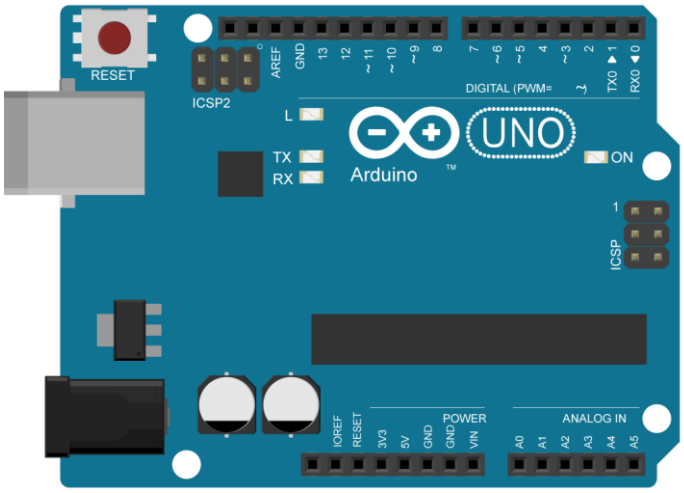
ARDUINO

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards can 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 on Wiring), 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. The software, too, is open-source, and it is growing through the contributions of users worldwide.

1. Introduction

Welcome to Arduino, Before Starting controlling this world you will need to configure the software to program your board.

Arduino IDE (Integrated development environment) is an all in one software to write, compile and upload your software to the Arduino Board.

Arduino gives you two options:

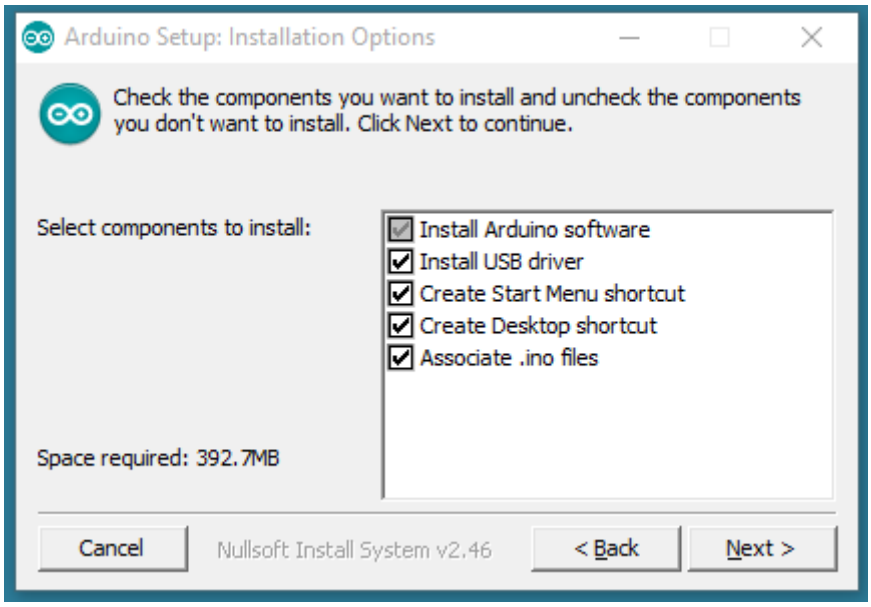
a. If you have a very reliable Internet connection you can use Online IDE. Here you can create your sketches and store them on an online cloud, having them available from any device.
b. If you want to work offline you download the latest version of Arduino IDE.

2. Installing Arduino IDE

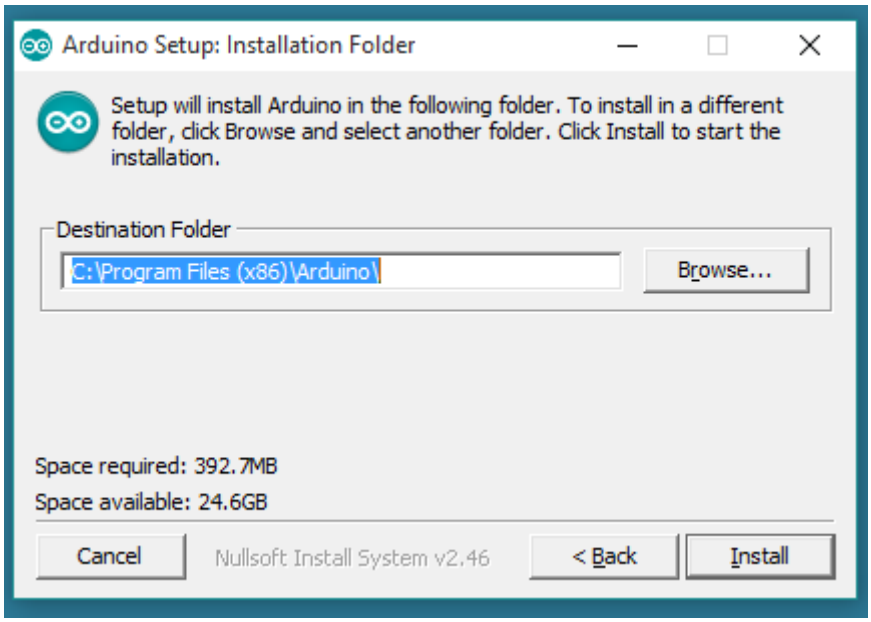
a. Installing on Windows

Download the Latest Version of Arduino IDE. There are two options available, installer (.exe) and zip package(.zip). I would recommend you to use the first option (.exe) because it will automatically install everything including drivers whereas in zip package you will need to install the drivers manually.

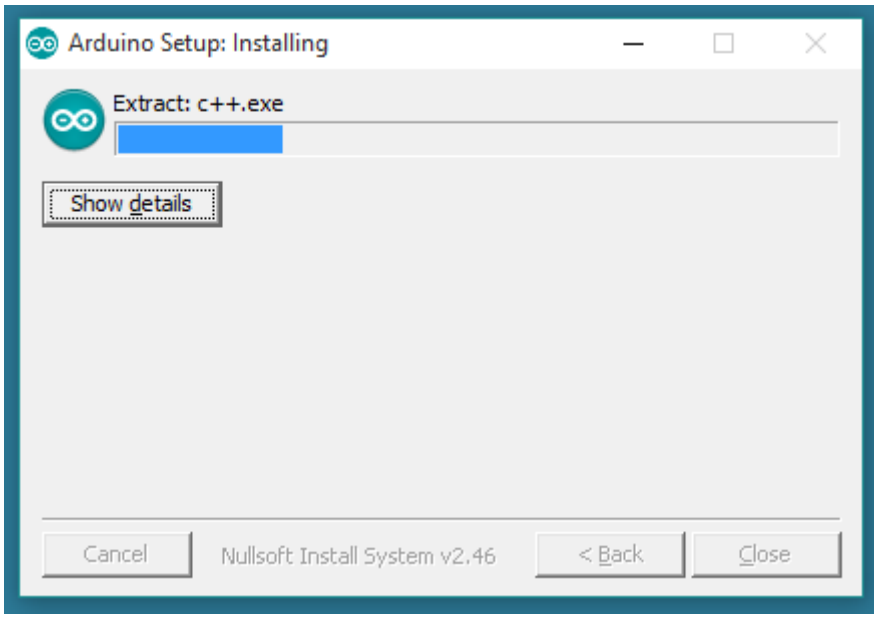
When downloading finishes, proceed with the installation and allow driver installation when you are prompted by the operating system. Choose all the components.



Choose installation directory (suggestion: Keep it default)

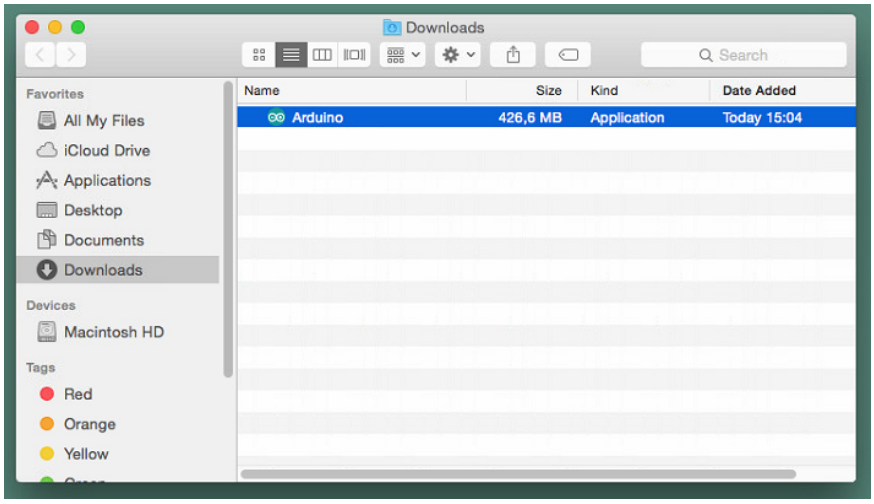


This process will install the software and all the drivers required on your system

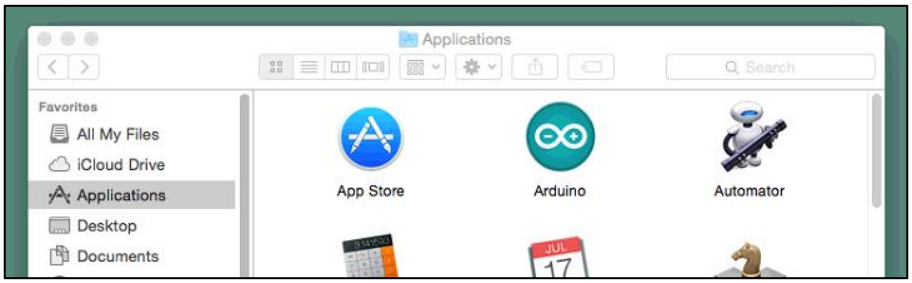


b. Installing on OS X

Download the Latest Version of Arduino IDE. The files will be in zip format, using safari they will be automatically expanded whereas if you use a different browser you have to extract it manually.

