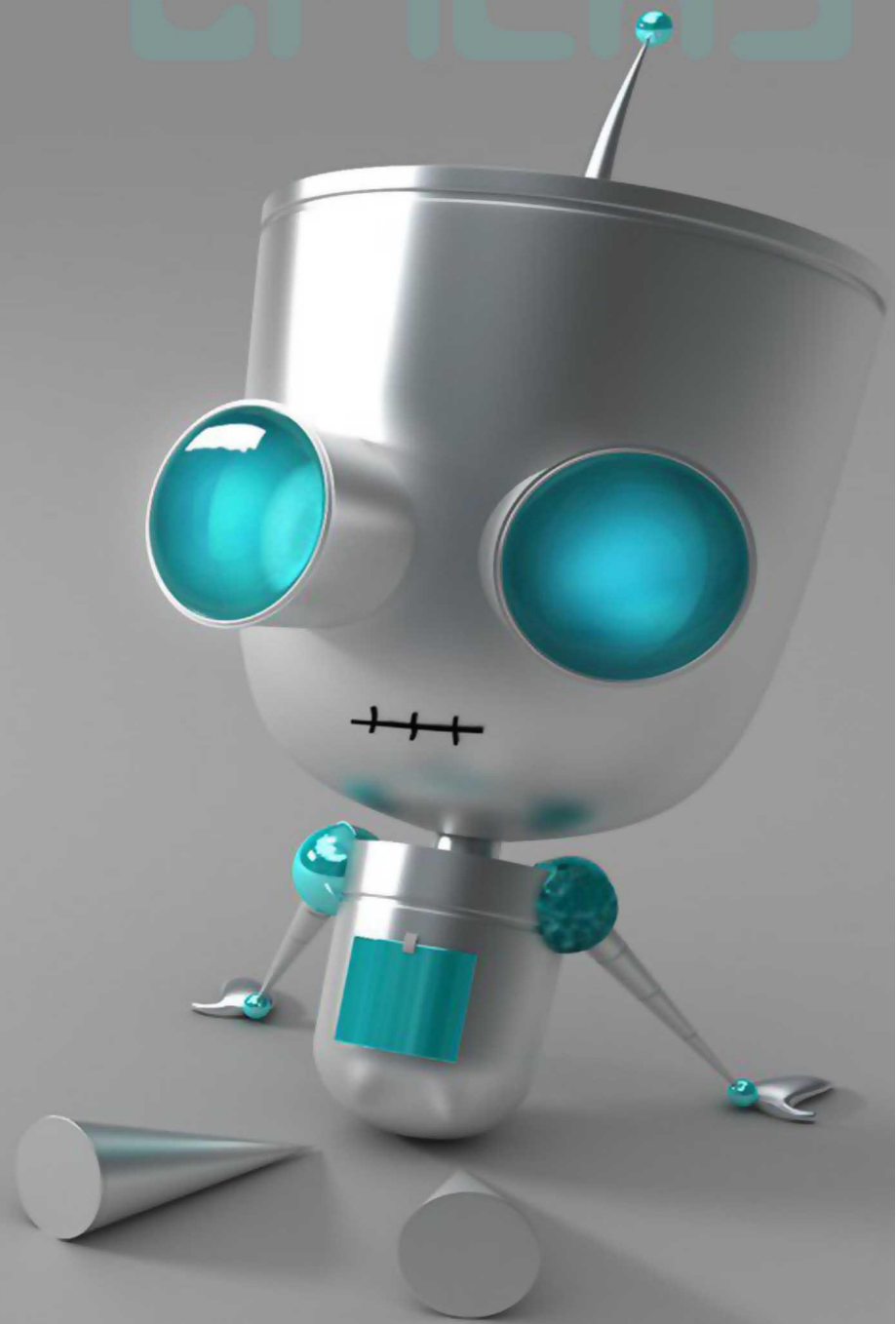
# the bare minimum tutorial

a brief of the workshop

robo-tricks



Rishav Saigal
Shashank Shekhar
Faizaan Ahmed Khan

Department of CSE,,
RCCIIT

# PREFACE

We would like to thank our HOD **Mr. Harinandan Tunga** and our beloved teachers **Mr. Rajib Saha** and **Mrs. Monica Singh** for providing us the opportunity to conduct a robotics workshop, to have faith on us and to encourage us to take the challenge. The success of the workshop is the result of their hard work and perseverance.

We would also like to thank our mentors **Mr. Sujit Kumar Ghosh and Mr. Pramit Ghosh**, who helped us and inspired us to work with development boards in the last years.

We would like to appreciate our classmates and juniors who supported us, and tried their best to learn during the sessions.

We are highly obliged by the support of Dipankar sir, and Bishal Sir who provided us the confidence to teach such a huge no. of students at the right moment when we were losing the track.

Since we have made this tutorial in very less time, we would like the readers to point out the mistakes for further rectification and suggest ideas to make the tutorial more interesting to learn.

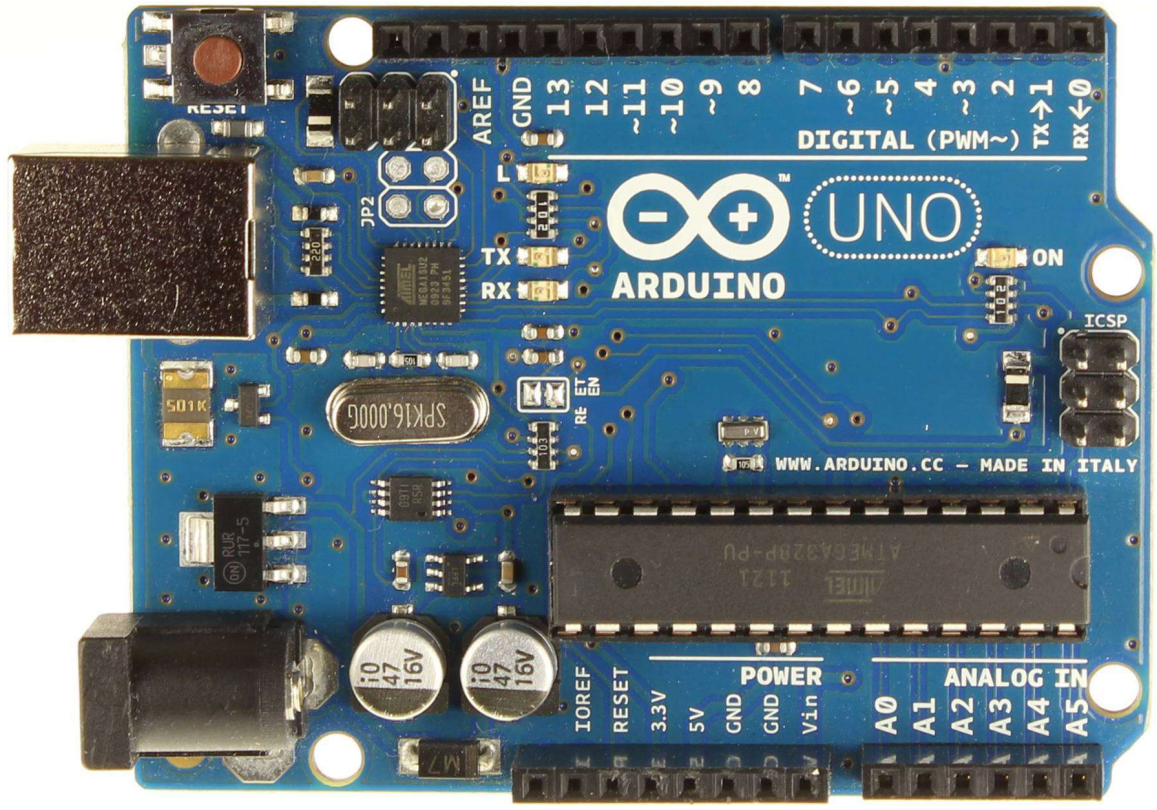
This tutorial contains the brief of the workshop on Autonomous Robotics "Robo-Tricks" and we suggest reading this only to the attendees of the workshop.

