital state of IR_pin
    int state = digitalRead(IR_pin);

    if( state == HIGH){
        Serial.println("Bright surface detected");
    }
    else{
        Serial.println("Dark surface detected");
    }

    //Slow down the code so that serial monitor
    // does not flood with characters
    delay(500);
}
```

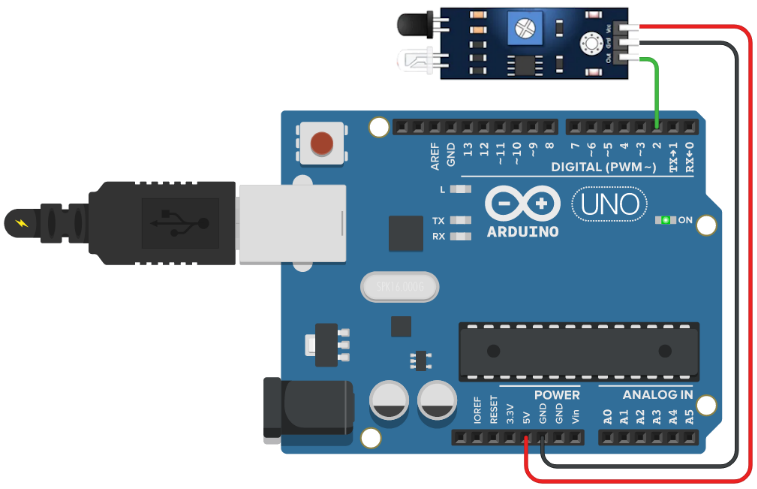


Figure 6.6: Interfacing IR with Arduino

Set the serial monitor at 9600 baud rate and see the results. We can find that when there is no object in front of sensor or in the presence of a dark object, the serial monitor shows "Dark surface detected". The monitor would show "Bright surface detected" when there is a white/reflective surface is introduced. Keep in mind that the sun light/flames also emit IR radiations that can be detected by IR sensors.

6.6 *Code example 2*

Objective: Program to turn on the in-build LED at pin 13 of Arduino Uno if IR sensor detects a while surface. Else keeps the LED off. The circuit of previous example can be used.

```
int IR_pin = 2;
int LED = 13;

void setup(){
    pinMode(IR_pin,INPUT);
    pinMode(LED,OUTPUT);
}
void loop(){
    int state = digitalRead(IR_pin);
    digitalWrite(LED, state);

    //alternatively use the single line code
    //digitalWrite(LED, digitalRead(IR_pin));

    delay(200);
}
```

6.7 *Line Follower bot*

Line follower bot is an simple bot that make use of IR sensors. The bot consist of IR sensors (usually an IR sensor array - figure 6.7), wheel and motors, motor driver, Voltage source all placed in a chassis. An IR array is a collection of 4 to 6 IR sensor receive and transmitter. We would be reading values from individual pair of sensors. Another major component is the motor driver. As the name suggests, they are used to drive motors. Jump to the chapter 5 to setup the bot - wheels and motors.

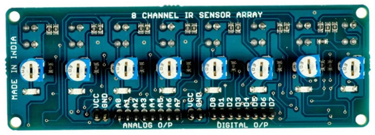


Figure 6.7: IR Array

Line follower bot follows a line to its destination. To detect the more precisely, the paths is formed by black lines in a white background or white path in black background. Usually the later is chosen as it is easy to construct black path in white background. For this simple bot, we would be making use of two IR sensors instead of an IR array. Our bot would have two wheels instead of four, although it is easy to extend to four wheels.

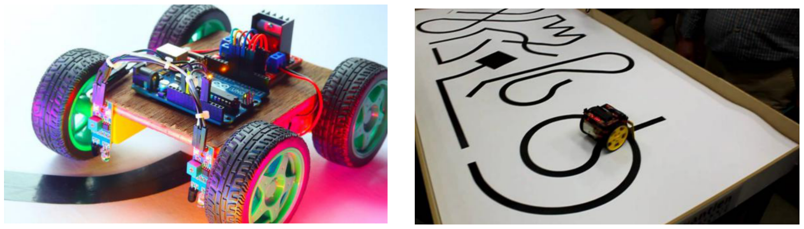

(a) IR Bot (b) Line follower path

Figure 6.8: Line follower - bot and path

6.8 *Tracing line - Line follower*

The main event of the line follower is to trace the line. In our sample environment, we would be using two IR sensors to detect the black line on the while surface. The bot is placed such that the black line moves right through the center of the bot. Both the sensors are places right next to the dark line.

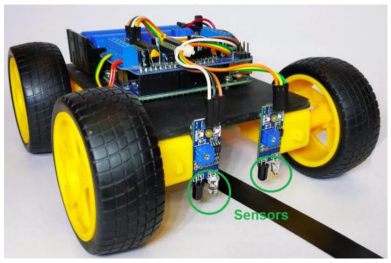
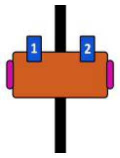
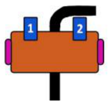
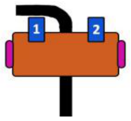
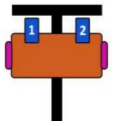


Figure 6.9: Line follower sensor alignment

Table 6.2 have listed all sort of possible combinations with 2 IR sensors. Now lets implement them in our bot. Take a quick peek at the section 5.2 where we have programmed a small bot. Lets add the additional IR sensors to them to complete our line follower.

Table 6.2: Line follower movement

Path property	Left IR	Right IR	Bot Motion	Scenario
Straight path	HIGH (white)	HIGH (white)	Forward	
Makes a right curve	HIGH (white)	LOW (black)	Turn right	
Makes a left curve	LOW (black)	HIGH (white)	Turn left	
Cross bar	LOW (black)	LOW (black)	Stop	

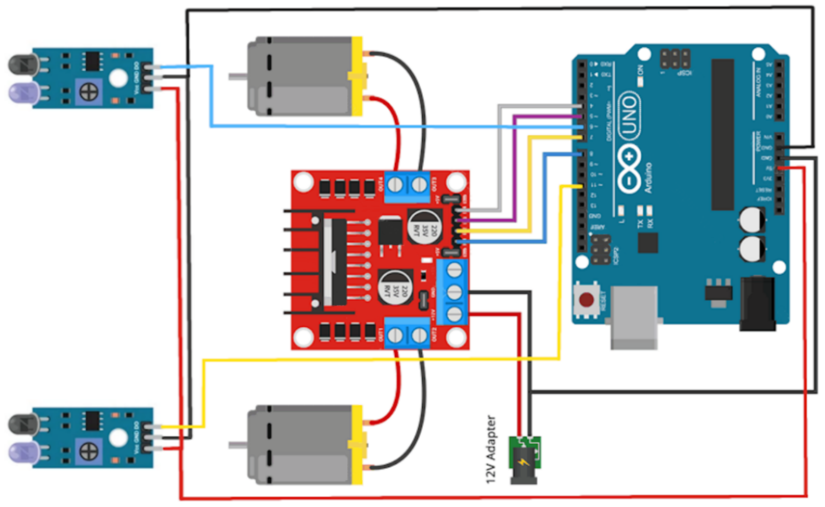


Figure 6.10: IR Line follower Circuit

Lets detail the connection of the circuit show in figure 6.10. The VCC and GND of IR sensors are connected to 5V and GND of Arduino. The left IR sensor (bottom) gives its output to 11th pin of Arduino while the right IR sensor (top) gives its output to 6th pin of Arduino. The GND of motor driver is connected to the GND of Arduino and input1, input2, input3, input4 of driver are connected to pins 8, 7, 5, 4 pins of Arduino respectively. Left motor is connected to the channel A and right motor is connected to channel B of the motor driver. Finally we powered the driver using a 12v adapter and connected the GND of driver and Arduino to it. Now lets implement the logic depicted on the table 6.2.