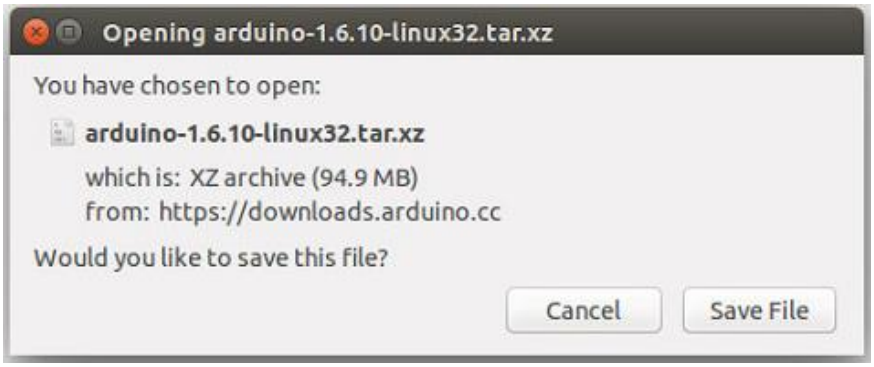


Now Copy Arduino application into any folder on your computer

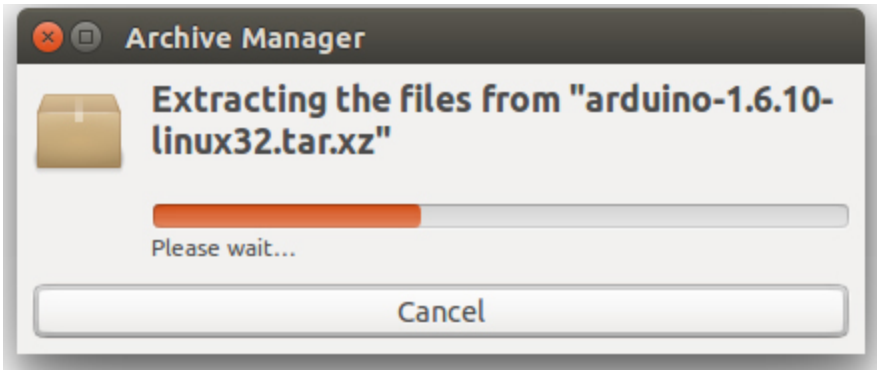


c. Installing on Linux

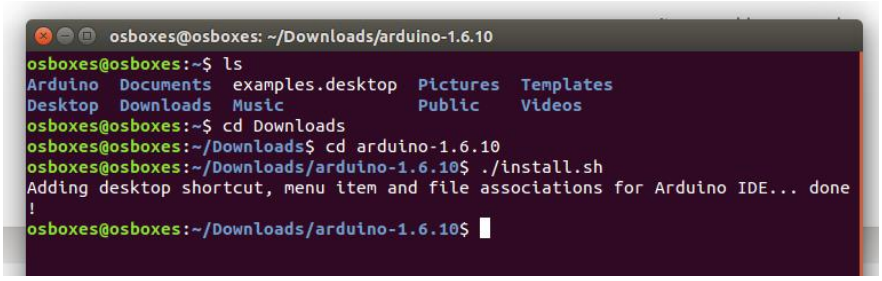
Download the Latest Version of Arduino IDE. It is very important that you choose 32,64 or ARM versions according to the Linux distro you are using. After choosing the right version you can download and the file on your system.



Extract the file to a suitable folder. Remember that it will be executed from there.



Open **Arduino-1.x.x** created by the extraction process. Right Click on Install.sh file and choose **Run in Terminal** from the menu. After the installation process is completed you will find a new icon on your desktop.
If you are unable to run the script from a contextual menu, you can open a new terminal and browse Arduino-1.x.x folder. Type "./install.sh" and wait for it to complete.



Please Note:
If you get an error *"Error opening serial port",* you need to set serial port permission

Open Terminal and type following command
ls -l /dev/ttyACM*

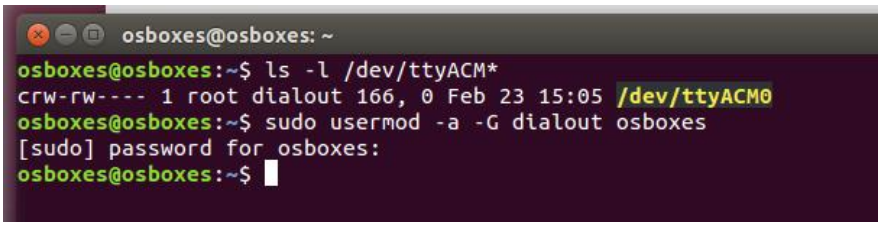
you will get this
crw-rw---- 1 root dialout 188, 0 5 apr 23.01 ttyACM0

The "0" at the end of ACM might be a different number, or multiple entries might be returned. The data we need is "dialout" (is the group owner of the file).

Now we just need to add our user to the group:

sudo usermod -a -G dialout <username>

where <username> is your Linux user name. **You will need to log out and log in again for this change to take effect.**



After this procedure, you should be able to proceed normally and upload the sketch to your board or use the Serial Monitor.

3. Arduino IDE

The Arduino IDE (Integrated Development Environment) contains a text editor for writing your program, a message box and a toolbar used to compile verify and Upload your program.

a. **Writing Sketches**
Programs that you write inside Arduino IDE are known as sketches. The sketches can be saved with a file extension ".ino".

b. **Toolbar options**

- Verify - Check your sketch for any errors
- Upload - Compiles your code and uploads it to the board. Hold down the shift key along with this button if you are using any external programmer.
- New - Used to create new sketch.
- Open - Use it to open Arduino sketches that are saved on your system.
- Save - Save the current Sketch.
- Serial Monitor - Opens the Serial Monitor.

Additional Commands can be found within Five context-sensitive Menus; **File, Edit, Sketch, Tools, Help**.

FILE

- **New -** Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
- **Open** - Allows loading a sketch file browsing through the computer drives and folders.

- **Open Recent -** It provides a shortlist of the most recent sketches, ready to be opened.
- **Sketchbook -** Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
- **Examples** - Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
- **Close** - Closes the instance of the Arduino Software from which it is clicked.
- **Save -** Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
- **Save as... -** Allows saving the current sketch with a different name.
- **Page Setup -** It shows the Page Setup window for printing.
- **Print** - Sends the current sketch to the printer according to the settings defined in Page Setup.
- **Preferences** - Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
- **Quit** - Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

- **Undo/Redo -** Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
- **Cut** - Removes the selected text from the editor and places it into the clipboard.
- **Copy** - Duplicates the selected text in the editor and places it into the clipboard.
- **Copy for Forum -** Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
- **Copy as HTML** - Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
- **Paste** - Puts the contents of the clipboard at the cursor position, in the editor.
- **Select All -** Selects and highlights the whole content of the editor.
- **Comment/Uncomment** - Puts or removes the // comment marker at the beginning of each selected line.
- **Increase/Decrease Indent** - Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
- **Find -** Opens the Find and Replace window where you can specify the text to search inside the current sketch according to several options.
- **Find Next** - Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
- **Find Previous -** Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

SKETCH

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

- **Export Compiled Binary -** Saves a .hex file that may be kept as an archive or sent to the board using other tools.
- **Show Sketch Folder -** Opens the current sketch folder.
- **Include Library -** Adds a library to your sketch by inserting #include statements at the start of your code. Additionally, from this menu item, you can access the Library Manager and import new libraries from .zip files.
- **Add File... -** Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right side o the toolbar.

TOOLS

- **Auto Format -** This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
- **Archive Sketch** - Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
- **Fix Encoding & Reload** - Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
- **Serial Monitor** - Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over the serial port opening.
- **Board -** Select the board that you're using. See below for descriptions of the various boards.
- **Port -** This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
- **Programmer** - For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're burning a bootloader to a new microcontroller, you will use this.
- **Burn Bootloader** - The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for the normal use of an Arduino or Genuino board but is useful if you purchase a new ATmega microcontroller (which normally comes without a bootloader). Ensure that you've selected the correct board from the Boards menu before burning the bootloader on the target board. This command also sets the right fuses.
- **Help** - Here you find easy access to many documents that come with the Arduino Software (IDE). The documents are a local copy of the online ones and may link back to our online website.

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

4. Uploading Sketch (Arduino Uno Board)

a. **Step 1 :**

Connect the Arduino board with your System using USB Cable and open the Sketch inside Arduino IDE which you want to upload.

b. **Step 2 :**

In Menu goto **Tools > Board >Arduino/Genuino Uno**

c. **Step 3 :**

Now Select com port from **Tools > Port > *Select your Port***

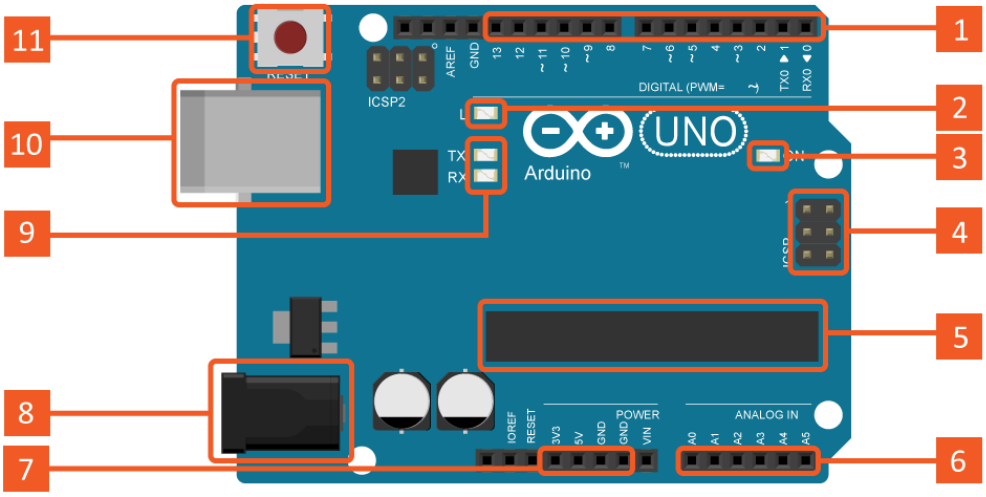
d. **Step 4 :**

Click Upload  Button.

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

5. Anatomy of Arduino

Here is the anatomy of Arduino/Genuino Uno.



1 **Digital Pins:** Use these pins with digitalWrite(), digitalWrite() and analogWrite(). These are digital input and output pins moreover some of them with PWM symbol (~) can be used to send the analog output as well using analogWrite().

2 **LED:** A built-in LED on Pin 13 of Arduino Uno Board. This pin can be used to test your blinking sketch and it is also helpful for debugging.

3 **Power LED:** Indicates that the board is receiving power. Useful for debugging.

4 **ICSP:** Stands for **"**In-Circuit Serial Programming". It is the cheapest and most practical solution to burn a bootloader on another Arduino board with ATmega, 32U4 or ATtiny.

5 **Microcontroller:** This is the Heart of this Board which controls all the things.

6 **Analog In:** Use these pins to receive analog Inputs using analogRead().

7 **GND and 5V pins:** These pins are used to power other components which you want to use with Arduino.

8 **Power Connector:** This connector is used to power Arduino when it is not connected to USB. It can accept voltages between 7 to 12 Volts.

9 **Tx and Rx LEDs:** These LEDs indicate any communication using TX and RX Pins on Arduino Board. They can also be seen flickering when you upload your program. Useful for debugging.

10 **USB Port:** Used for powering Arduino, uploading your sketches, and for communicating with Arduino sketch (via Serial.println() etc).

11 **Reset button:** Resets the ATmega microcontroller.

6. Language Reference

Arduino programming language can be divided into three main parts: functions, values (variables and constants), and structure.

a. **Functions:** For controlling the Arduino board and performing computations.