Rishav Saigal

Shashank Shekhar

Faizaan Ahmed Khan

# ARDUINO




What is Arduino?

Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based onWiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

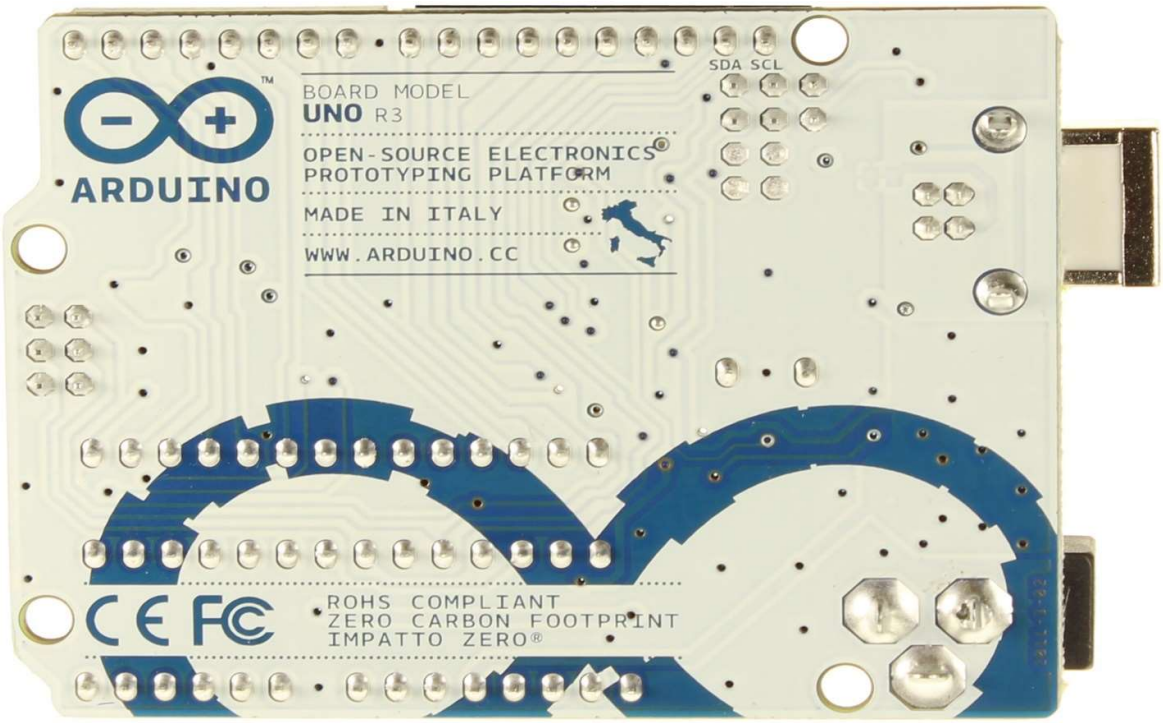
Why Arduino?

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start

tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive
- Cross-platform
- Simple, clear programming environment
- Open source and extensible software
- Open source and extensible hardware



# MICROPROCESSOR

- A **microprocessor** is an electronic component that is used by a computer to do its work.

- It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

- Microprocessors help to do everything from writing to searching the Web.

- Everything a computer does is described by lots of precise instructions, and microprocessors carry out these instructions at incredible speed.

# MICROCONTROLLER

- A microcontroller is a computer present in a single integrated circuit which is dedicated to perform one task and execute one specific application.

- It contains memory, programmable input/output peripherals as well a processor.

- Microcontrollers are mostly designed for embedded applications and are heavily used in automatically controlled electronic devices such as cellphones, cameras, microwave ovens, washing machines, etc.

# ARDUINO IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus.


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

The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

*Verify*
Checks your code for errors compiling it.

*Upload*
Compiles your code and uploads it to the configured board.
See uploading below for details.

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

*New*
Creates a new sketch.

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

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

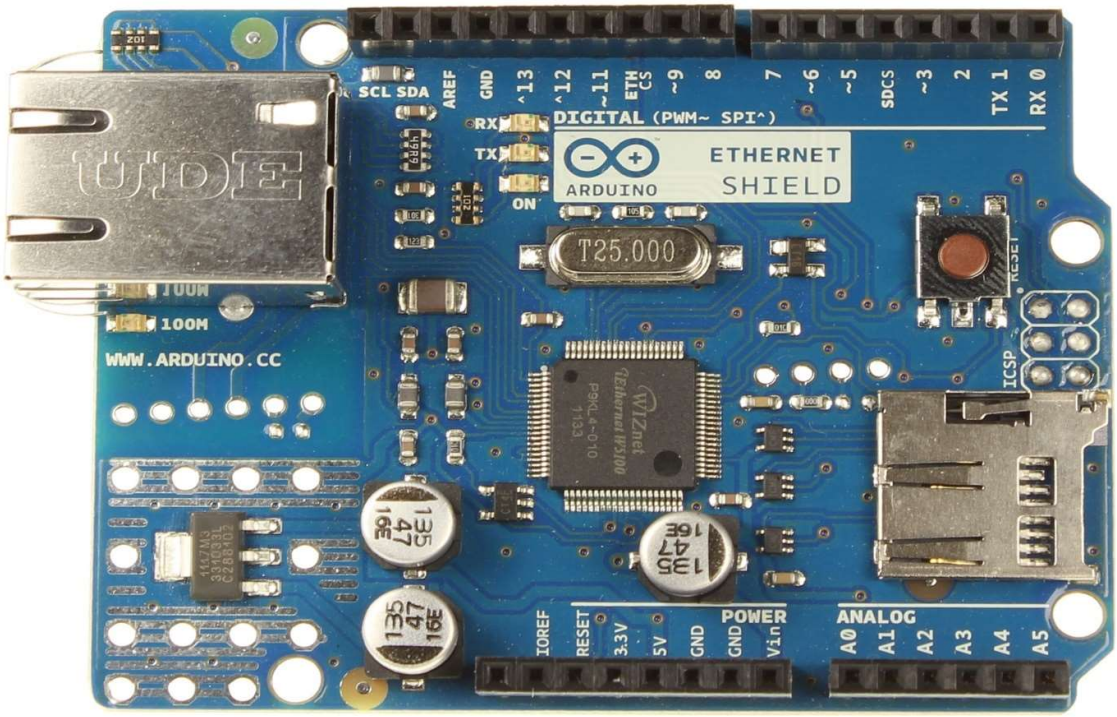
*Save*
Saves your sketch.

*Serial Monitor*
Opens the serial monitor.

For more info : https://www.arduino.cc/en/Guide/Environment

# ARDUINO SHIELDS

Shields are boards that can be plugged on top of the Arduino PCB extending its capabilities. The different shields follow the same philosophy as the original toolkit: they are easy to mount, and cheap to produce.