```
int m1 = 8, m2 = 7;     //left  motor pins
int m3 = 5, m4 = 4;     //right motor pins
int ir1 = 11, ir2 = 6; //left and right \ac{IR} sensor inputs

void setup(){
    //set motor pins as output for Arduino.
    pinMode(m1,OUTPUT); pinMode(m2,OUTPUT);
    pinMode(m3,OUTPUT); pinMode(m4,OUTPUT);

    //set IR pins as input for Arduino
    pinMode(ir1, INPUT); pinMode(ir2, INPUT);

    Serial.begin(9600);
}

//A function to control motor movement
void turn_motor(int input1, int input2, char dir){
    if( dir == 'F'){
        //clockwise rotation
        digitalWrite(input1,HIGH);
        digitalWrite(input2,LOW);
    }
    else if( dir == 'S'){
        //no rotation
        digitalWrite(input1,LOW);
        digitalWrite(input2,LOW);
    }
}

void loop(){
    //reading the IR sensors
    int left  = digitalRead(ir1);
    int right = digitalRead(ir2);

    if( left == HIGH && right == HIGH){//Forward motion
        turn_motor(m1, m2, 'F');
        turn_motor(m3, m4, 'F');
        Serial.println("Bot moving forward");
    }
    else if(left == HIGH && right == LOW){//Right motion
        turn_motor(m1, m2, 'F');
        turn_motor(m3, m4, 'S');
        Serial.println("Bot turning right");
    }
    else if(left == LOW && right == HIGH){//Left motion
        turn_motor(m1, m2, 'S');
        turn_motor(m3, m4, 'F');
        Serial.println("Bot turning left");
    }
    else{               //Stopping the bot
        turn_motor(m1, m2, 'S');
        turn_motor(m3, m4, 'S');
        Serial.println("Bot paused");
    }
    delay(100);
}
```

You might have noticed that the bot drifts or becomes unstable when it changes it direction suddenly. This is because the wheels are rotating with higher speed than our bot can handle. The solution to such problems is the control the speed of the wheels. Refer the section 5.3 to build speed controlled bot. You may also find that bot travels in a zig-zag fashion, the curves are not covered smoothly or the bot may miss some sharp curves. To avoid such problems, line followers are build using an IR array, where there are more sensors to detect the intensity of curves etc.

Line follower is one of the popular projects in Arduino-ROBOTICS. Various competitions are held across the world with varying difficulty and challenges. Maze solver is one of the most common event held. Speed, accuracy and reliability of bots are noted to announce the winner. To improve the probability of bagging the price, various statistical theories, mathematical models are applied and coded into the programs. Here we have detailed the basics of line follower. The rest is left to you to explore!

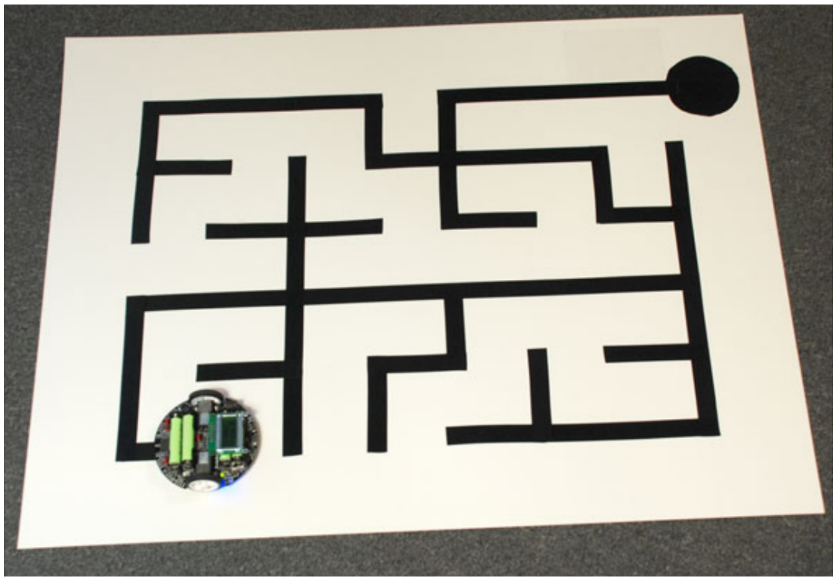


Figure 6.11: Line follower maze

7
Interfacing UltraSonic sensors

Defined by the American National Standards Institute, a ultra-sound are sound frequencies greater than 20Khz. They are normal sound waves which humans cannot hear, but some animals can. Humans ears can detect sound of frequencies in the range from 20Hz to 20KHz. Ultra sonic waves behave similar to normal sound waves i.e., they propagate at 340m/s and reflects upon striking a surface. They are widely employed in detection and raging, imaging, under water communication etc.

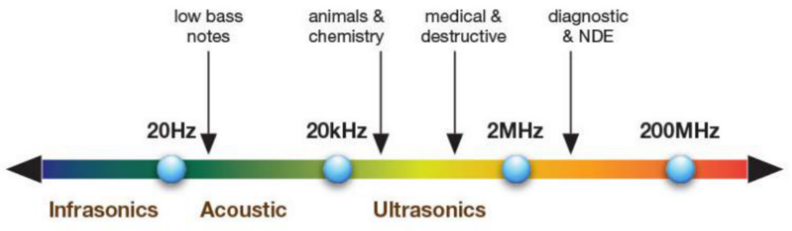


Figure 7.1: The Ultrasonic Spectrum

The production of ultrasonic waves requires a transmitter that transmit wave pulses, which are then received by a receiver. These two unites are imprinted in a board to be synchronous in achieving the task. The sensor is controlled by digital signals (pulses), produced/controlled by micro controllers or processors.

7.1 Ultrasonic sensor