Digital I/O
digitalRead()
digitalWrite()
pinMode()

Analog I/O
analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family
analogReadResolution()
analogWriteResolution()

Advanced I/O
noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time
delay()
delayMicroseconds()
micros()
millis()

Math
abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

Trigonometry
cos()
sin()
tan()

Characters
isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

Random Numbers
random()
randomSeed()

Bits and Bytes
bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

External Interrupts
attachInterrupt()
detachInterrupt()

Interrupts
interrupts()
noInterrupts()

Communication
Serial
Stream

USB
Keyboard
Mouse

b. **Variables:** Arduino data types and constants

Constants
Floating Point Constants
Integer Constants
HIGH | LOW
INPUT | OUTPUT
INPUT_PULLUP
LED_BUILTIN
true | false

Data Types
String()
array
bool
boolean
byte
char
double
float
int
long

Variable Scope &Qualifiers
const
scope
static
volatile

Conversion
(unsigned int)
(unsigned long)
byte()
char()
float()
int()
long()
word()

Data Types
short
size_t
string
unsigned char
unsigned int
unsigned long
void
word

Utilities
PROGMEM
sizeof()

c. **Structure:** Elements of the Arduino (C++) code.

Sketch
loop()
setup()

Control Structure
break
continue
do...while
else
for
goto
if
return
switch...case
while

Arithmetic Operators
% (remainder)
* (multiplication)
+ (addition)
- (subtraction)
/ (division)
= (assignment operator)

Comparison Operators
!= (not equal to)
< (less than)
<= (less than or equal to)
== (equal to)
> (greater than)
>= (greater than or equal to)

Pointer Access Operators
& (reference operator)
* (dereference operator)

Bitwise Operators
& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Further Syntax
#define (define)
#include (include)
/* */ (block comment)
// (single line comment)
; (semicolon)
{} (curly braces)

Boolean Operators
! (logical not)
&& (logical and)
|| (logical or)

Compound Operators
%= (compound remainder)
&= (compound bitwise and)
*= (compound multiplication)
++ (increment)
+= (compound addition)
-- (decrement)
-= (compound subtraction)
/= (compound division)
^= (compound bitwise xor)
|= (compound bitwise or)

7. Writing digital Signal on Pins

When getting started with Arduino Board we will start with a very basic project of blinking an LED on Pin 13 of Arduino just to make sure everything is working fine.

Functions:

`void setup()`

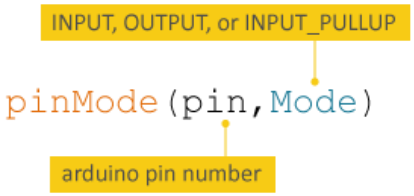
This function is called when a sketch starts. You can use to initialize variables, pin modes, start using libraries, etc. This function will only run once, after each power-up or reset of the Arduino board.

`void loop()`

This function is called just after calling setup(). As the name suggests this function actively controls the Arduino board by looping consecutively.

`pinMode()`

- **Description:** Configure specific pin to behave either as input or output.
- **Note:** It is only applicable for digital pins.
- **Syntax:**



`digitalWrite()`

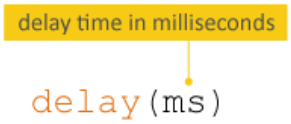
- **Description:** Write a HIGH or LOW value to a digital pin.
- **Note:**
  It is very important to set pinMode() as OUTPUT first before using digitalWrite() function on that pin.

  If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim.

  The analog input pins can be used as digital pins, referred to as A0, A1, etc. The exception is the Arduino Nano, Pro Mini, and Mini's A6 and A7 pins, which can only be used as analog inputs.
- **Syntax:**

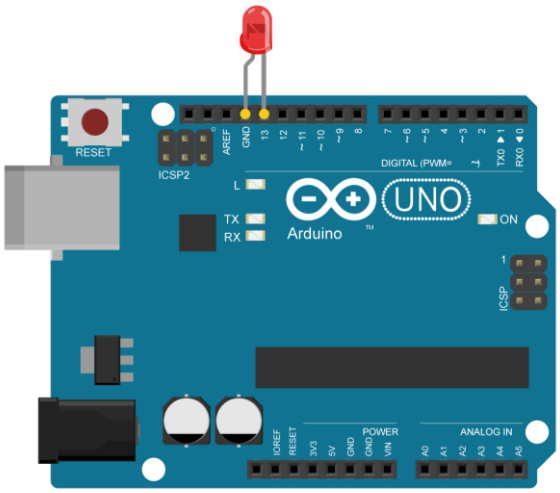
`delay()`

- **Description:** Pauses the program for the amount of time (in milliseconds) as specified inside the bracket. (1sec = 1000 ms).
- **Note:** Certain things do go on while the delay() function is controlling the Atmega chip, however, because the delay function does not disable interrupts. Serial communication that appears at the RX pin is recorded, PWM (analogWrite) values and pin states are maintained.
- **Syntax:**



PROJECT: **To Blink LED connected on PIN 13 of Arduino Uno**

Connections:



Program:

```
/*
 * Basic LED Blink
 * turns on and off a light-emitting diode(LED) connected to digital
 * pin 13.
 */

void setup() {
  pinMode(13, OUTPUT);     // sets the digital pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH); // sets the digital pin 13 on
  delay(1000);            // waits for a second
  digitalWrite(13, LOW);  // sets the digital pin 13 off
  delay(1000);            // waits for a second
}
```

8. Reading digital Signal

Now as we know how to send a digital output signal. In this project, we will try to read the signal on digital PIN 2 using the push button (The pushbutton is a component that connects two points in a circuit when you press it). The example turns on an LED when you press the button.

Functions:

`digitalRead()`

- **Description:** It is used to read a digital value from any of the digital pins.
- **Note:** If the pin isn't connected to anything, digitalRead() can return either HIGH or LOW (and this can change randomly).
- **Syntax:**

`millis()`

- **Description:** It returns the number of milliseconds passed since the Arduino board started running the program. This number will overflow and return to zero in around 50 days.
- **Note:** Please note that the return value for millis() is of type unsigned long, logic errors may occur if a programmer tries to do arithmetic with smaller data types such as int. Even signed long may encounter errors as its maximum value is half that of its unsigned counterpart.
- **Syntax:**

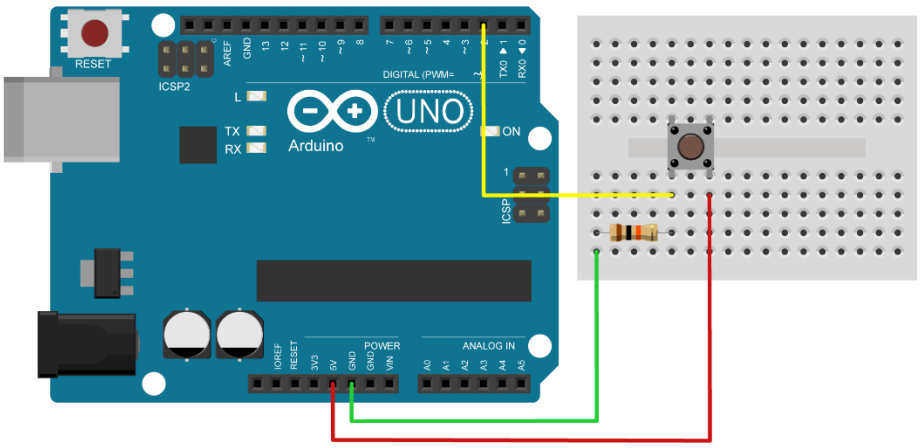
```
time = millis()
```

PROJECT a: **To control LED on pin 13 using Push Button at pin 2**

Hardware:

- Arduino Board
- momentary button or switch
- 10k ohm resistor
- hook-up wires
- breadboard

Connections:



Program:

```
/*
 * Basic Digital Read
 * turns on and off a light-emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 2.
 *
 */

int ledPin = 13; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {

  pinMode(ledPin, OUTPUT);              // declare LED as output

  pinMode(inPin, INPUT);                // declare pushbutton as input

}

void loop(){
  val = digitalRead(inPin);             // read input value

  if (val == HIGH) {                    // check if the input is HIGH
    digitalWrite(ledPin, LOW);          // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH);         // turn LED ON
  }
}
```

PROJECT b: **Debounce**

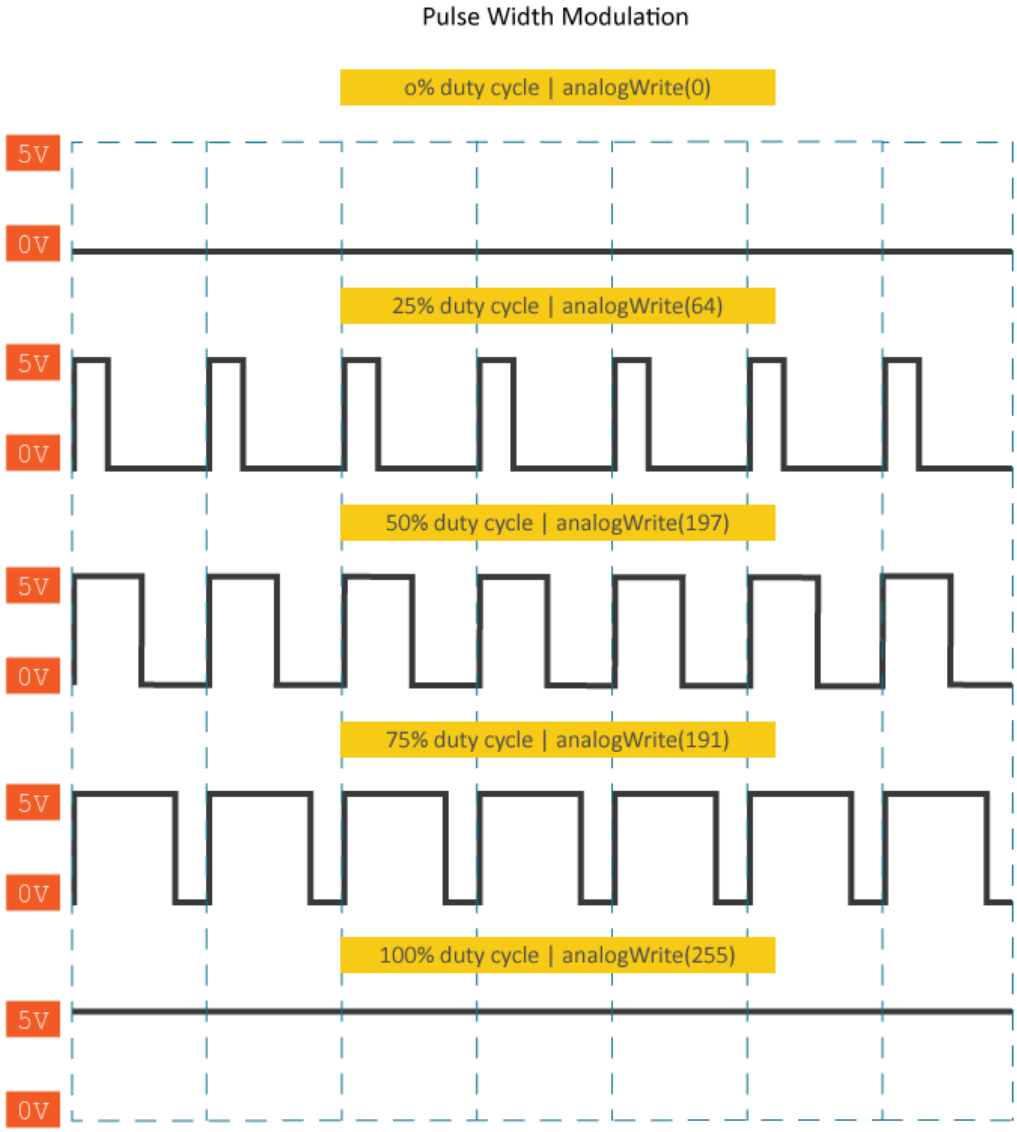
Push buttons often generate noise when pressed because of open-close transitions. These transitions can be read as multiple inputs by the controller and the program might not work as predicted. So to avoid this we debounce the input, by checking twice in a short period of time to make sure that the button is definitely pressed.