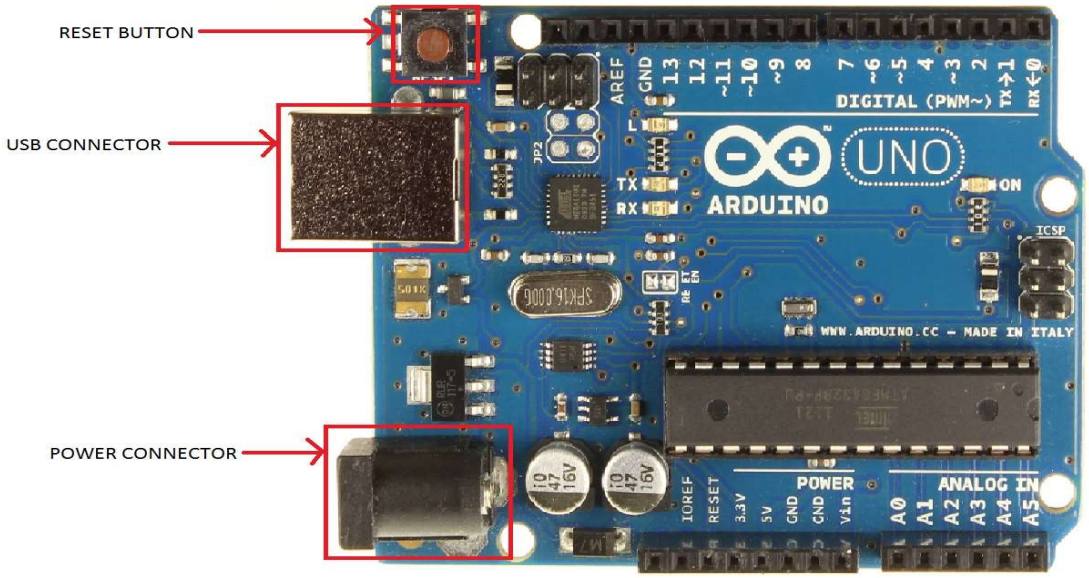



# POWER/RESET

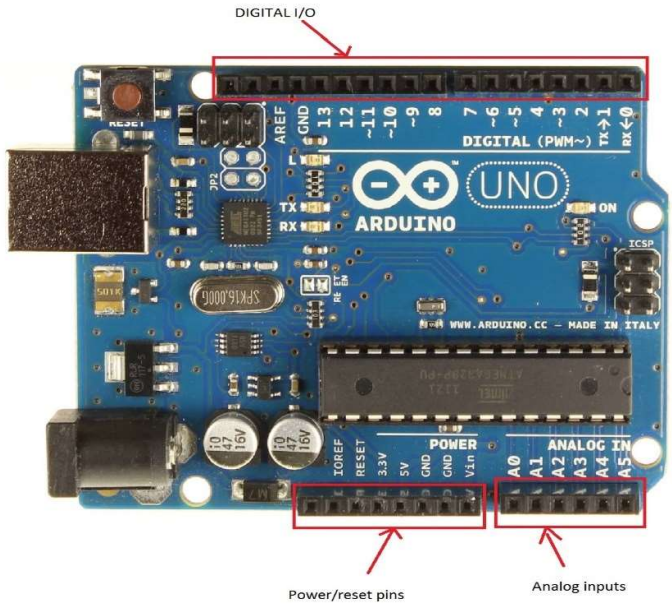
Power button starts the Arduino.

When you push the reset button, it resets the AVR. Also, the Arduino IDE sends a special signal that causes the Arduino board to reset the AVR.

After the AVR is powered up or reset, it immediately begins running the bootloader. The Arduino bootloader this watches for a few seconds to see if a new sketch is being downloaded from the Arduino IDE -- if so, it erases whatever was in flash and burns in the new sketch. Then the bootloader starts running whatever sketch is in the flash.



# INPUT/OUTPUT PINS




Digital pins are used for digital input / output and analog pins are used for analog input. Remember that analog pins can also be used for digital i/o.

Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode() when you're using them as inputs. Pins configured this way are said to be in a **high-impedance state**. Input pins make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 megohm in front of the pin. This means that it takes very little current to move the input pin from one state to another, and can make the pins useful for such tasks as implementing a capacitive touch sensor, reading an LED as a photodiode, or reading an analog sensor with a scheme such as RCTime.

This also means however, that pins configured as pinMode(pin, INPUT) with nothing connected to them, or with wires connected to them that are not

connected to other circuits, will report seemingly random changes in pin state, picking up electrical noise from the environment, or capacitively coupling the state of a nearby pin.

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), or run many sensors, for example, but not enough current to run most relays, solenoids, or motors.

For further details :

https://www.arduino.cc/en/Tutorial/DigitalPins

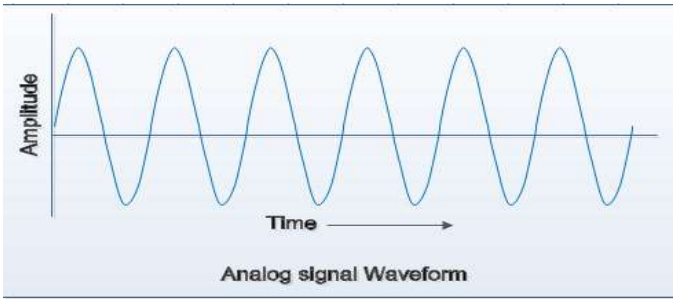
https://www.arduino.cc/en/Tutorial/AnalogInputPins

# ANALOG SIGNAL

An **analog signal** can be represented as a series of sine waves.

The term originated because the modulation of the carrier wave is analogous to the fluctuations of the human voice or other sound that is being transmitted.

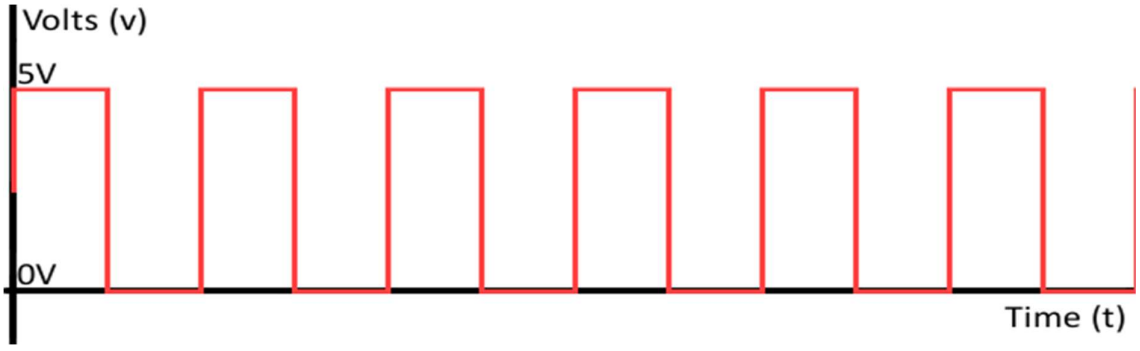
**Analog** describes any fluctuating, evolving, or continually changing process.




# DIGITAL SIGNAL

A Digital signal refers to an electrical signal that is converted into a pattern of bits.

Unlike an Analog signal, which is a continuous signal that contains time-varying quantities, a digital signal has a discrete value at each sampling point



# MICROCONTROLLER IN UNO:


ATmega16U2 handles USB communication

ATmega328 programmed by USER FIRMWARE

Two types of code run on a simple microcontroller.

## 1. Application Code

- ✓ Executes the System's main functionality
- ✓ We write this code

## 2. Firmware

- ✓ Low level code: supports the main function
- ✓ USB interface, power modes, reset etc.

# BOOTLOADER

Firmware on a microcontroller

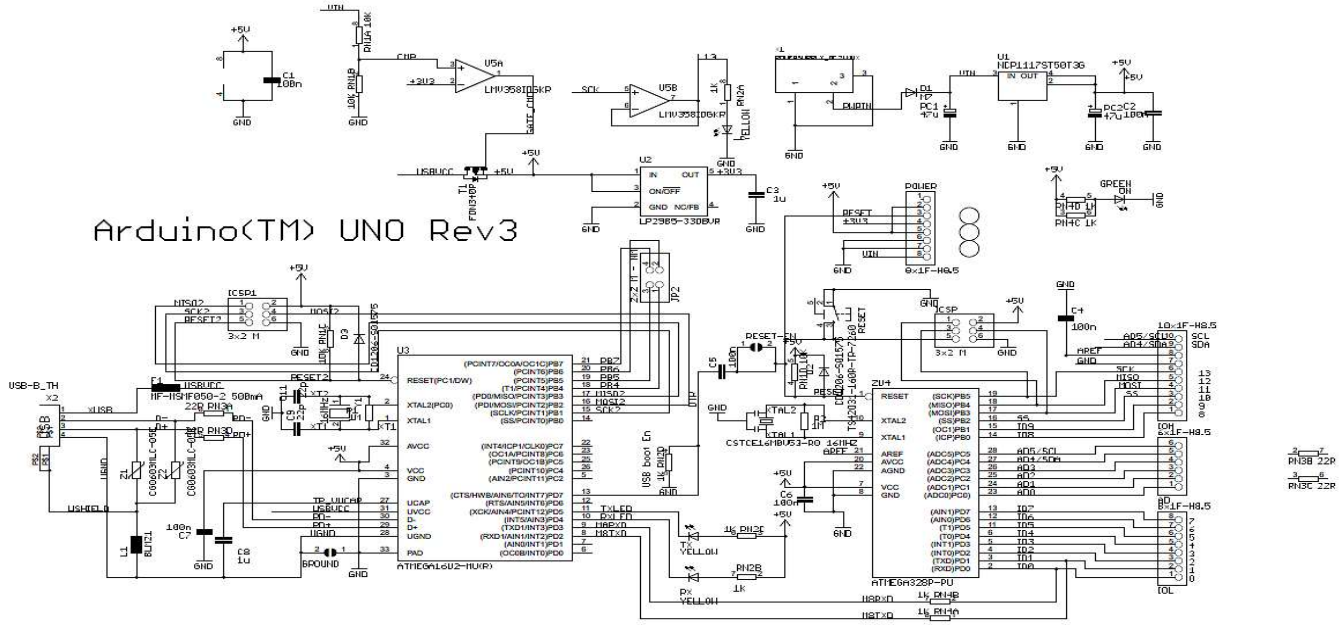
- Allows the Flash and EEPROM  to be programmed
- Manages USB communication, since application programming is via USB