There are various series of ultrasonic sensor available. They differ in their functionality on the projects. Few sensors have higher sensitivity, other few have more or less number of connection pins etc. The utility of each sensor is based on the projects they are used. Few sensors can be preferred more that other sensor in a particular project. In this session, we would be detailing about HC-SR04, one of the commonly used ultrasonic sensors.

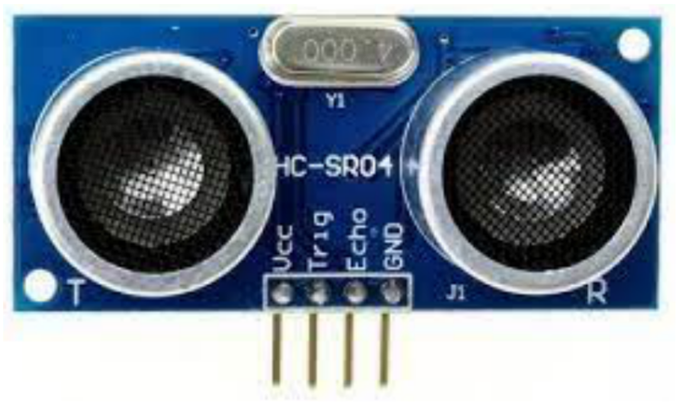


Figure 7.2: HC-SR04 Ultrasonic Sensor

7.2 *Detailing HC-SR04 ultrasonic sensor*

HC-SR04 is the most commonly used ultrasonic sensor. It is interfaced with Arduino via four pins. The board have a two cylinder like structures with a mesh/net on them. They are the transmitter and receivers made up of crystals of materials such as quartz that vibrate very fast when electricity is passed through them—an effect called "piezoelectricity." As they vibrate, they manipulate the air around them and the fluids they come in contact with, producing ultrasonic waves. In transmitter, the electricity is passed to the crystal to produce waves. In the receiver, the vibrating crystal produce small voltages that are detected and amplified. The cylinder that transmit waves have a small marking "T" at the edge and the receiver have a "R" marking at its edge. To synchronize the activity, the board has and 4Mhz oscillator to act as a timer. This specific board can detect an object if it is between 2-40cm range in front of the sensors.

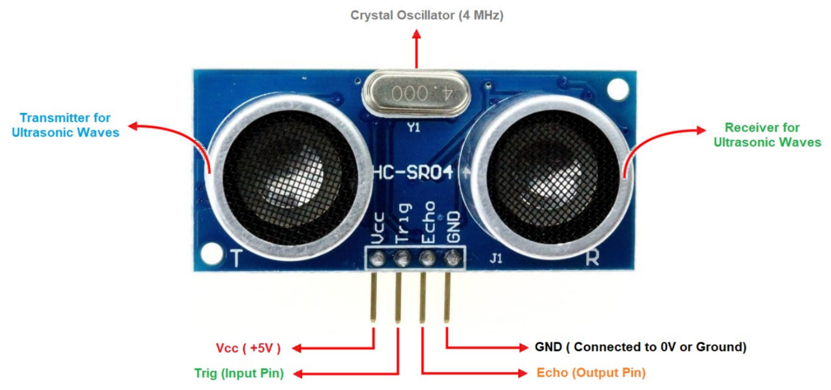
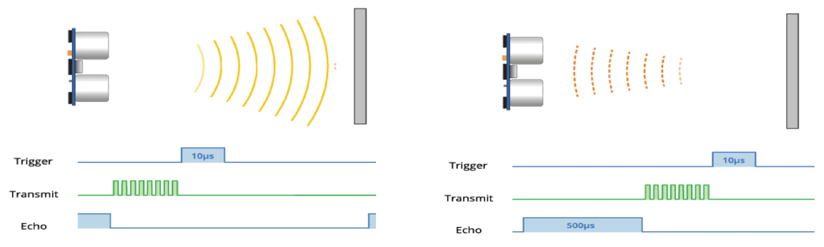

Crystal Oscillator (4 MHz)
Transmitter for Ultrasonic Waves
Receiver for Ultrasonic Waves
HC-SR04
Vcc (+5V)
Trig (Input Pin)
GND (Connected to 0V or Ground)
Echo (Output Pin)

Figure 7.3: HC-SR04 Pinout

7.3 Pins on HC-SR04

The board have 4 pins, they are VCC, TRIG, ECHO, GND. To power up the sensor, 5V and GND of Arduino are connected to the VCC and GND pins of the sensor. The TRIG is the input pin, that denote how long should the sensor produce ultrasonic waves. The ECHO is the output pin that denotes how long have we waited to receive the ultrasonic wave. Keep in mind that the input of the sensor will be connected to the output pin of Arduino and the output pin of sensor will be interfaced as input pin for Arduino. In short, the signal travels from sensor to Arduino in ECHO pin and from Arduino to sensor in TRIG pin.

7.4 Working of HC-SR04 sensor



(a) Transmission (b) Receiving

Figure 7.4: Ultrasonic waves bounces from surface

Initially we would put TRIG pin high for about 10 micro seconds. The sensor, upon receiving this signal, produce eight 40KHz pulses automatically. After producing short pulse burst, it would start the timer and pulls the ECHO pins to a high signal. It would then wait for the transmitted signal to reach back to the sensor. Upon receiving the first wave, it pulls the ECHO pin to low. We would record the time the ECHO pin remained high to calculate the distance between the sensor and the object.

7.5 Calculation of the distance

We know that

$$speed = \frac{distance}{time}$$

Or,

$$distance = speed * time$$

We have,

Speed of sound = 340m/s
Time can be fetched from the ultrasonic sensors.

Do note that the speed is in m/s and the time we receive is in μs. Lets formulate a formula to convert the units so that the distance can be measured in centi-meters (cm).

Let ' t ' denote the time we get from sensor in μs. Keep in mind that this time is the time taken by the waves to travel from sensors to the object and back to the sensors. In short, they have covered the distance twice. So the time required by the wave to travel from sensors to object is $t/2$ μs. Lets substitute the values in the formula

$$distance = 340m/s * \frac{t}{2}\ \mu s$$
$$distance = 340 * \frac{t}{2}\ \frac{m\mu s}{s}$$
$$distance = 170t\ \mu m$$
$$distance = 170t * 10^{-6}\ m$$
$$distance = 170t * 10^{-4}\ cm$$

$$\mathbf{distance = 0.017\ t\ cm}$$

7.6 *Code example 1*

Objective : To print the distance of a object from sensor in serial monitor.