Program:

```
const int buttonPin = 2;
const int ledPin = 13;

int ledState = HIGH;
int buttonState;
int lastButtonState = LOW;

unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;     // the debounce time;
//increase debounceDelay if the output flickers

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);
}


void loop() {
  int reading = digitalRead(buttonPin);  // read button pin

  if (reading != lastButtonState) {      // If the switch changed
    lastDebounceTime = millis();         // reset the debouncing timer
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
// delay, so take it as the actual current state

    if (reading != buttonState) {
// if the button state has changed:

      buttonState = reading;
      if (buttonState == HIGH) {
// only toggle the LED if the new button state is HIGH

        ledState = !ledState;
      }
    }
  }

  digitalWrite(ledPin, ledState);
  lastButtonState = reading;
// save the reading. Next time, it'll be the lastButtonState
}
```

9. Sending Analog Output Signal using PWM

In this project, we are going to fade an led connected to PWM pin 11 of the Arduino Board. Although digital devices cannot generate a pure analog signal, therefore, we use this technique called Pulse Width Modulation or PWM. Digital control is used to create a square wave, a signal switched between ON and OFF. This ON and OFF pattern can simulate voltages in between full ON(5V) and Full OFF(0V).



In the graphic above, the vertical dotted lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

Once you get this example running, grab your Arduino and shake it back and forth. What you are doing here is essentially mapping time across space. To our eyes, the movement blurs each LED blink into a line. As the LED fades in and out, those little lines will grow and shrink in length. Now you are seeing the pulse width.

Functions:

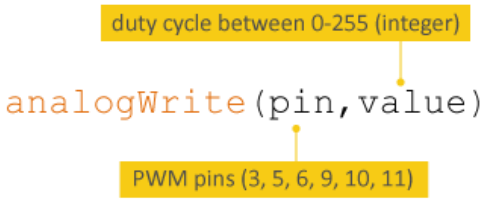
`analogWrite()`

- **Description:** Writes an analog value (PWM) to the pin. You can use it to change the brightness of an LED or vary the speed of the motor.

PWM Pins	Frequency
3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)

- **Note:**
  In Arduino Uno, the output resolution lies between 0 to 255.
  There is no need to set pinMode() before using analogWrite(). The analogWrite() function has nothing to with analog pins or analogRead().

  Pin 5 and 6 can have a higher-than-expected duty cycle because of delay() and millis() as they share the same timer which is used to generate PWM outputs.
- **Syntax:**

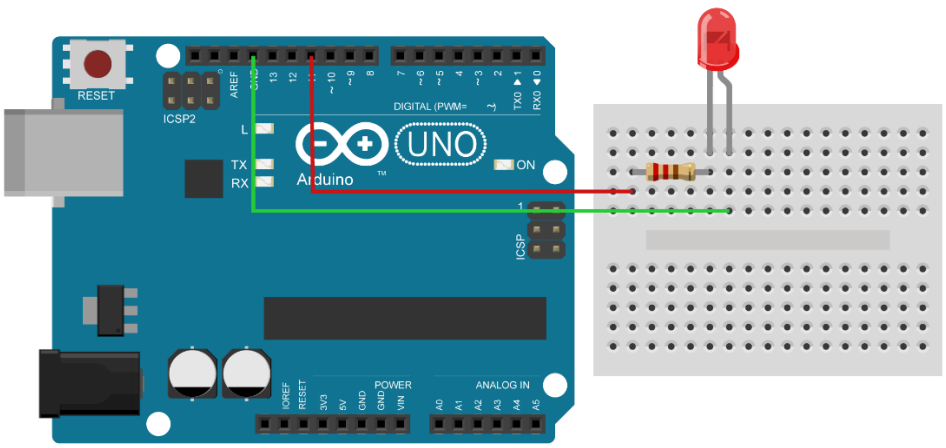


PROJECT a: **To Fade an LED connected on pin 11 using PWM**

Hardware:

- Arduino or Genuino board
- LED
- 220 ohm resistor
- hook-up wires
- breadboard

Connections:



Program:

```
/*
 * Basic Analog Write
 * Fades a Light Emitting diode connected on PIN 11 of Arduino
 * Using technique called PWM Pulse width Modulation
 *
 */


int ledPin = 11;     // LED connected to digital pin 11
void setup() {
}

void loop() {
  //to increase the brightness from minimum to maximum

  for (int i=0;i<=255; i+=5) {
    analogWrite(ledPin,i);
    delay(30);
  }

   //to decrease the brightness from maximum to minimum

  for (int j= 255;j>=0;j-= 5) {
    analogWrite(ledPin,j);
    delay(30);
  }
}
```

Notes: The table below shows PWM pins with Frequency on various Arduino Boards

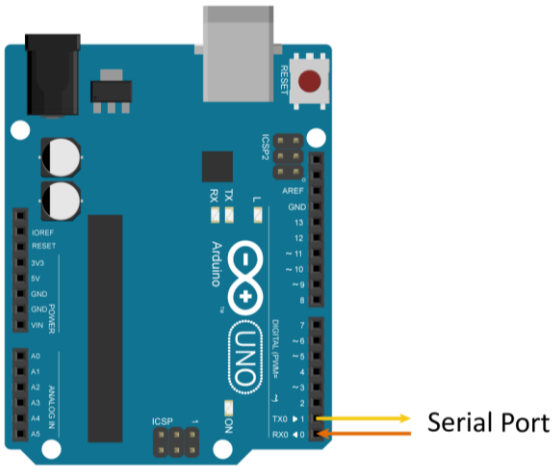
BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev.2	3, 5, 6, 9, 10	976 Hz
MKR boards *	0 - 8, 10, A3 (18), A4 (19)	732 Hz
MKR1000 WiFi *	0 - 8, 10, 11, A3 (18), A4 (19)	732 Hz
Zero *	3 - 13, A0 (14), A1 (15)	732 Hz
Due **	2-13	1000 Hz
101	3, 5, 6, 9	pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz

10. Reading Analog Signals and Using Serial Monitor

In this project, we are going to read analog input from pin A0 of Arduino. Later we can display it on laptop screen using Serial Port or use it to vary the brightness of an LED connected to pin 11 of the Arduino Board.

Serial Communication:

Serial communication (also known as USART or UART) is used to transmit data sequentially, over a communication channel. We can use it to communicate between the Arduino board and a computer or other devices. In Arduino Uno pin 0 and 1 are used for Serial communication.



Here is the list of functions associated with Serial communication. We will discuss them whenever they are needed in any of the Projects.

if(Serial)
available()
availableForWrite()
begin()
end()
find()
findUntil()
flush()
parseFloat()
parseInt()
peek()
print()
println()
read()
readBytes()
readBytesUntil()
readString()
readStringUntil()
setTimeout()
write()
serialEvent()

Now we are going to discuss all the functions required for this project

`analogRead()`

- **Description:** Reads the value from a specific analog Input pin by mapping the input voltages between 0 and the operating voltage (3.3V or 5V) into integer values between 0 to 1023
- **Note:** On Arduino Uno, this yields a resolution of 4.9mV per unit. However, we can change this range with the help of analogReference() function.
- **Syntax:**

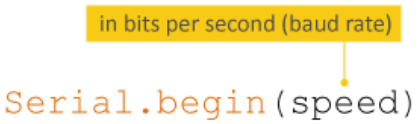
Serial.begin()

- **Description:** Sets the baud rate; the number of bits per second.
- **Note:**
  Make sure to use one of the baud rates available.

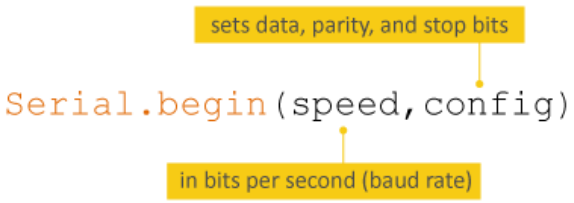
  An optional second argument configures the data, parity, and stop bits.

  For Boards like Leonardo which uses USB CDC serial ports Serial.begin() is irrelevant. You can use any baud rate and configuration for communication using these boards,

- **Syntax 1:**



- **Syntax 2:**

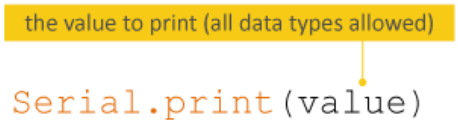


Serial.print()

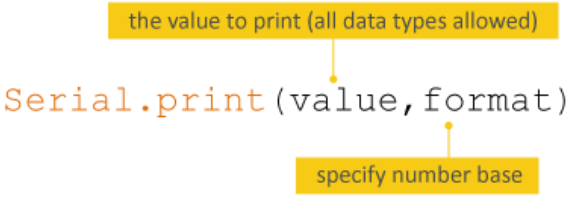
- **Description:** Prints data to the Serial port as a human-readable form.
- **Note:**
  You can also write it as Serial.println(value); this will print the data followed by a carriage return character '\r' or '\n'.

  you can pass flash-memory based strings to Serial.print() by wrapping them with F(). For example: Serial.print(F("Hello World"))

- **Syntax 1:**



- **Syntax 2:**

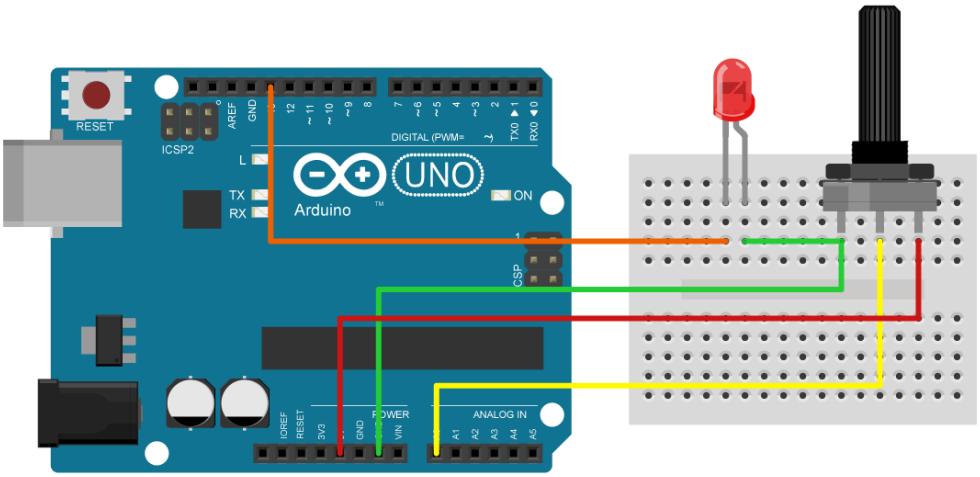


PROJECT 4a: **To read data from Potentiometer and control the speed of the blinking LED.**