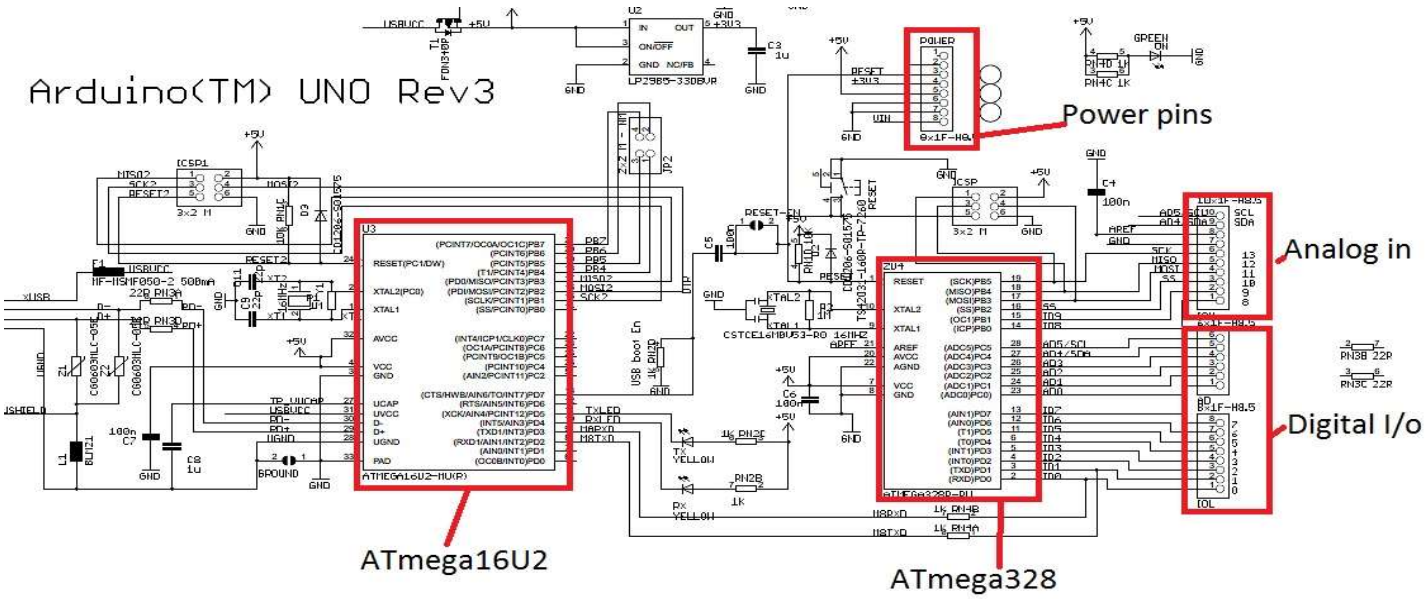
In-Circuit Serial Programming(ICSP)

- A special programming method to program the firmware
- Needed because the bootloader can’t program itself


ICSP Headers(In-Circuit Serial Programming)

# ARDUINO UNO R3 SCHEMATICS



# SERIAL COMMUNICATION

The word **serial** means "one after the other." For example, a serial killer doesn't stop with one murder, but stabs many people one after the other. Serial data transfer is when we transfer data one **bit** at a time, one right after the other.

Information is passed back & forth between the computer and Arduino by, essentially, setting a pin high or low. One side sets the pin and the other reads it.

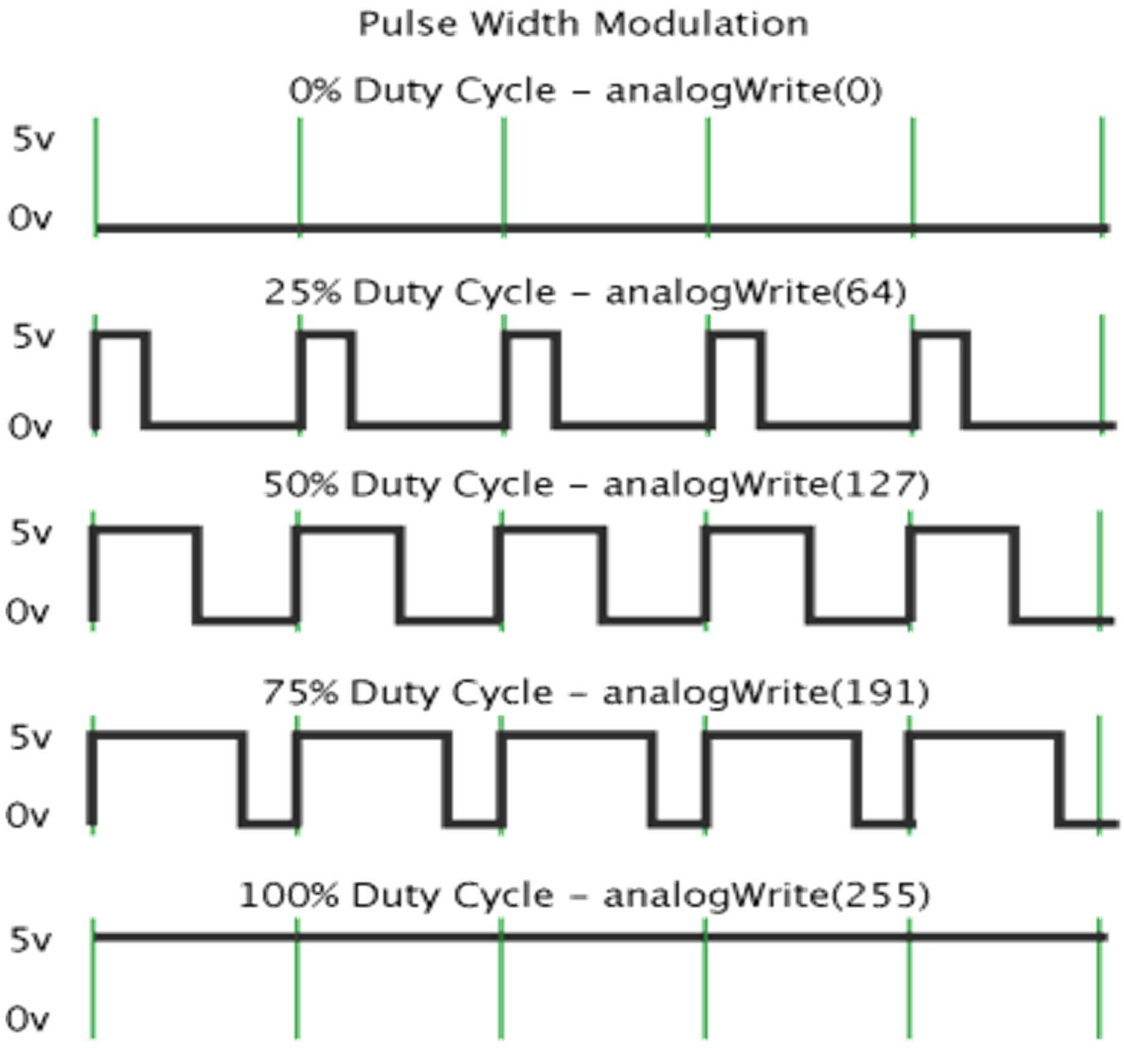
Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