```
int TRIG = 10;           //trigger pin of sensor
int ECHO =  9;           //echo pin of sensor
long duration;           //to store the time (micro seconds)
float distance;          //to store the distance (cm)

void setup(){
    pinMode(TRIG,OUTPUT);      //TRIG as output of Arduino
    pinMode(ECHO,INPUT);       //ECHO as input of Arduino
    Serial.begin(9600);        //Baud rate
}

void loop(){
    //wait for some time to clear the ultrasonic
    //waves if present in environment
    digitalWrite(TRIG,LOW);
    delayMicroseconds(2);

    //create ultrasonic burst
    digitalWrite(TRIG,HIGH);
    delayMicroseconds(10);
    degitalWrite(TRIG,LOW);

    //record the duration ECHO pin stood HIGH
    duration = pulseIn(ECHO,HIGH);

    //calculate the distance
    distance = 0.017 * duration;

    //print the distance
    Serial.print("The object is at ");
    Serial.print(distance);
    Serial.println(" cm");

    //Slow down the code so that serial monitor
    //does not flood with characters
    delay(1000);
}
```

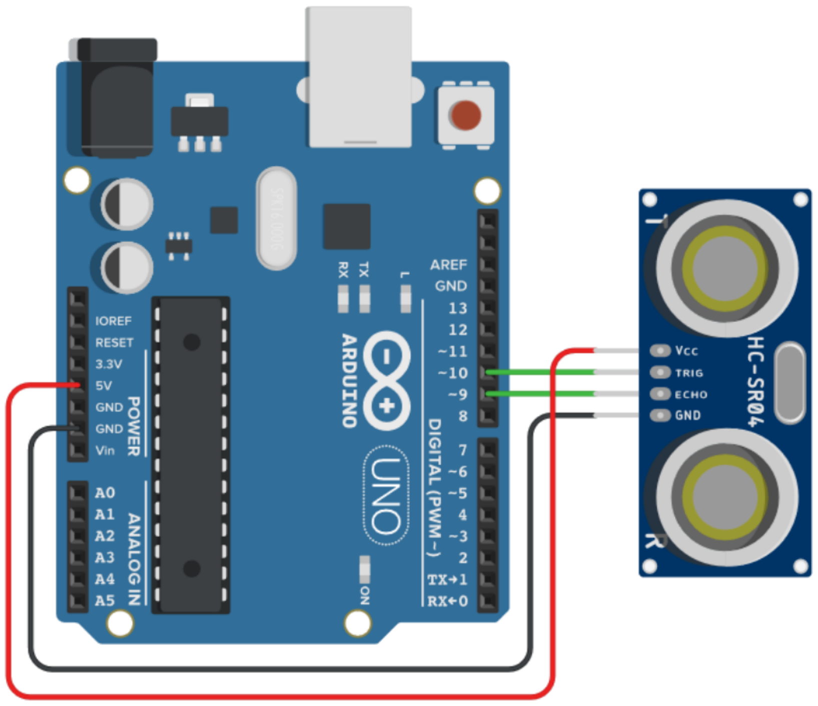


Figure 7.5: Ultrasonic Sensor with Arduino

It is always advised to wait for at least 2µs to clear the already existing ultrasonic wave in the environment. Try moving the hand in front of sensor to see varying distances. What would happen if we kept our hand on the sensor (i.e. 0cm distance)? Why does it show high value instead of printing zero? Keep in mind that the receiver needs to detect some ultrasonic waves to pull the ECHO pin down.

7.7 *Object avoider-bot*

Let us now try to create a bot that can avoid the objects in front of it and keep travelling. There are various models of such bot available. In this session we would be making use of two ultrasonic sensors and a bot (with chassis, motor, motor driver, wheel) to construct an object avoider bot. Get an peek into the chapter 5 to know various part of constructing a bot. We would be making use of bot build in the section 5.2 to construct our object avoider bot.

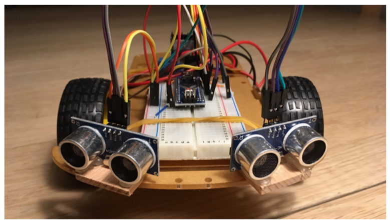
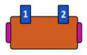
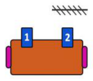
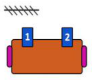
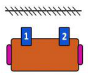


Figure 7.6: Obstacle Avoider Bot using Arduino Nano

We will be programming the bot to avoid the obstacle if it comes less than 10cm in front of the bot. The table 7.1 summarizes our flow. Here we have listed all sort of possible combinations with 2 IR sensors. Now lets implement them in our bot. Take a quick peek at the section 5.2 where we have programmed a small bot. Lets add the additional ultrasonic sensors to them to complete our obstacle avoider.

Table 7.1: Obstacle Avoider Motion

Left Ultrasonic sensor	**Right Ultrasonic sensor**	**Bot Motion**	**Scenario**
>10cm	>10cm	Forward	
>10cm	<=10cm	Turn left	
<=10cm	>10cm	Turn right	
<=10cm	<=10cm	Default:Rotate left	

Lets details the connections figure 7.7 . First connect motors to motor driver. Now lets interface motor driver with Arduino Uno. Connect the pins input1, input2, input3, input4 of motor drive to Arduino pins 8, 7, 5 and 4 receptively. The left ultrasonic sensor (top) have its TRIG and ECHO pins connected to 12 and 11 pins of Arduino. Connect its VCC and GND to Arduino 5V and GND respectively. The right ultrasonic sensor (bottom) have its TRIG and ECHO pins connected to 3 and 2 pins of Arduino. Connect its VCC to 5V pin of motor driver and GND to GND pin of Arduino. Each ultrasonic sensor needs separate 5V line. If there are no other 5V source, then you can set any unused pin of

Arduino to HIGH and extract 5V from that pin. Finally connect the 12V power line to 12V pin of motor driver and GND pin of motor driver to GND of Arduino and the negative terminal of the 12V source. Power the Arduino using USB cable. With that we have completed the circuits. Now lets build the code.

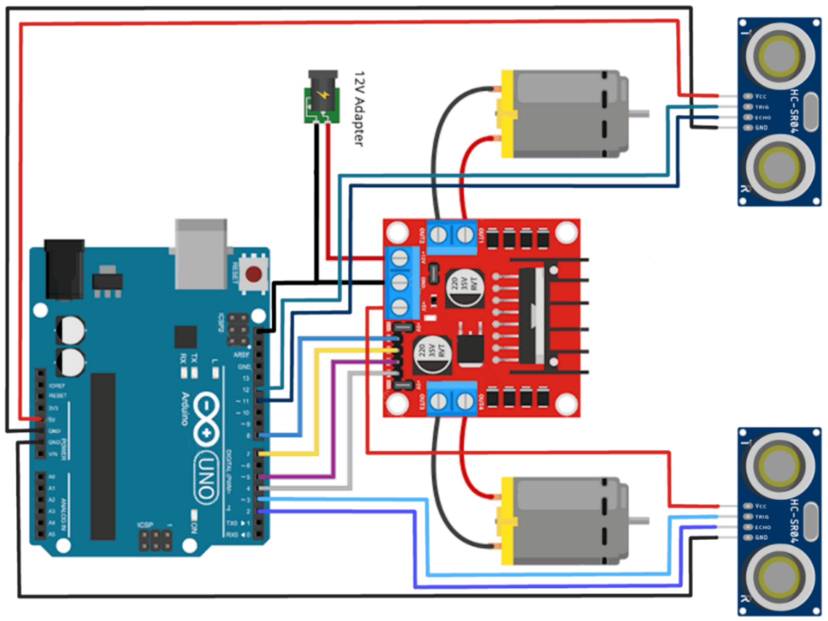
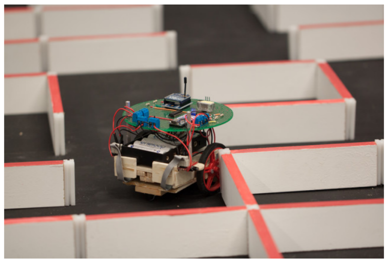


Figure 7.7: Obstacle Avoider Circuit

```
int m1 = 8, m2 = 7;              //left  motor pins
int m3 = 5, m4 = 4;              //right motor pins

int trig_L = 12, echo_L = 11;    //left ultrasonic sensors
int trig_R = 3, echo_R = 2;      //right ultrasonic sensors

float distance_L, distance_R;    //to store each distances
long duration;                   //to store time temporarily
float obj_dist = 10.0;           //threshold distance

void setup(){
    //set motor pins as output for Arduino.
    pinMode(m1,OUTPUT); pinMode(m2,OUTPUT);
    pinMode(m3,OUTPUT); pinMode(m4,OUTPUT);

    //set TRIG pins as OUTPUT and ECHO pins as INPUT
    pinMode(trig_L,OUTPUT); pinMode(echo_L,INPUT);
    pinMode(trig_R,OUTPUT); pinMode(echo_R, INPUT);

    Serial.begin(9600);
}

//A function to control motor movement
void turn_motor(int input1, int input2, char dir){
    if( dir == 'F'){
        //clockwise rotation
        digitalWrite(input1,HIGH);
        digitalWrite(input2,LOW);
    }
    else if( dir == 'B'){
        //anti-clockwise rotation
        digitalWrite(input1,LOW);
        digitalWrite(input2,HIGH);
    }
    else if( dir == 'S'){
        //no rotation
        digitalWrite(input1,LOW);
        digitalWrite(input2,LOW);
    }
}

void loop(){

    // reading value from left sensors
    digitalWrite(trig_L,LOW);
    delayMicroseconds(2);
    digitalWrite(trig_L,HIGH);
    delayMicroseconds(10);
    degitalWrite(trig_L,LOW);
    duration = pulseIn(echo_L,HIGH);
    distance_L = 0.017 * duration;