Hardware:

- Arduino or Genuino board
- 10K Potentiometer
- 220-ohm resistor
- LED
- hook-up wires
- breadboard

Connections:



Program:

```
/* Analog Read to LED
 * ------------------
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13. The amount of time the LED will be on and off depends on
 * the value obtained by analogRead().
 */

int potPin = A0;        // input pin for the potentiometer
int ledPin = 13;        // LED pin
int val = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  val = analogRead(potPin);     // read sensor value
  val= val/4;
  digitalWrite(ledPin, HIGH);   // turn the ledPin on
```

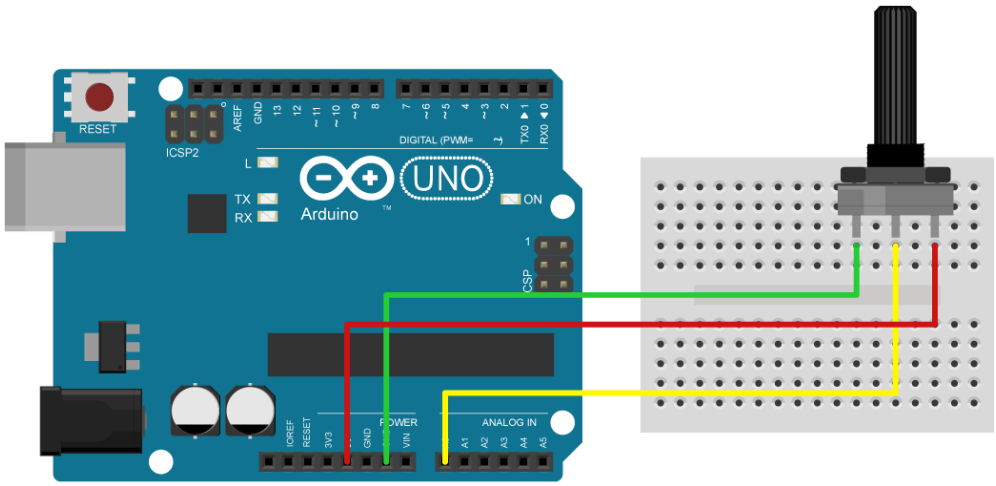
```
  delay(val);                    // stop the program for some time
  digitalWrite(ledPin, LOW);     // turn the ledPin off
  delay(val);                    // stop the program for some time
}
```

PROJECT 4b: **To read data from Potentiometer and display readings on Serial Monitor**

Hardware:

- Arduino or Genuino board
- 10K Potentiometer
- 220-ohm resistor
- hook-up wires
- breadboard

Connections:



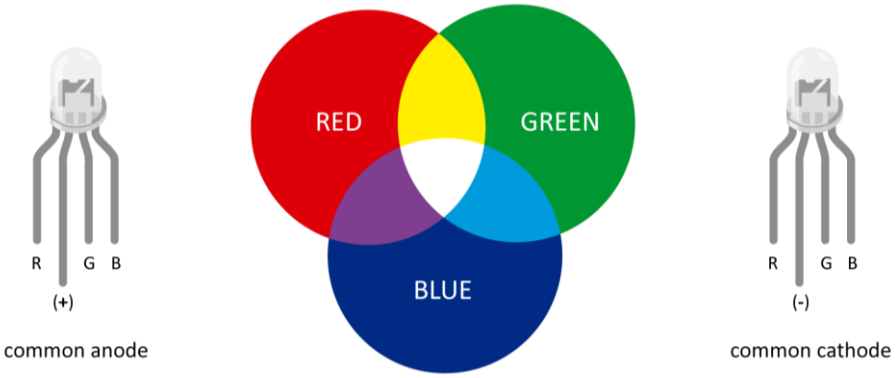
Program:

```
/* Analog Read to Serial Monitor
 * Display the data received on analog pin A0 on Serial Monitor
 */

int potPin = A0;    //input pin for the potentiometer
int val = 0;
void setup() {
  Serial.begin(9600);
}

void loop() {
  val = analogRead(potPin);    // read the value from the sensor
  Serial.println(val);         // print the values on serial monitor
}
```

11. Producing Colours Using RGB LED

Using RGB LED you can produce many colors by varying the intensities of Red Green and Blue using Pulse Modulated Signal from Arduino. Because LEDs are so close to each other that our eyes see the result of the combination of colors rather than individual color.

There are two types of RGB LEDs available; common Anode and Common Cathode:

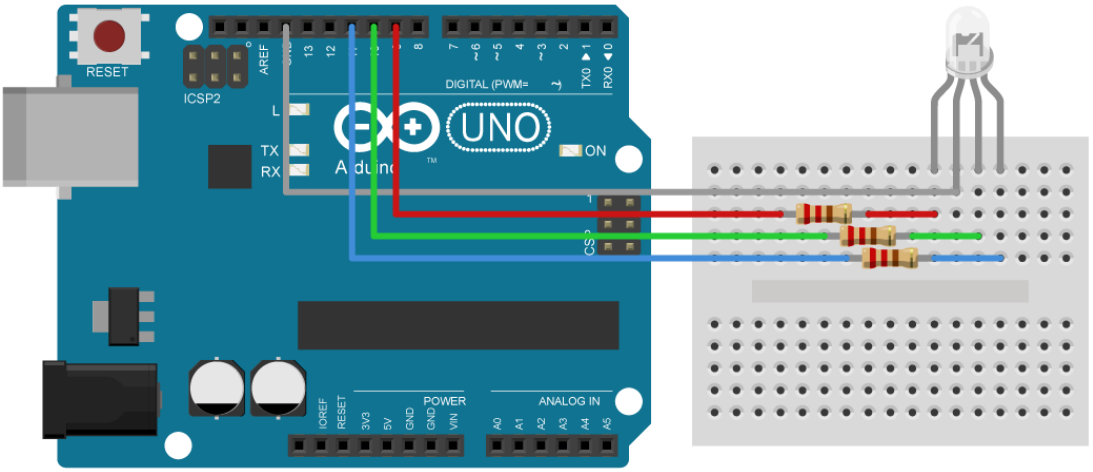


PROJECT 5: **Controlling colors of RGB LED**

Hardware:

- Arduino or Genuino board -1
- RGB LED -1
- 220-ohm resistor - 3
- hook-up wires
- breadboard

Connections:



Program 1:

```
// constants to name the pins
const int Red = 9;     const int Green = 10;  const int Blue = 11;

void setup() {}
void loop() {
analogWrite(Red,255);   //Glow Red
analogWrite(Green,0);
analogWrite(Blue,0);
delay(1000);
analogWrite(Red,0);
analogWrite(Green,255); //Glow Green
analogWrite(Blue,0);
delay(1000);
analogWrite(Red,0);
analogWrite(Green,0);
analogWrite(Blue,255);  //Glow Blue
delay(1000);
}
```

Program 2:

```
// variables to hold the LED color
int Rvalue = 254;      int Gvalue = 1;        int Bvalue = 127;
int Rdirection = -1;   int Gdirection = 1;    int Bdirection = -1;

// constants to name the pins
const int Red = 9;     const int Green = 10;  const int Blue = 11;
void setup() {
  pinMode(Red, OUTPUT);
  pinMode(Green, OUTPUT);
  pinMode(Blue, OUTPUT);
}

void loop() {
//send PWM signal on LEDs
analogWrite(Red, Rvalue);
analogWrite(Green, Gvalue);
analogWrite(Blue, Bvalue);

Rvalue = Rvalue + Rdirection;   //changing values of LEDs
Gvalue = Gvalue + Gdirection;
Bvalue = Bvalue + Bdirection;

//now change dirction for each color if it reaches 255 or 0
if (Rvalue>= 255 || Rvalue <= 0)
  {Rdirection = Rdirection * -1;}
if (Gvalue>= 255 || Gvalue <= 0)
  {Gdirection = Rdirection * -1;}
if (Bvalue>= 255 || Bvalue <= 0)
  {Bdirection = Bdirection * -1;}

//important: give some delay so that you can see the changes
delay(10);
}
```

12. Infra-Red Sensors

All objects which have a temperature greater than absolute zero (0 Kelvin) posses thermal energy and are sources of infrared radiation as a result.

What is an infrared sensor?

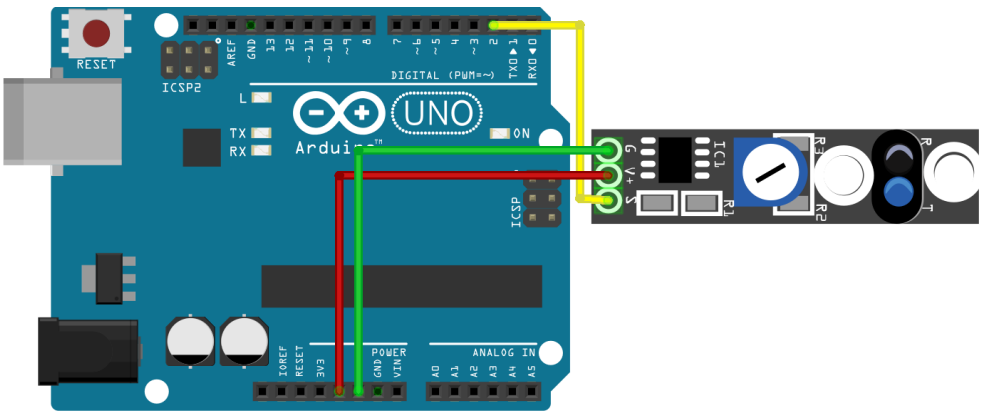
An infrared sensor is an electronic device, that emits IR rays in order to sense some aspects of the surroundings**.** This module contains a pair of IR Transmitter and receiver placed parallel to each other. As we know the White surface is a good reflector of radiation and the black surface is a good absorber. So when we keep these two surfaces in the proximity of this sensor we receive either High or Low voltage at the receiver. By reading these values we can come to know about the lightness or darkness of the color placed near it.

PROJECT 6: **Reading Values from IR Sensor and print them on Serial Monitor**

Hardware:

- Arduino or Genuino board -1
- IR Sensor Module - 1
- hook-up wires
- breadboard

Connections:

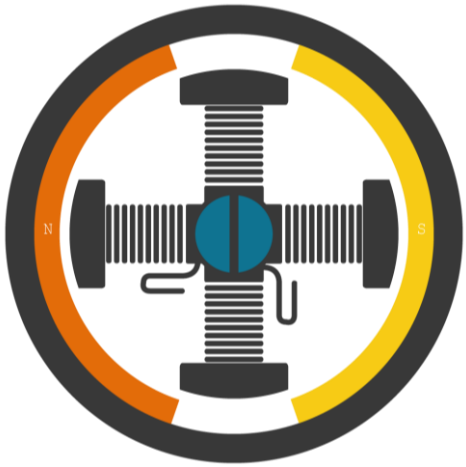


Program:

```
/*
 *  IR sensor module to Serial
 * Read values from IR sensor Module connected on pin 2 of Arduino.
 * Note: We are using digital IR sensor Module in this Example
 */

const int IR = 2;      //IR sensor signal pin
void setup() {
Serial.begin(9600);    //start Serial commuication at 9600 baud rate
pinMode(IR,INPUT);
}

void loop() {
int val = digitalRead(IR);  //read value and store it in val

if (val==HIGH)
{Serial.println("white surface");}

if (val==LOW)
{Serial.println("black surface");}

}
```

13. DC Motors

A motor is an electric machine that converts electric energy into mechanical energy. A normal dc motor produces torque using electromagnetic forces using the principle that when a current-carrying conductor is placed in a magnetic field it experiences a mechanical force.