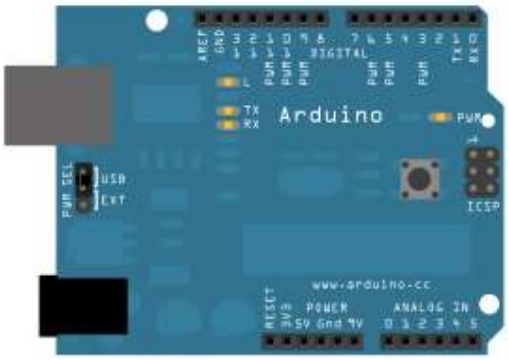

- Serial. begin() : function starts off the communication channel
- Serial.println(): function takes in a string or variable and transmits it, followed by a new line.
- Serial. read():function returns the last received character by the Arduino
- Serial. available():returns true at the moment the Arduino receives data

# ANALOG OUTPUT (PWM)

- Six Pins(3,5,6,9,10,11) on Arduino UNO  are marked as PWM pins.
- These pins are able to send out a PWM signal.
- PWM stands for Pulse width modulation, which is a technique for getting analog results from digital means.



# THE BARE MINIMUM CODE




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

After creating a setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond as it runs. Code in the loop() section of your sketch is used to actively control the board.

The code below won't actually do anything, but it's structure is useful for copying and pasting to get you started on any sketch of your own. It also shows you how to make comments in your code.

Any line that starts with two slashes (//) will not be read by the compiler, so you can write anything you want after it. The two slashes may be put after functional code to keep comments on the same line. Commenting your code like this can be particularly helpful in explaining, both to yourself and others, how your program functions step by step.

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

# BLINK THE PIN 13 LED

```
void setup() {
  //this function is use to set the pins for output/input purpose
  //and this portion runs only once
  pinMode(13,OUTPUT);
//this function sets the 13 number pin as output as it has an internal led
}

void loop() {
  //this function runs always in a loop after the setup function
  digitalWrite(13,HIGH);
//this is use to set the 13 pin as high

  delay(1000);
//this function is used to hold the program for 1 sec as it is written in
millisecond and it acts like an interrupt

  digitalWrite(13,LOW);
//this is use to set the 13 pin as low

  delay(1000);
//this function is used to hold the program for 1 sec as it is written in
millisecond and it acts like an interrupt
}
```

# LET'S START USING THE I/O PINS

To build the circuit connect the long leg of the LED (the positive leg, called the anode) to the pins and the short leg of the LED (the negative leg, called the cathode) to the Arduino GND using bread board and jumper wires. You may require to use a resistor (220 ohm to 1K ohm) depending upon the LED you use. Here we are connecting 4 LED'S with pins 9,10,11, and 12.

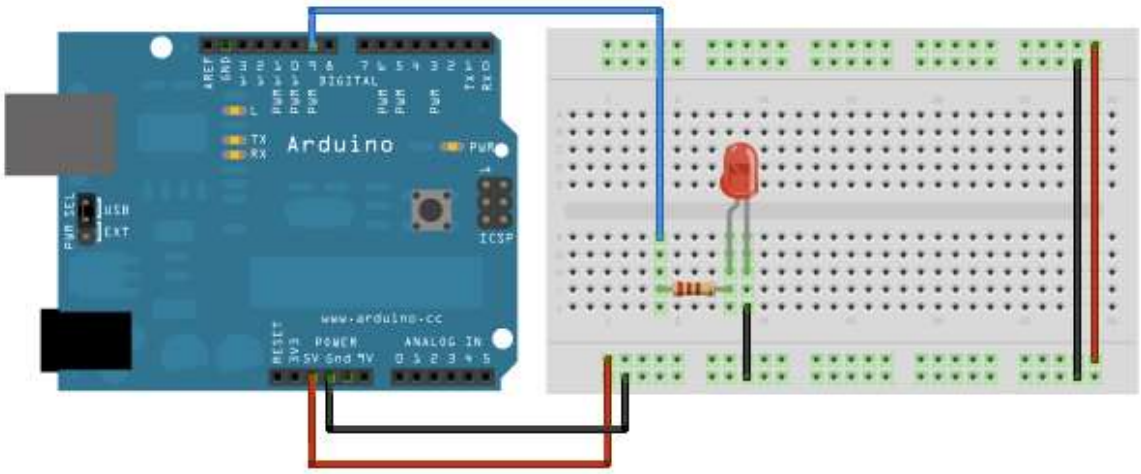
```
void setup() {
  //this function is use to set the pins for output/input purpose
  //and this portion runs only once
  pinMode(12,OUTPUT);
//this function sets the 12 number pin as output and so on up to 9th
  pinMode(11,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(9,OUTPUT);
}
int i=0;
//this is a global variable use as an increment

void loop() {
  //this function runs always in a loop after the setup function
  if(i==0)
  {
//it checks whether the value is 0 or not if it is 0 it sets the first led high and all the other low
    digitalWrite(12,HIGH);
```

```
   digitalWrite(11,LOW);
   digitalWrite(10,LOW);
   digitalWrite(9,LOW);
   ++i;                                                    //increment i
  }
  delay(1000);
//this function is used to hold the program for 1 sec as it is written in
millisecond and it acts like an interrupt


  if(i==1)
  {
//it checks whether the value is 1 or not if it is 1 it sets the second led high and
all the other low
   digitalWrite(12,LOW);
   digitalWrite(11,HIGH);
   digitalWrite(10,LOW);
   digitalWrite(9,LOW);
   ++i;                                                    //increment i
  }
  delay(1000);
//this function is used to hold the program for 1 sec as it is written in
millisecond and it acts like an interrupt


  if(i==2)
  {
//it checks whether the value is 2 or not if it is 2 it sets the third led high and all
the other low
   digitalWrite(12,LOW);
   digitalWrite(11,LOW);
```

```
   digitalWrite(10,HIGH);
   digitalWrite(9,LOW);
   ++i;                                                    //increment i
  }


  delay(1000);
//this function is used to hold the program for 1 sec as it is written in millisecond and it acts like an interrupt
  if(i==3)
  {
//it checks whether the value is 3 or not if it is 3 it sets the fourth led high and all the other low
   digitalWrite(12,LOW);
   digitalWrite(11,LOW);
   digitalWrite(10,LOW);
   digitalWrite(9,HIGH);
   ++i;                                                    //increment i
  }
  delay(1000);
//this function is used to hold the program for 1 sec as it is written in millisecond and it acts like an interrupt
  if(i==4)
  {
//it checks whether the value is 4 if it is 4 the value i is again intialised as 0
   i=0;
  }
}
```

# EXAMPLE TO SHOW THE USE OF PWM PINS (LED FADE)

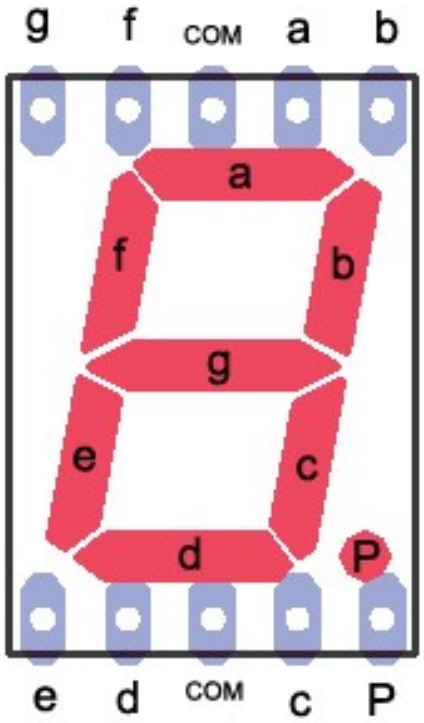
To build the circuit connect the long leg of the LED (the positive leg, called the anode) to the pins and the short leg of the LED (the negative leg, called the cathode) to the Arduino GND using bread board and jumper wires. You may require to use a resistor (220 ohm to 1K ohm) depending upon the LED you use. Use your own logics to find which pin to connect the LED with as per the code.