    // reading value from right sensors
    digitalWrite(trig_R,LOW);
    delayMicroseconds(2);
    digitalWrite(trig_R,HIGH);
    delayMicroseconds(10);
    degitalWrite(trig_R,LOW);
    duration = pulseIn(echo_R,HIGH);
    distance_R = 0.017 * duration;

    if( distance_L > obj_dist && distance_R > obj_dist ){
        //forward motion
        turn_motor(m1, m2, 'F');  //left  wheel : clockwise
        turn_motor(m3, m4, 'F');  //right wheel : clockwise
        Serial.println("Bot moving forward");
    }
    else if( distance_L > obj_dist && distance_R <= obj_dist ){
        //left motion
        turn_motor(m1, m2, 'S');  //left  wheel : stop
        turn_motor(m3, m4, 'F');  //right wheel : clockwise
        Serial.println("Bot moving left");
    }
    else if( distance_L <= obj_dist && distance_R > obj_dist ){
        //right motion
        turn_motor(m1, m2, 'F');  //left  wheel : clockwise
        turn_motor(m3, m4, 'S');  //right wheel : stop
        Serial.println("Bot moving right");
    }
    else if( distance_L <= obj_dist && distance_R <= obj_dist ){
        //by default, rotate left
        turn_motor(m1, m2, 'B');  //left  wheel : anti-clockwise
        turn_motor(m3, m4, 'F');  //right wheel : clockwise
        Serial.println("Bot rotating left");
        delay(1000);  //let the bot rotate a bit!!
    }
    delay(100);
```