DC motor consists of 2 wires if you connect these wires with opposite terminals of a battery, the motor starts rotating in either direction. To change the direction we can change the polarity on two wires.

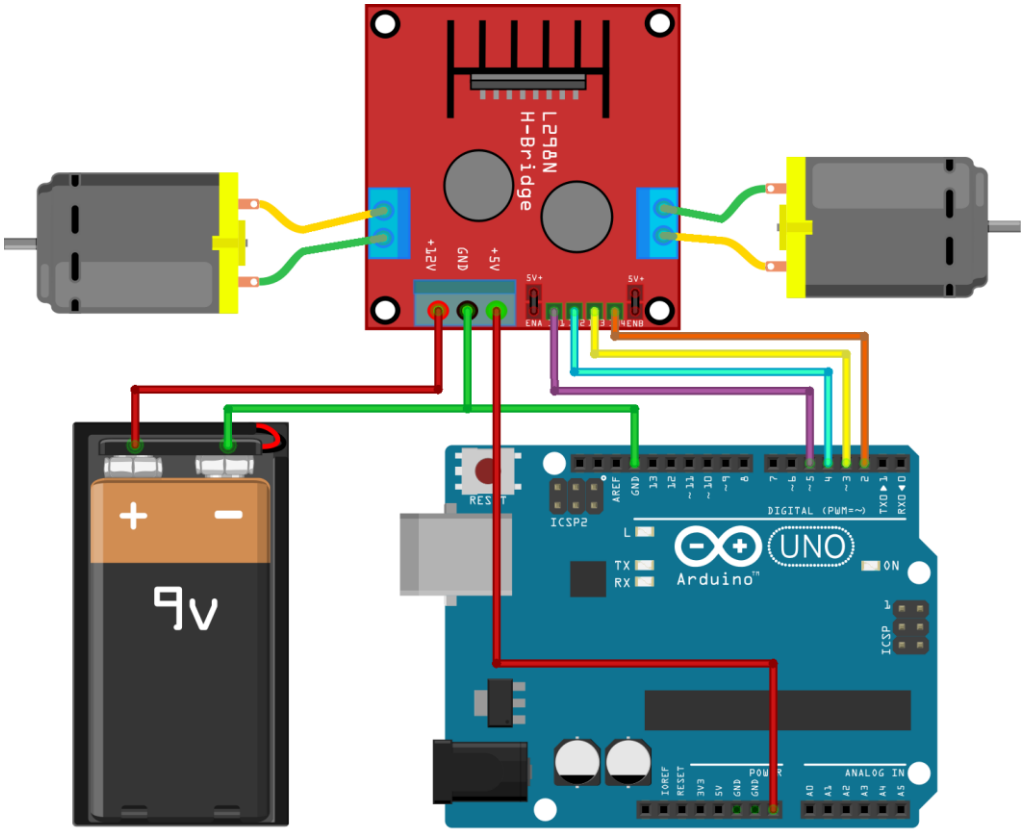
This motor can also be controlled using a microcontroller like Arduino but we need a motor driver for this purpose because the circuit operates on low voltages which are not enough to drive a dc motor. Moreover, if we try to do so the motor will produce heat and back EMF which can harm the circuit board.

PROJECT 7: **Rotate DC motors clockwise and anticlockwise using motor driver**

Hardware:

- Arduino or Genuino board -1
- L298N – 1
- DC motors - 2
- hook-up wires
- breadboard

Connections:



Program 1: To rotate dc motors clockwise and anti-clockwise

```
/* To rotate motors connected with pin 2, 3, 4 and 5 of the Arduino.
*/
const int Lmotor1 = 2;const int Lmotor2 = 3;
const int Rmotor1 = 4;const int Rmotor2 = 5;

void setup() {
pinMode(Lmotor1,OUTPUT);pinMode(Lmotor2,OUTPUT);
pinMode(Rmotor1,OUTPUT);pinMode(Rmotor2,OUTPUT);}

void loop() {
digitalWrite(Lmotor1,HIGH);digitalWrite(Lmotor1,LOW); //clockwise
digitalWrite(Rmotor2,LOW);digitalWrite(Rmotor2,HIGH);}//anticlockwise
```

Program 2: To vary the speed of motors using PWM

```
/* To vary the speed of motors connected with pin 6, 9, 10 and 11 of
 * the Arduino.
 */
const int Lmotor1 = 6;const int Lmotor2 = 9;
const int Rmotor1 = 10;const int Rmotor2 = 11;

void setup() {}
void loop() {
analogWrite(Lmotor1, 255);
analogWrite(Lmotor2, 0);       //rotate left motor with full speed
analogWrite(Rmotor1, 200;
analogWrite(Rmotor2, 0);}      //decrease the speed of right motor
```

14. Line Follower and Edge Avoider Robot

Line follower is a robot that follows a line, either a black line on a white surface or a white line on a black surface. The line follower robot moves on the line with a type of feedback mechanism. As we know, the IR sensor module is able to detect black and white colors. Now we will discuss the logic behind a line follower robot.

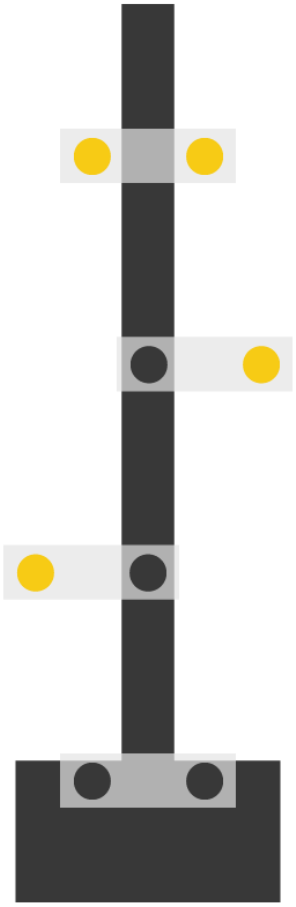
The logic of the Line Follower Robot

Case 1: **Both sensors on white-**
In this case, the robot has to move forward in order to follow the line

Case 2: **Right Sensor on white and Left sensor on black-**
In this case, the robot has to take the right turn in order to follow the line

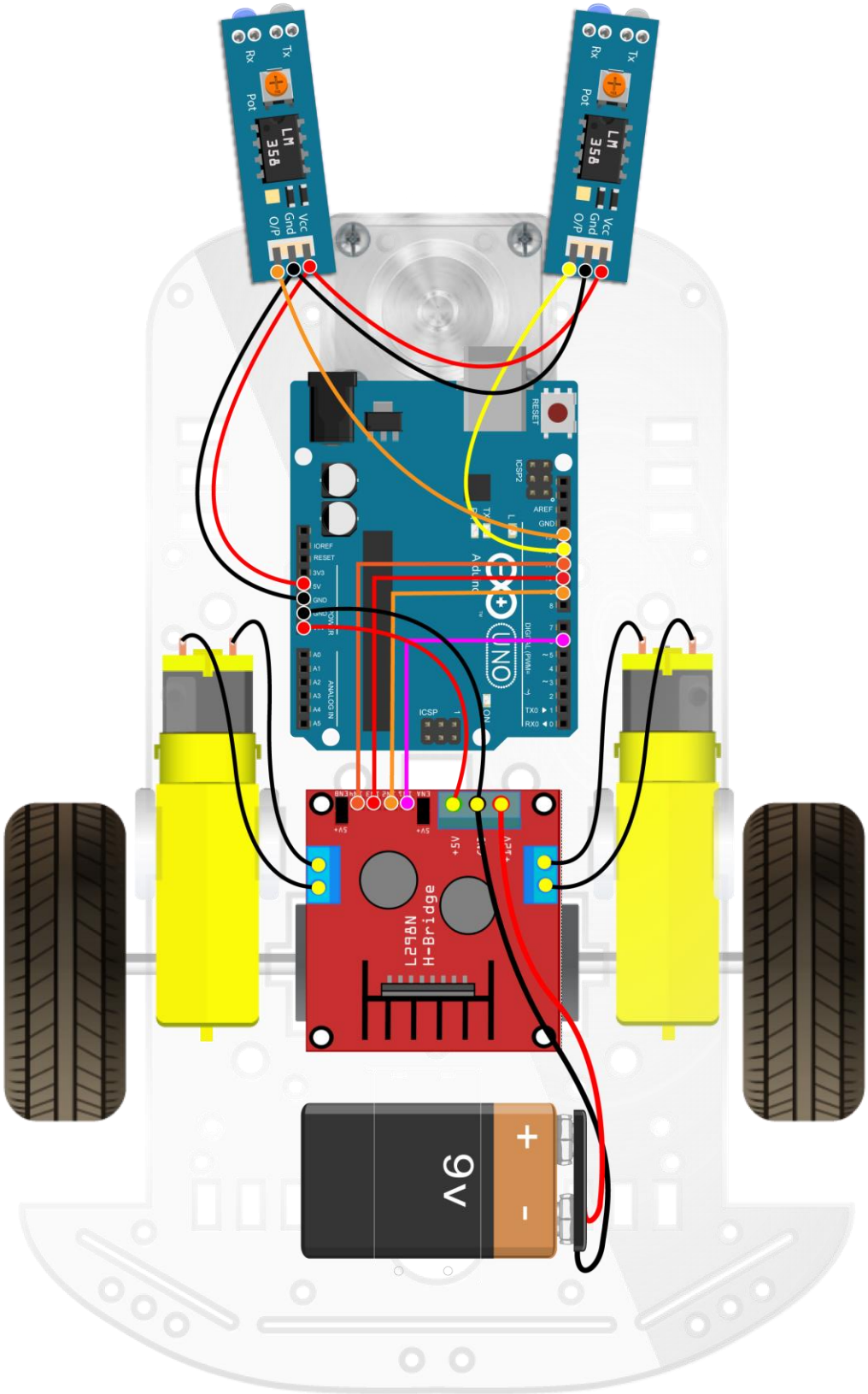
Case 3: **Right Sensor on black and Left sensor on white-**
In this case, the robot has to take the left turn in order to follow the line.

Case 3: **Both sensors on black-**
In this case, the robot can either stop or reverse the direction.