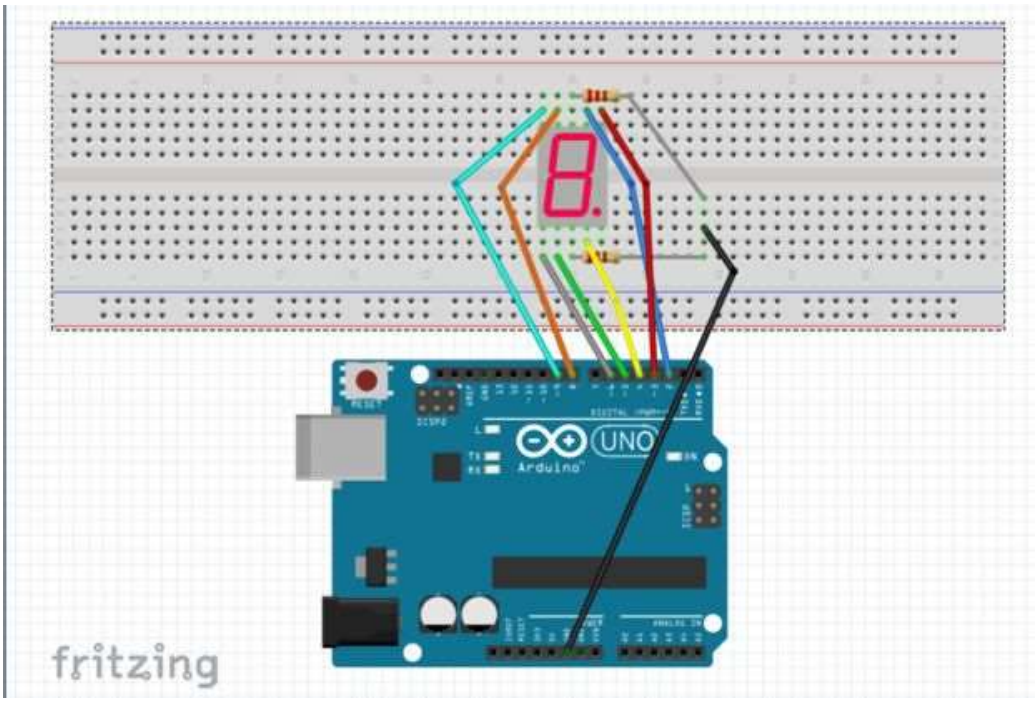



```
void setup() {

  //this function is use to set the pins for output/input purpose
  //and this portion runs only once

  pinMode(9 , OUTPUT);
//this function sets the 9 number pin as output
```

```
}
int i;
//this is a global variable used to fade the led


void loop() {
  //this function runs always in a loop after the setup function


  for(i=0;i<255;i+=50)
  {
//this part of the program is use to increase the value of i by 50
// Note : You can use for loops here just like C , yes you can also use
// while loop , do-while loop, switch case in Arduino Sketches
    analogWrite(9 ,i);
    /*
     * this function analogWrite is use to give the value to the pwm pins these are generally
     * used for the pwm pins i.e pulse width modulation pins , over here we send pulses ranging from
     * 0 to 255 as we can see in the loop with a delay of 1 second.
     */


    delay(1000);
  }
}
```

# TRYING THE 7 SEGMENT DISPLAY



In the below code we print 1 to 9 sequentially

```
int i=0;//this is the global variable use to increase the 7segement led

void setup() {
  //this function is used to setup the pins that are to be used as a input/ouput purpose
  //and this portion runs only once
  for(i=5;i<=12;i++)
  {
    pinMode(i,OUTPUT);
//here we are initialising the pins as input starting from 5 to 12
  }
  i=0;
//again we are initialising i to 0
}

void loop() {
  //this function runs always in a loop after the setup function
  if(i==0)
  {
//this checks whether the value of i is 0 or not if it is true then the 7 segment led pins are been initialised to zero
    digitalWrite(12,HIGH);
    digitalWrite(11,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(8,HIGH);
```

```
   digitalWrite(7,HIGH);
   digitalWrite(6,LOW);
   digitalWrite(5,LOW);
   ++i;//increment of i
  }
  delay(1000);
//this function is used to hold the program for 1 sec as it is written in millisecond and it acts like an interrupt
  if(i==1)
  {
//this checks whether the value of i is 1 or not if it is true then the 7 segment led pins are been initialised to one
   digitalWrite(12,LOW);
   digitalWrite(11,HIGH);
   digitalWrite(10,HIGH);
   digitalWrite(9,LOW);
   digitalWrite(8,LOW);
   digitalWrite(7,LOW);
   digitalWrite(6,LOW);
   digitalWrite(5,LOW);
   ++i;//increment of i
  }
  delay(1000);
//this function is used to hold the program for 1 sec as it is written in millisecond and it acts like an interrupt
  if(i==2)
  {
//this checks whether the value of i is 2 or not if it is true then the 7 segment led pins are been initialised to two
```

```
  digitalWrite(12,HIGH);
  digitalWrite(11,HIGH);
  digitalWrite(10,LOW);
  digitalWrite(9,HIGH);
  digitalWrite(8,HIGH);
  digitalWrite(7,LOW);
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);
  ++i;//increment of i
 }
 delay(1000);
//this function is used to hold the program for 1 sec as it is written in millisecond and it acts like an interrupt

 if(i==3)
 {
//this checks whether the value of i is 3 or not if it is true then the 7 segment led pins are been initialised to three
  digitalWrite(12,HIGH);
  digitalWrite(11,HIGH);
  digitalWrite(10,HIGH);
  digitalWrite(9,HIGH);
  digitalWrite(8,LOW);
  digitalWrite(7,LOW);
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);
  ++i;//increment of i
 }
```

```
  delay(1000);
//this function is used to hold the program for 1 sec as it is written in millisecond and it acts like an interrupt
  if(i==4)
  {
//this checks whether the value of i is 4 or not if it is true then the 7 segment led pins are been initialised to four
    digitalWrite(12,LOW);
    digitalWrite(11,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(9,LOW);
    digitalWrite(8,LOW);
    digitalWrite(7,HIGH);
    digitalWrite(6,HIGH);
    digitalWrite(5,LOW);
    ++i;//increment of i
  }
  delay(1000);

  if(i==5)
  {
//this checks whether the value of i is 5 or not if it is true then the 7 segment led pins are been initialised to five
    digitalWrite(12,HIGH);
    digitalWrite(11,LOW);
    digitalWrite(10,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(8,LOW);
    digitalWrite(7,HIGH);
```

```
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);
  ++i;//increment of i
 }
 delay(1000);

 if(i==6)
 {
//this checks whether the value of i is 6 or not if it is true then the 7 segment led pins are been initialised to six
  digitalWrite(12,HIGH);
  digitalWrite(11,LOW);
  digitalWrite(10,HIGH);
  digitalWrite(9,HIGH);
  digitalWrite(8,HIGH);
  digitalWrite(7,HIGH);
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);
  ++i;//increment of i
 }
 delay(1000);

 if(i==7)
 {

digitalWrite(12,HIGH);
  digitalWrite(11,HIGH);
  digitalWrite(10,HIGH);
```

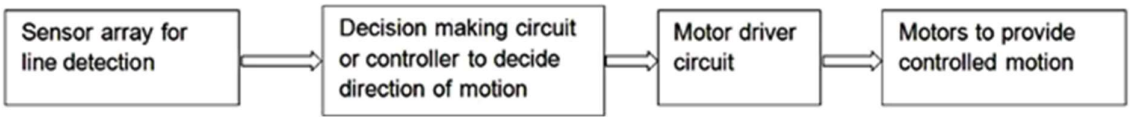
```
  digitalWrite(9,LOW);
  digitalWrite(8,LOW);
  digitalWrite(7,LOW);
  digitalWrite(6,LOW);
  digitalWrite(5,LOW);
  ++i;//increment of i
 }
 delay(1000);

if(i==8)
 {

digitalWrite(12,HIGH);
  digitalWrite(11,HIGH);
  digitalWrite(10,HIGH);
  digitalWrite(9,HIGH);
  digitalWrite(8,HIGH);
  digitalWrite(7,HIGH);
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);
  ++i;//increment of i
 }
 delay(1000);

 if(i==9)
 {

digitalWrite(12,HIGH);
```

```
   digitalWrite(11,HIGH);
   digitalWrite(10,HIGH);
   digitalWrite(9,HIGH);
   digitalWrite(8,LOW);
   digitalWrite(7,HIGH);
   digitalWrite(6,HIGH);
   digitalWrite(5,LOW);
   ++i;//increment of i
  }
  delay(1000);
  if(i==10)
  {
 i=0;
  }
}
```

# LINE FOLLOWER BOT

**Line follower is an autonomous robot** which follows either black line in white are or white line in black area. Robot must be able to detect particular line and keep following it.