```
}
```

Try out the same bot by controlling the speeds of wheels. Setup a dummy cardboard maze puzzle and try to solve the puzzle using the bot. You might need to make use of additional concepts to effectively navigates through the edges of walls. You could also rebuild the object avoider bot using single ultrasonic sensor and a servo motor. Put your thoughts and imaginations to get new ideas and solve new challenges.


(a) Maze solver

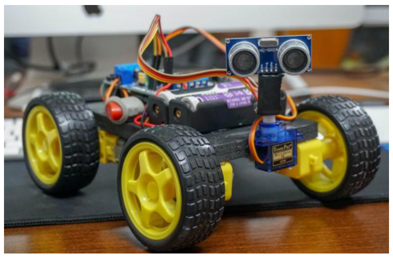

(b) Object Avoider

Figure 7.8: Different models and usage of object avoider

8
Interfacing DTMF

Dual Tone Multi Frequncy (DTMF) is a common method used by phone companies to conduct surveys and other requests. It is a telecommunication signaling system using the voice-frequency band over telephone lines between telephone equipment and other communications devices and switching centers. First developed at Bell Systems in United States in 1963, it found its wide spread usage ever since.

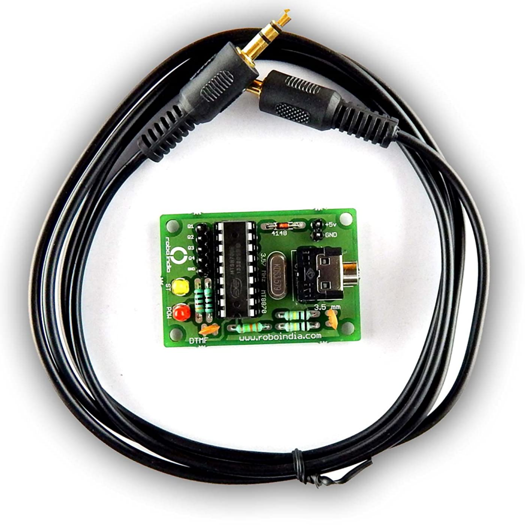


Figure 8.1: A DTMF module

The keypad is given a fixed set of pure tone (pure sine wave) that generates an audio of that particular frequency. This audio is analyzed at the switching centers and telephone equipment and decoded back to detect the key pressed. The keypad grid is divided into two groups of audio frequency ranges, The Row (Low group) frequencies and The Column (High group) frequencies. Whenever a key is pressed, corresponding mixture of audio are generated and transmitted via telephone line. Table 8.1 depicts how the frequency are distributed between rows and column.

		Column (High Group) frequencies			
		1209 Hz	**1336 Hz**	**1477 Hz**	**1633 Hz**
Row (Low Group) frequencies	**697 Hz**	1	2	3	A
	770 Hz	4	5	6	B
	852 Hz	7	8	9	C
	941 Hz	#	0	#	D

Table 8.1: Each key generates unique combination of frequency

For example if the customer presses 8 , audio of frequency 2188Hz (1336+852) is generated and transmitted via telephone line. At the receiving center, they are passed through a decoder to generate the key value 8.

8.1 Number systems - Binary and Decimal

In the digital world, number are represented in various forms. For humans to read smoothly, we in our day to day life make use of Decimal numbers. Decimal numbers have 10 possible digits, ranging from 0 to 9. All number are generated by placing these symbol in one's, ten's, hundred's places. Notice that the places are represented as powers of 10. To remind that the number is in decimal system and not in some other system, we write 10 as its subscript. Since decimal numbers are the mostly used, its automatically assumed that the number is in decimal form, if there are no subscript written. Table 8.2 depicts few examples of decimal numbers.

Decimal pattern	***Thousand's place*** 10^3	***Hundred's place*** 10^2	***Ten's place*** 10^1	***One's place*** 10^0
$(1598)_{10}$	1	5	9	8
$(559)_{10}$		5	5	9
$(48)_{10}$			4	8
7_{10}				7

Table 8.2: Decimal number system

Consider a system that can accept N number of decimal digits (places), it means that the system can have $(base)^N$ possible values i.e., 10^N possible values ranging from 0 to $10^N - 1$ numbers. Assume we can accept any 4 digit decimal number. It means that there can be $10^4 = 10000$ values ranging from 0 to 9999

Computers are digital systems where signal/states are represented by 0V or 5V. They have only 2 states possible. To execute any instruction, they require an array of 2 states called "bi-nary". Each state are called as a **bit**. Bit zero represents 0V and bit one represents 5V. To work upon decimal numbers, digital system needs to convert and represent decimal number as its equivalent binary numbers.