PROJECT 8a: **Line Follower Robot**

Connections and Components:



Program: Line Follower Robot

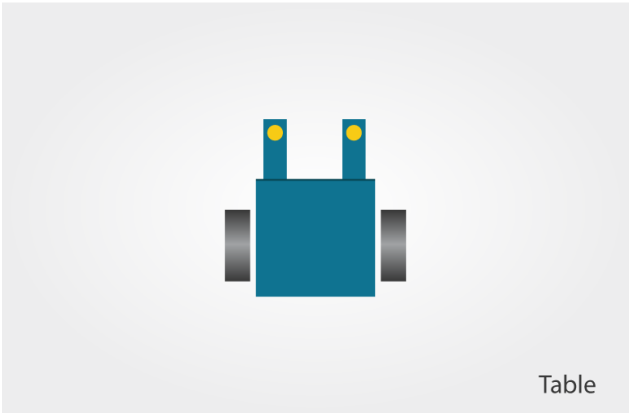
```
/*
 * LINE FOLLOWER ROBOT
 * Follows a black line on a White surface using sensors connected at
 * pin 12 and 13 of Arduino
 *
 */

const int Lmotor1 = 6;          //constants to store motor pins
const int Lmotor2 = 9;
const int Rmotor1 = 10;
const int Rmotor2 = 11;
const int Lsensor = 12;         //constants to store sensor pins
const int Rsensor = 13;

void setup() {
pinMode(Lmotor1,OUTPUT);
pinMode(Lmotor2,OUTPUT);
pinMode(Rmotor1,OUTPUT);
pinMode(Rmotor2,OUTPUT);
pinMode(Lsensor,INPUT);
pinMode(Rsensor,INPUT);
}

void loop() {
int Lvalue = digitalRead(Lsensor);  //read the value from left sensor
int Rvalue = digitalRead(Rsensor);  //read the value from right sensor

if(Lvalue==HIGH&&Rvalue==HIGH)      //if both sensors are on white
  { digitalWrite(Lmotor1,HIGH);     //move forward
    digitalWrite(Lmotor2,LOW);
    digitalWrite(Rmotor1,HIGH);
    digitalWrite(Rmotor2,LOW);
  }

if(Lvalue==HIGH&&Rvalue==LOW)       //if only left sensor is on white
  { digitalWrite(Lmotor1,HIGH);     //take turn left/right
    digitalWrite(Lmotor2,LOW);
    digitalWrite(Rmotor1,LOW);
    digitalWrite(Rmotor2,LOW);
  }

if(Lvalue==LOW&&Rvalue==HIGH)       //if only right sensor is on white
  { digitalWrite(Lmotor1,LOW);      //take turn left/right
    digitalWrite(Lmotor2,LOW);
    digitalWrite(Rmotor1,HIGH);
    digitalWrite(Rmotor2,LOW);
  }

if(Lvalue==LOW&&Rvalue==LOW)      //if both sensors are on black
  { digitalWrite(Lmotor1,LOW);     //stop
    digitalWrite(Lmotor2,LOW);
    digitalWrite(Rmotor1,LOW);
    digitalWrite(Rmotor2,LOW);
  }

}
```

PROJECT 8b: **Edge Avoider Robot**

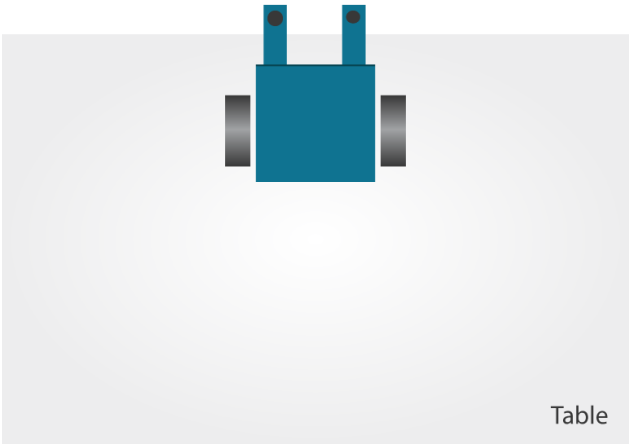
Edge avoider is a robot which can prevent itself from falling down from a table or any such surface by detecting edges using IR sensors. We only have to change the program in the previous robot to make it work as an Edge avoider robot.

Working Logic:

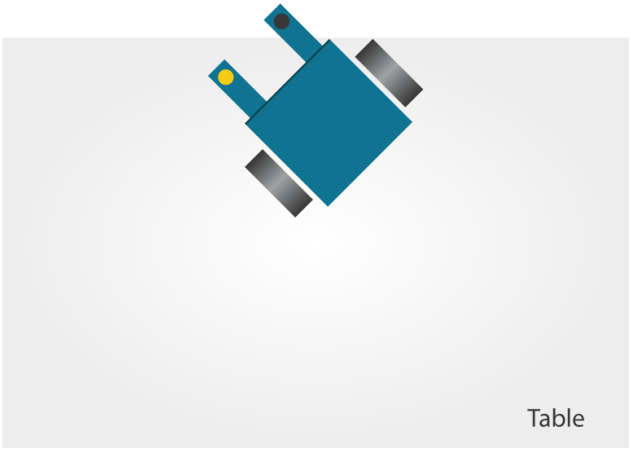
When a robot is kept on white table it will encounter these four cases. We will tell the robot what to do in each case.



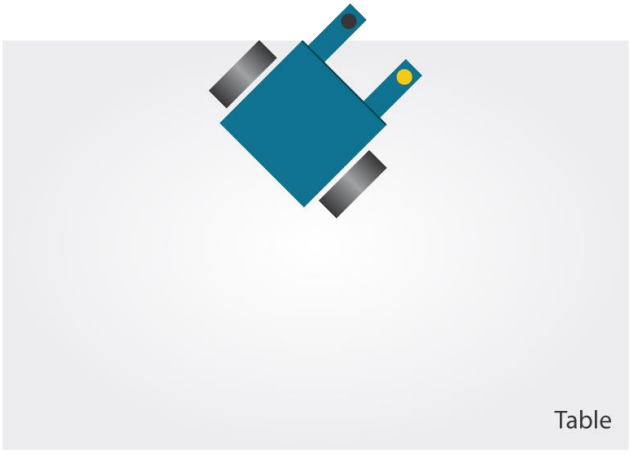
Case 1: When both sensors are on the table:
In this case, the robot can move forward because there is no edge



Case 2: When no sensor is on the table
In this case, the robot has to move backward for some time and then take a turn.



Case 3: When the only left sensor is on table
In this case, the robot has to move backward for some time and then take a turn.



Case 4: When only the right sensor is on table
In this case, the robot has to move backward for some time and then take a turn.

Notes:

- Try to use low rpm motors for this purpose. You can also vary the speed using PWM signal
- To increase the accuracy of this robot we can increase the number of IR sensors.

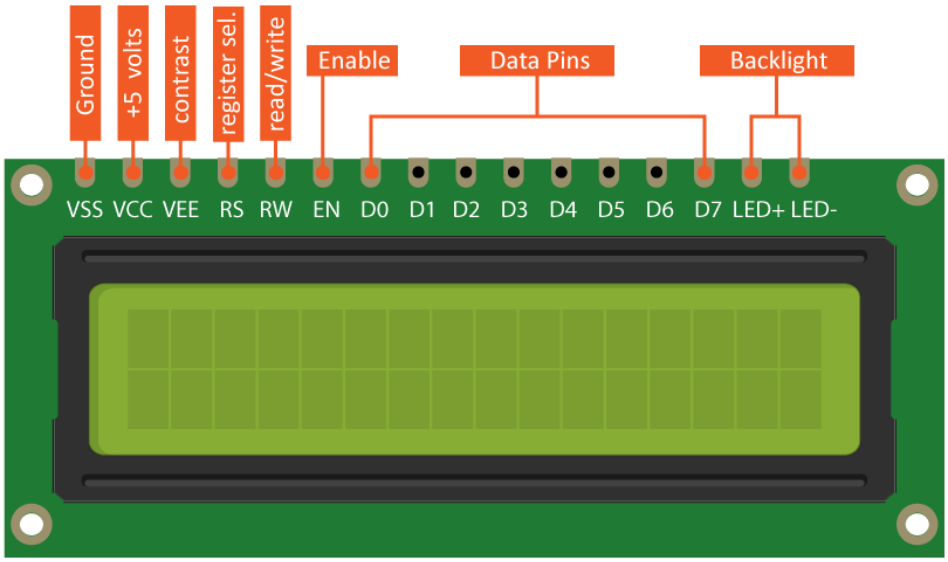
Program: Edge Avoider Robot

```
/*
 * EDGE AVOIDER ROBOT
 * Avoids edges and prevents itself from falling down
 *
 */

const int Lmotor1 = 6;            //constants to store motor pins
const int Lmotor2 = 9;
const int Rmotor1 = 10;
const int Rmotor2 = 11;
const int Lsensor = 12;          //constants to store sensor pins
const int Rsensor = 13;

int reverseDelay = 1000;         //change delay values according to your
robot
int turnDelay = 500;

void setup() {
pinMode(Lmotor1,OUTPUT);
pinMode(Lmotor2,OUTPUT);
pinMode(Rmotor1,OUTPUT);
pinMode(Rmotor2,OUTPUT);
pinMode(Lsensor,INPUT);
pinMode(Rsensor,INPUT);
}

void loop() {
int Lvalue = digitalRead(Lsensor);  //read the value from left sensor
int Rvalue = digitalRead(Rsensor);  //read the value from right sensor

if(Lvalue==HIGH&&Rvalue==HIGH)     //if both sensors are on the table
table
 { moveforward();                   //move forward
  }

if(Lvalue==HIGH&&Rvalue==LOW)    //if only left sensor is on the table
 { movebackward();
   delay(reverseDelay);             //give time to reverse
   moveright();                     //turn right
   delay(turnDelay);                //give time to turn
  }

if(Lvalue==LOW&&Rvalue==HIGH)    //if only right sensor is on the table
 { movebackward();
   delay(reverseDelay);             //give time to reverse
   moveleft();                      //turn left
   delay(turnDelay);                //give time to turn
  }

if(Lvalue==LOW&&Rvalue==LOW)     //both sensors are beyond table
 { movebackward();
   delay(reverseDelay);             //give time to reverse
   moveright();                     //turn right
   delay(turnDelay);                //give time to turn
  }
}
```

```
void moveforward()
{ digitalWrite(Lmotor1,HIGH);        //move forward
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}

void moveright()
{ digitalWrite(Lmotor1,HIGH);        //take right turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,LOW);
}

void moveleft()
{ digitalWrite(Lmotor1,LOW);         //take left turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}

void movebackward()
{ digitalWrite(Lmotor1,LOW);         //reverse
  digitalWrite(Lmotor2,HIGH);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,HIGH);
}
```

15. LCD 16x2 (Liquid Crystal Display)

LCD or Liquid Crystal Display is a display module which uses Liquid crystal to produce visible image used in various applications. These displays are super thin, cheap and consume less voltage. Here we will discuss the most commonly used 16x2 LCD. These LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display.