For special situations such as cross overs where robot can have more than one path which can be followed, predefined path must be followed by the robot.




An array of sensor is used to detect the line. Based on the status of sensors, special circuit or controller decides the position of line and also the required direction of motion required to follow the line. Motor driver circuit is used to ON/OFF the LEFT/RIGHT motors of the robot to provide desired motion.

Sensors are required to detect position of the line to be followed with respect to the robot's position. Most widely used sensors for the line follower robot are PHOTOSENSERS. They are based on the basic observation that "the white surface reflects the light and the black surface absorbs it".

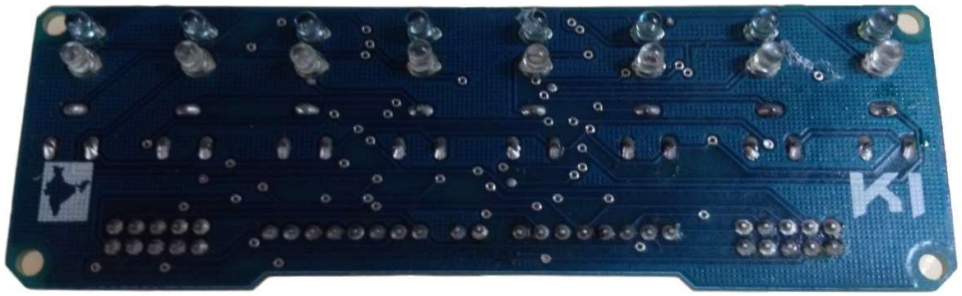
Sensor circuit contains emitter, detector and comparator assembly.

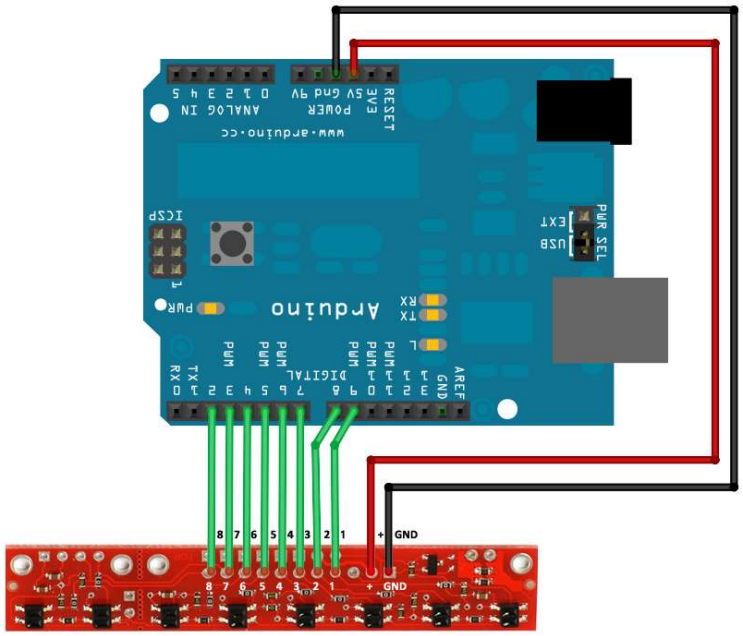
IR or VISIBLE light is emitted from the emitter (IR light is mostly preferred to avoid interference from the visible light which is generally around the robot. However IR light is also present in atmosphere but its intensity is much less than that of visible light, so IR light can give much reliable output. For better accuracy of the sensors, they must be covered properly for the isolation from the surrounding.)

This emitted light strikes the surface and gets reflected back. If the surface is white, more intensity of light gets reflected and for black surface very less intensity of light is reflected.

Photo detector is used to detect the intensity of light reflected. The corresponding analog voltage is induced based on the intensity of reflected light.

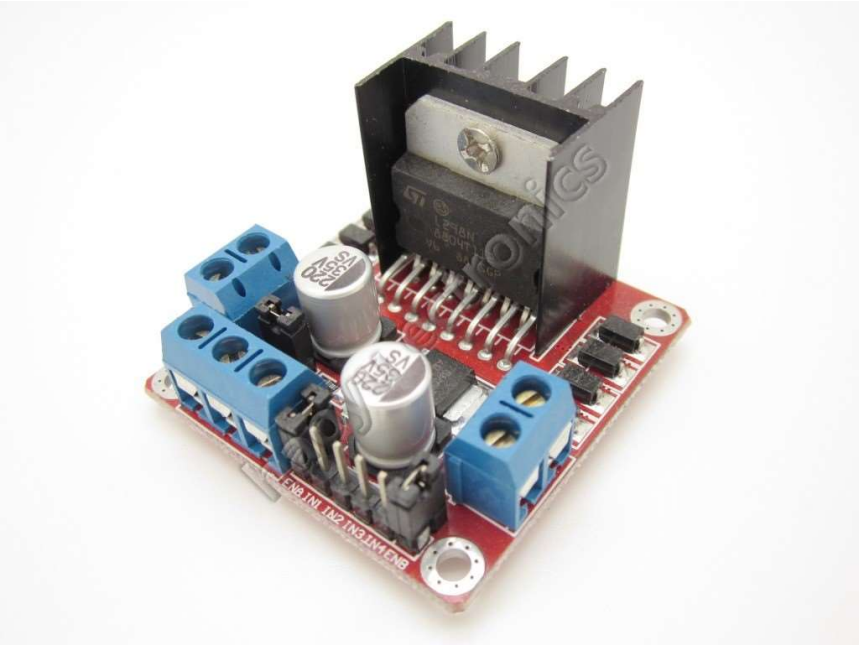
This voltage is compared with the fixed reference voltage in comparator circuit and hence it is converted into logic 0 or logic 1 which can be used by the controller.






Sample connection or IR sensor array

A **motor driver** is a little current amplifier; the function of **motor drivers** is to take a low-current control signal and then turn it into a higher-current signal that can drive a **motor**.