Consider a 3 bit binary number. There can be total of $2^3 = 8$ possible values, ranging from 0 to 7 as shown in the table 8.3.

Binary pattern	2^2	2^1	2^0	***Decimal equivalent***
$(000)_2$	0	0	0	0
$(001)_2$	0	0	1	1
$(010)_2$	0	1	0	2
$(011)_2$	0	1	1	3
$(100)_2$	1	0	0	4
$(101)_2$	1	0	1	5
$(110)_2$	1	1	0	6
$(111)_2$	1	1	1	7

Table 8.3: A 3 bit binary to decimal mapping

You might have heard of byte, kilo byte, megabyte etc. They denote range of collection of bits to represent an information. If "int" is 4 bytes long, it means it can have $4 * 8 = 32 bits = 4294967296$ possible values. See table 8.4

4 bit	=1 nibble	2^4 = 16 combinations
8 bit	= 1 byte	2^8 = 256 combinations
2^{10} byte	= 1 kilobyte	2^{18} = 262144 combinations
2^{10} kilobyte	= 1 megabyte	2^{28} = 268435456 combinations
2^{10} megabyte	= 1 gigabyte	2^{38} = 274877906944 combinations

Table 8.4: Computer memory capacity

8.2 *Converting Decimal to Binary*

To convert a decimal number to its equivalent binary number, keep dividing the number by 2 and append the remainders in reverse order (bottom to top) as shown in figure 8.2.

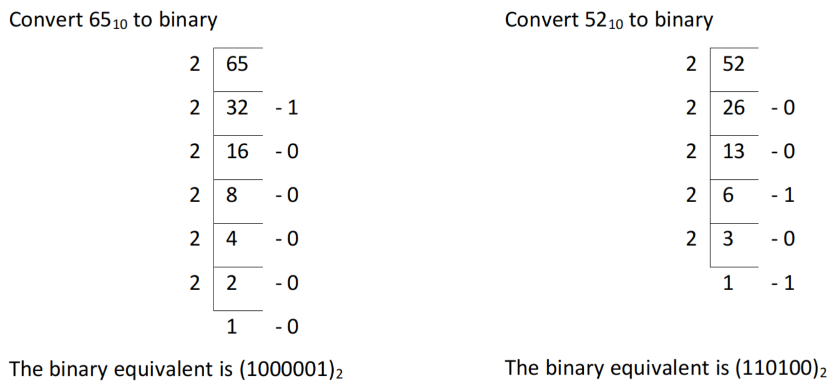

Convert 65_{10} to binary
2 | 65
2 | 32 - 1
2 | 16 - 0
2 | 8 - 0
2 | 4 - 0
2 | 2 - 0
1 - 0
The binary equivalent is $(1000001)_2$
Convert 52_{10} to binary
2 | 52
2 | 26 - 0
2 | 13 - 0
2 | 6 - 1
2 | 3 - 0
1 - 1
The binary equivalent is $(110100)_2$

Figure 8.2: Converting to binary

8.3 *Converting Binary to Decimal*

To convert binary number to its equivalent decimal number, keep multiplying the binary digit with 2^p , where p is digit position starts from zero and then find its sum as depicted in table 8.5.

Position values	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	***Decimal value***
Binary pattern1 = $(1011)_2$	1	0	1	1	$1{*}2^3 + 0 * 2^2 + 1{*}2^1 + 1{*}2^0 = 11_{10}$
Binary pattern2 = $(110)_2$		1	1	0	$1{*}2^2 + 1{*}2^1 + 0 * 2^0 = 6_{10}$

Table 8.5: Converting to decimal

8.4 *MT8870 DTMF Decoder*

DTMF decoder is a module the accepts the audio frequency and converts them to the digital signals. The audio can be fetched from a mobile phone via aux cable (headphone set) and connect it to the aux jack of the module.

The board needs to be powered up via 5V pins and the decoded output is received via pins Q1,Q2,Q3,Q4 in binary format. Table 8.6 shows the mapping of key pressed from binary to decimal value.

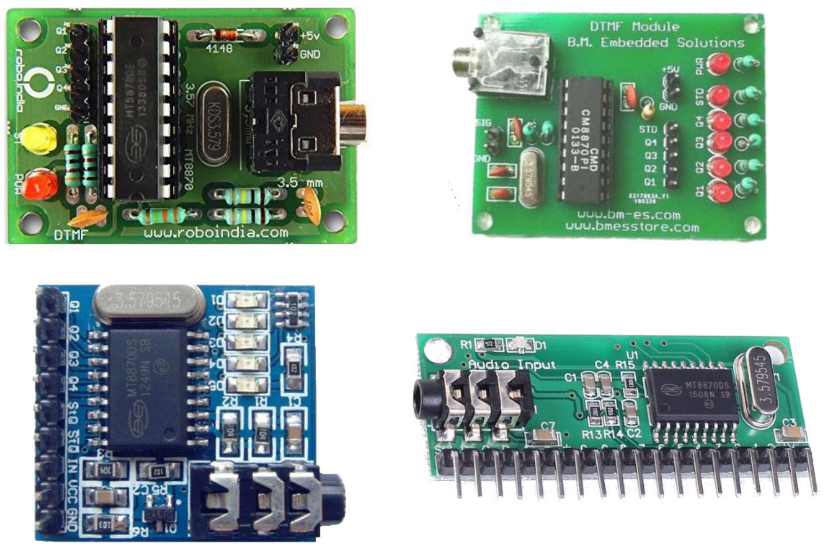


Figure 8.3: Various models of DTMF decoder

Key pressed	Audio frequency[Hz]	Q4	Q3	Q2	Q1	Binary pattern	Decimal equivalent
1	1906	0	0	0	1	1	1
2	2033	0	0	1	0	10	2
3	2174	0	0	1	1	11	3
4	1979	0	1	0	0	100	4
5	2106	0	1	0	1	101	5
6	2247	0	1	1	0	110	6
7	2061	0	1	1	1	111	7
8	2188	1	0	0	0	1000	8
9	2329	1	0	0	1	1001	9
0	2150	1	0	1	0	1010	10
*	2277	1	0	1	1	1011	11
#	2418	1	1	0	0	1100	12
A	1906	1	1	0	1	1101	13
B	2033	1	1	1	0	1110	14
C	2174	1	1	1	1	1111	15
D	1979	0	0	0	0	0	0

Table 8.6: Mapping DTMF inputs to decimal values

8.5 *Programming DTMF decoder*

As shown in figure 8.4, the q1,q2,q3,q4 of DTMF is connected to 7,6,5,4 pins of Arduino respectively. The StD pin is connected to pin3, which turns HIGH whenever the module receives a new frequency. The module is powered via 5V line and is connected to a cell phone via aux cable. The Phone acts as receiver/switch center. Lets write a code to interpret the DTMF signals.

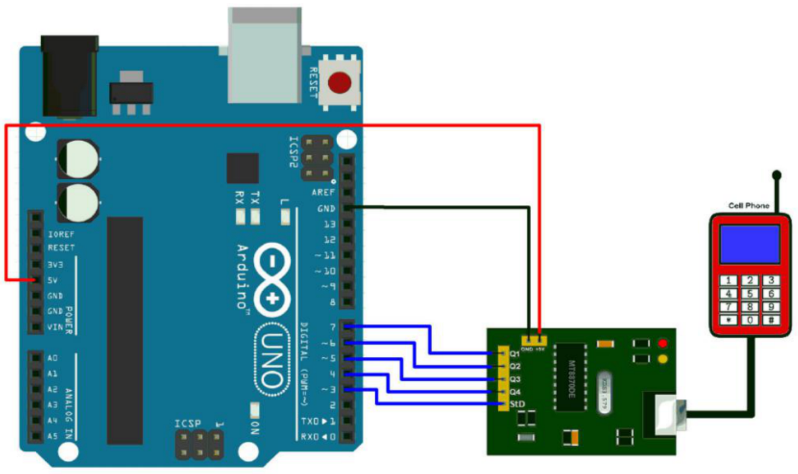


Figure 8.4: Circuit diagram for interfacing DTMF with Arduino Uno

```
int Q1=7, Q2=6, Q3=5, Q4=4, StD = 3; //pin allocations
int dec_number;

void setup(){
    //set pins as input for Arduino.
    pinMode(Q1,INPUT);pinMode(Q2,INPUT);
    pinMode(Q3,INPUT);pinMode(Q4,INPUT);
    pinMode(StD,INPUT);

    Serial.begin(9600);
}

void loop(){
    dec_number = 0;

    if(digitalRead(StD)==HIGH){ //received new frequency

        //converting binary signals to decimal number
        //using bit manipulation in C
        dec_number |= digitalRead(Q1)<<0 ;  //2^0 position
        dec_number |= digitalRead(Q2)<<1 ;  //2^1 position
        dec_number |= digitalRead(Q3)<<2 ;  //2^2 position
        dec_number |= digitalRead(Q4)<<3 ;  //2^3 position
        Serial.print("The code is ");
        Serial.println(dec_number);
    }
}
```

Now make a call to the receiver phone **"A"** from another phone **"B"**. Make key presses in the phone B so that corresponding audio is heard at the phone A. Observe the Serial monitor for the output. It might happen that you cannot observe proper detection in the Serial monitor. This can be due to various factor like different audio jack material, internal phone circuitry, phone model and version etc. Try with some other phones. Have you ever noticed that whenever we press the button on earphone mic, it behaves differently in different phones?

Can you reconfigure the system to control a bot? Go through the section 5.2 to get an idea to integrate the bot with DTMF. Your wireless bot is ready for action.

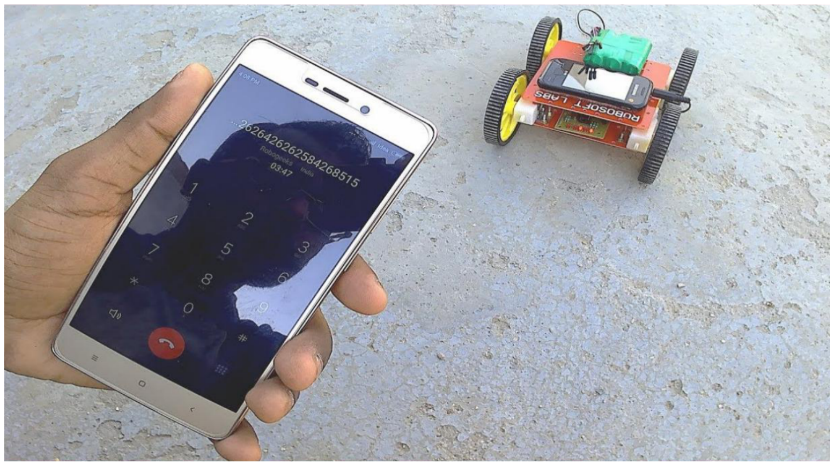


Figure 8.5: Wireless bot using DTMF

Conclusion

Robotics is indeed a vast field that experience rapid advancements every day. It finds its usages ranging from home appliances to commercial factories. Technology have made possible for a large population of enthusiasts to test and build new ideas to uplift human lifestyle. Introducing Arduino and its working can stand as foundational steps into robotics. IoT and embedded project contributes heavily to automation of various facilities. A wide range of projects can be found at Hackster `https://www.hackster.io/` that make use of basic units to build amazing ideas. Higher to Arduino, Raspberry Pi scores the most when it comes to higher IoT and other projects. Image processing, Voice controller applications, surveillance, automation etc demands Open Source platforms like raspberry pi. Other software's and advancements keeps on progressing as we speak! As the technology improves, there will be new ways to use robots which will bring new hopes and new potentials.