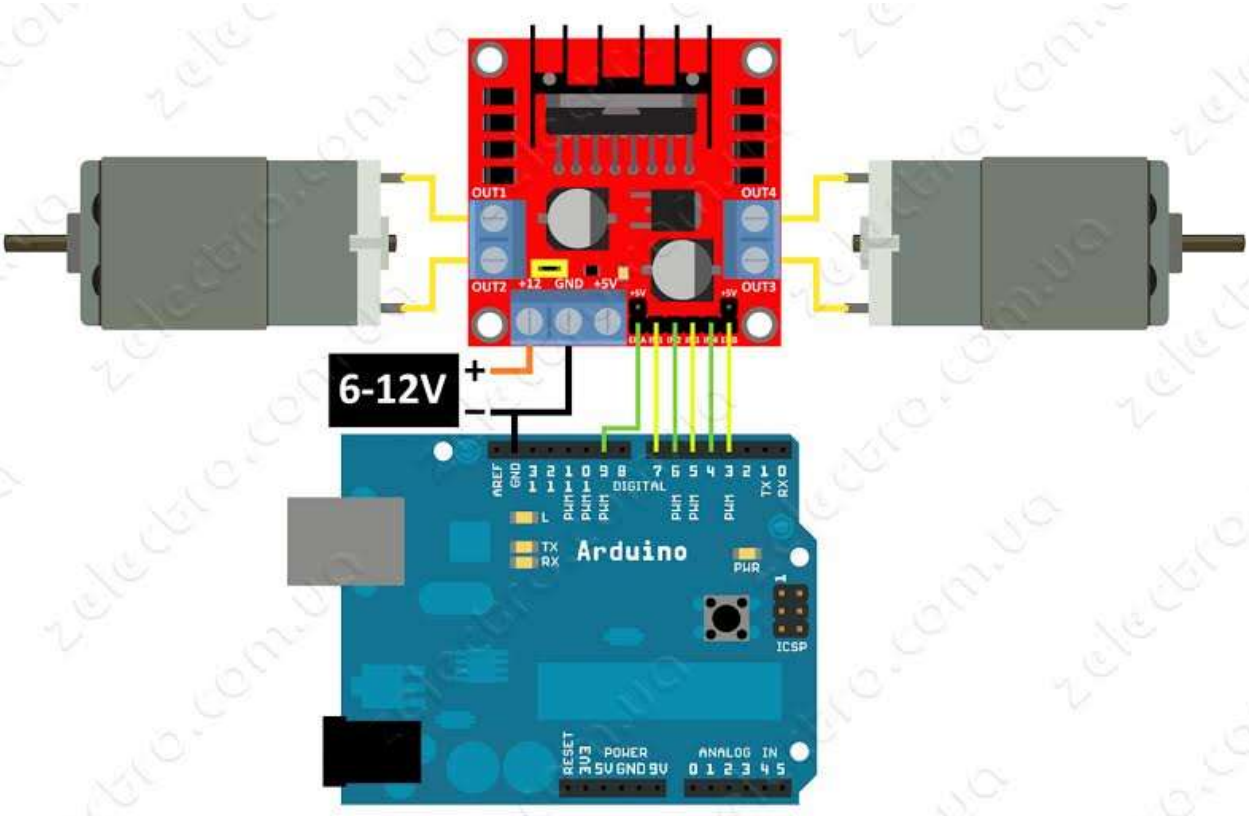
Know more about motor drivers:

http://www.rakeshmondal.info/L293D-Motor-Driver

https://www.robotix.in/tutorial/auto/motor_driver/

http://www.engineersgarage.com/electronic-components/l293d-motor-driver-ic

Connection of a L293D motor driver with motors and Arduino board is shown below:



```
/*
 * this program is for the BLACK line follower we demonstrated in the workshop
 */

int i,a[8];
/*
 * these are the global variables i and array a size having 8.
 * i is used for loop
 * array stores the data recieved from the pins
 */

void setup()
{

  /*
   * this function is used to setup the pins that are to be used for the input/ouput purpose
   * and this portion runs only once
   * in this pins are been intialised some are been intialised as output/input
   */

  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);

  /*
```

```
  * 2 to 4 pins are been used for motor driver and used as a digital pins
  */

 pinMode(9,OUTPUT);
 pinMode(10,OUTPUT);

 /*
  * 9 and 10 pins are the pwm pins used for the purpose of controlling the
speed of the motor
  */

 pinMode(11,INPUT);
 pinMode(12,INPUT);
 pinMode(A0,INPUT);
 pinMode(A1,INPUT);
 pinMode(A2,INPUT);
 pinMode(A3,INPUT);
 pinMode(A4,INPUT);
 pinMode(A5,INPUT);

 /*
  *   from A0 to A5 and 11 and 12 pins are used to input the array sensor data
  *   here A0 to A5 pins are the analog pins they are used as a digital input pins
  *   and 11 and 12 are the digital pins used for input
  */

Serial.begin(57600);
 /*
```

57600 is the Baud rate at which we will view the output in our serial monitor

The **baud rate** is the **rate** at which information is transferred in a communication channel. In the serial port context, "9600 **baud**" means that the serial port is capable of transferring a maximum of 9600 bits per second.

```
  This function start the serial montior to print the received data so that we can
* get a better view of results
* we can even use serial plotter to view our results
   */
}


void loop()
{
  /*
   * this function runs always in a loop after the setup function
   */
  a[0]=digitalRead(A0);
  a[1]=digitalRead(A1);
  a[2]=digitalRead(A2);
  a[3]=digitalRead(A3);
  a[4]=digitalRead(A4);
  a[5]=digitalRead(A5);
  a[6]=digitalRead(11);
  a[7]=digitalRead(12);
  /*
   * this takes the input from the respective pins initialised before from the array sensor and it has been stored in array created before hand
   */
  for(i=0;i<8;i++)
  {
```

```
   Serial.print(a[i]);


  /*
    * this prints the data into the serial monitor
    */


  }
  Serial.println();


  /*
   * this if statement is done to check whether the line follower is in straight condition or not
   * if yes it will contiue to follow this otherwise it will move to next if statement
   */


  if(a[0]&&a[1]&&(!a[2]||!a[3]||!a[4]||!a[5])&&a[6]&&a[7])
  {
/*
In our method of following the line, we are checking if the two extreme left and 2 extreme right IR sensors reads 1, i.e. are placed in the white line and any of the 4 central sensors reads 0 (which means it's on the black line) we move Forward
*/


    analogWrite(9,100);
    analogWrite(10,100);
    digitalWrite(2,LOW);
    digitalWrite(3,HIGH);
```

```
  digitalWrite(4,HIGH);
  digitalWrite(5,LOW);



  /*
   * in this analogWrite funtion is use to set motors speed
   * and the digitalWrite function is use to set the poles of the motor HIGH for positive and LOW for negative
   */
 }


 /*
  * this if statement is done to check whether line is towards left or not if so then the linefollower moves
  * towards left if yes it will contiue to follow this otherwise it will move to next if statement
  */
 if((!a[0]||!a[1])&&a[2]&&a[3]&&a[4]&&a[5]&&a[6]&&a[7])
 {
  analogWrite(9,80);
  analogWrite(10,80);
  digitalWrite(2,LOW);
  digitalWrite(3,HIGH);
  digitalWrite(4,LOW);
  digitalWrite(5,HIGH);

  /*
   * in this analogWrite funtion is use to set motors speed
```

```
    * and the digitalWrite function is use to set the poles of the motor HIGH for positive and LOW for negative
    */
  }

/*
   * this if statement is done to check whether line is towards right or not if so then the linefollower moves
   * towards right if yes it will contiue to follow this otherwise it will move to next if statement
   */

  if(a[0]&&a[1]&&a[2]&&a[3]&&a[4]&&a[5]&&(!a[6]||!a[7]))
  {
    analogWrite(9,80);
    analogWrite(10,80);
    digitalWrite(2,HIGH);
    digitalWrite(3,LOW);
    digitalWrite(4,HIGH);
    digitalWrite(5,LOW);
    /*
     * in this analogWrite funtion is use to set motors speed
     * and the digitalWrite function is use to set the poles of the motor HIGH for positive and LOW for negative
     */
  }
  /*
   * this if statement is done to check whether we have reached a checkpoint or not
```

```
   * if yes then it follows the straight path else it follows other condition if statisfied
   */
  if(!a[0]&&!a[1]&&!a[2]&&!a[3]&&!a[4]&&!a[5]&&!a[6]&&!a[7])
  {
    analogWrite(9,100);
    analogWrite(10,100);
    digitalWrite(2,LOW);
    digitalWrite(3,HIGH);
    digitalWrite(4,HIGH);
    digitalWrite(5,LOW);
    /*
     * in this analogWrite funtion is use to set motors speed
     * and the digitalWrite function is use to set the poles of the motor HIGH for positive and LOW for negative
     */


  }
  if(a[0]&&a[1]&&a[2]&&a[3]&&a[4]&&a[5]&&a[6]&&a[7])
  {
    analogWrite(9,0);
    analogWrite(10,0);
    /*
     * this is done to check whether the linefollower is in white portion
     * or not if yes the linefollower stops else it follows its path
     */
  }
}
```

# THE BEST PLACES TO VISIT NEXT:

https://www.arduino.cc/en/Tutorial/HomePage

https://www.robotix.in/tutorial/

http://playground.arduino.cc/Code/PIDLibrary