Pin Description:

Ground	Connect to the ground pin
+5 Volts	Connect to the +5 volt power supply
Contrast	This pin regulates the difference of the display, used to connect a changeable POT that can supply 0 to 5V.
Register Select	This pin toggles among command or data register used to connect a microcontroller unit pin and obtains either 0 or 1(0 = data mode, and 1 = command mode).
Read/Write	This pin toggles the display among the read or writes operation, and it is connected to a microcontroller unit pin to get either 0 or 1 (0 = Write, and 1 = Read).
Enable	This pin should be held high to execute Read/Write process, and it is connected to the microcontroller unit & constantly held high.
Data Pins (0-7)	These pins are used to send data to the display. These pins are connected in two-wire modes like 4-wire mode and 8-wire mode. In 4-wire mode, only four pins are connected to the microcontroller unit like 0 to 3, whereas in 8-wire mode, 8-pins are connected to a microcontroller unit like 0 to 7.

Libraries:

LiquidCrystal.h

- **Description:** This library allows an Arduino board to control LiquidCrystal displays. The library works within either 4- or 8-bit mode (i.e. using 4 or 8 data lines in addition to the rs, enable, and, optionally, the rw control lines).
- **Syntax:**

```
#include<LiquidCrystal.h>
```

Functions:

LiquidCrystal

- **Description:** Creates a variable of Type Liquid Crystal. The display can be used in either 4-bit mode or 8-bit mode. Omit d0,d1,d2,d3 pins and leave them disconnected to use it in 4-bit mode.
- **Note:** You can also omit rw pin and connect it to the ground.
- **Syntax:**

```
LiquidCrystal variable(rs,enable,d4,d5,d6,d7);
LiquidCrystal variable(rs,rw,enable,d4,d5,d6,d7);
LiquidCrystal variable(rs,enable,d0,d1,d2,d3,d4,d5,d6,d7);
LiquidCrystal variable(rs,rw,enable,d0,d1,d2,d3,d4,d5,d6,d7);
```

- **Parameters**

rs	Arduino pin which is connected to rs pin on the LCD
rw	Arduino pin which is connected to rw pin on the LCD
enable	Arduino pin which is connected to enable pin on the LCD
d0, d1, d2, d3, d4, d5, d6, d7	Arduino pins which are connected to d0,d1,d2,d3,d4,d5,d6,d7 of LCD

begin()

- **Description:** Used to initialize LCD interface and dimentions of the LCD (WxH).
- **Note:** It is compulsory to call .begin() before using any other library of `LiquidCrystal.h`
- **Syntax:**

```
lcd.begin(columns, rows);
```

- **Parameters:**
  lcd : a variable of type LiquidCrystal
  columns : the number of columns that the display has
  rows : the number of rows that the display has

setCursor()

- **Description:** brings the cursor to the position from where you want your text to appear
- **Syntax:**

```
lcd.setCursor(column,row);
```

- **Parameters:**
  lcd : a variable of type LiquidCrystal
  column : the column at which to position the cursor (with 0 being the first column)
  row : the row at which to position the cursor (with 0 being the first row)

print()

- **Description:** prints the text to LCD
- **Syntax:**

```
lcd.print(data);
```

- **Parameters:**
  lcd: a variable of type LiquidCrystal
  data: the data to print (char, byte, int, long, or string)

display() and noDisplay()

- **Description:** Used to turn ON and OFF the display without losing everything on it.
- **Syntax:**

```
Display();
noDisplay();
```

clear()

- **Description:** Clears the LCD
- **Syntax:**

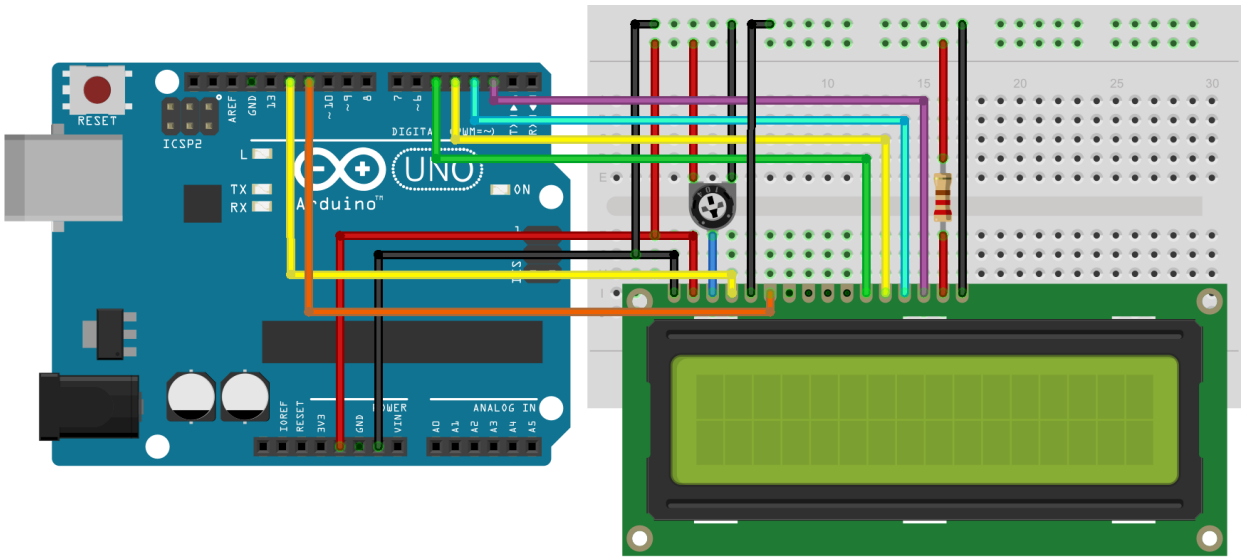
```
lcd.clear();
```

PROJECT 9: **Print Text on 16x2 LCD**

Hardware:

- Arduino or Genuino Board - 1
- LCD Screen (16x2) - 1
- 10k ohm potentiometer - 1
- 220 ohm resistor - 1
- hook-up wires
- breadboard

Connections:



Program 1: Print Hello World on LCD and Blink it

```
#include <LiquidCrystal.h>

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);  // rs,en,d4,d5,d6,d7

void setup() {
  lcd.begin(16, 2);              // LCD's number of columns and rows:
  lcd.setCursor(0,0);            // move cursor to require position
  lcd.print("hello, world!"); // Print a message to the LCD.
}

void loop() {
  lcd.noDisplay();               // Turn off the display
  delay(500);
  lcd.display();                 // Turn on the display
  delay(500);
}
```

Program 2: Print data received from the host computer through Serial Port

To use it, open Serial Monitor inside Arduino IDE and send some characters from there. The text will appear on the LCD.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // rs,en,d4,d5,d6,d7

void setup() {
  lcd.begin(16, 2);     // LCD columns and rows
  Serial.begin(9600);   // initialize the serial communications:
}

void loop() {
  if (Serial.available()) {           //check for data on Serial Port
    delay(100);                       //wait a bit
    lcd.clear();                      // clear the screen
    while (Serial.available() > 0) {  // read all the available data
      lcd.write(Serial.read());       // display characters received
    }
  }
}
```

Program 3: Set the LCD screen to automatically scroll the text

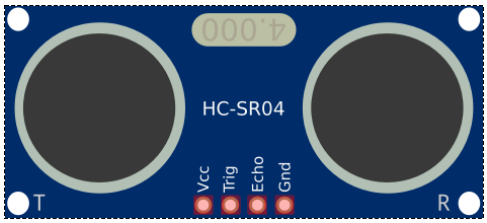
```
#include<LiquidCrystal.h>
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  lcd.begin(16, 2);
}

void loop() {
  lcd.setCursor(0, 0);
  // print from 0 to 9:

  for (int thisChar = 0; thisChar < 10; thisChar++) {
    lcd.print(thisChar);
    delay(500);

  }
  // set the cursor to (16,1):
  lcd.setCursor(16, 1);
  lcd.autoscroll();         //automatically scroll display
  for (int thisChar = 0; thisChar < 10; thisChar++) {
    lcd.print(thisChar);    //print data
    delay(500);
  }
  lcd.noAutoscroll();       //turnoff scrolling
  lcd.clear();              //clear the screen
}
```

16. Ultrasonic Sensor

Ultrasound is high-pitched sound waves with frequencies higher than the audible limit of human hearing.

What is an Ultrasonic sensor?

As the name indicates, ultrasonic sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception.



How Ultrasonic Sensors Work

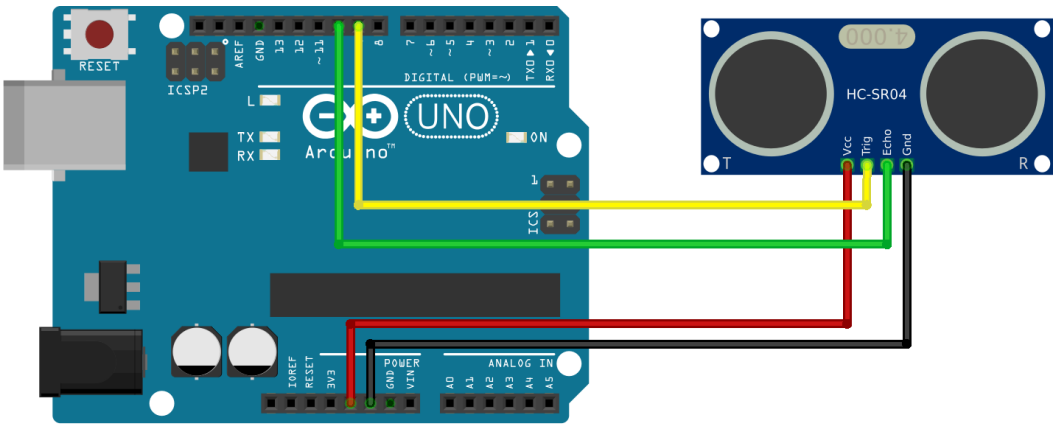
The HC-SR04 Ultrasonic distance sensor consists of two ultrasonic transducers. The one acts as a transmitter which converts the electrical signal into 40 kHz ultrasonic sound pulses. The receiver listens for the transmitted pulses. If it receives them it produces an output pulse whose width can be used to determine the distance the pulse traveled.

PROJECT 10a: **Measure distance using Ultrasonic sensor**

Hardware:

- Arduino or Genuino Board - 1
- Ultrasonic HC-SR04 - 1
- hook-up wires
- breadboard

Connections:



Program 1: To measure distance using ultrasonic sensor

```
/*
 * MEASURE DISTANCE USING ULTRASONIC SENSOR
 * ----------------------------------------
 * Used to measure distance of an obstacle using sound waves
 * Sensor used : HC-SR04
 *
 */


#define trigPin 9      //use 9th pin of Arduino as trigger pin
#define echoPin 10     //use 10th pin of Arduino as echo pin

void setup()
{
  Serial.begin(9600);       //start serial communication
  pinMode(trigPin,OUTPUT);  //set trigger pin as output
  pinMode(echoPin,INPUT);   //set echo pin as input
}

void loop()
{
  int duration, distance;     // variables to store duration and distance
  digitalWrite(trigPin,HIGH); // turn on trigger pin
  delayMicroseconds(10);      // wait 10 seconds
  digitalWrite(trigPin,LOW);  // turn off trigger pin
  duration = pulseIn(echoPin,HIGH); //store the duration for echo
  distance= (duration/2)/29.1;      //calculate the distance
  Serial.print(distance);           //print distance on serial moitor
  Serial.println("cm");
  delay(100);
}
```

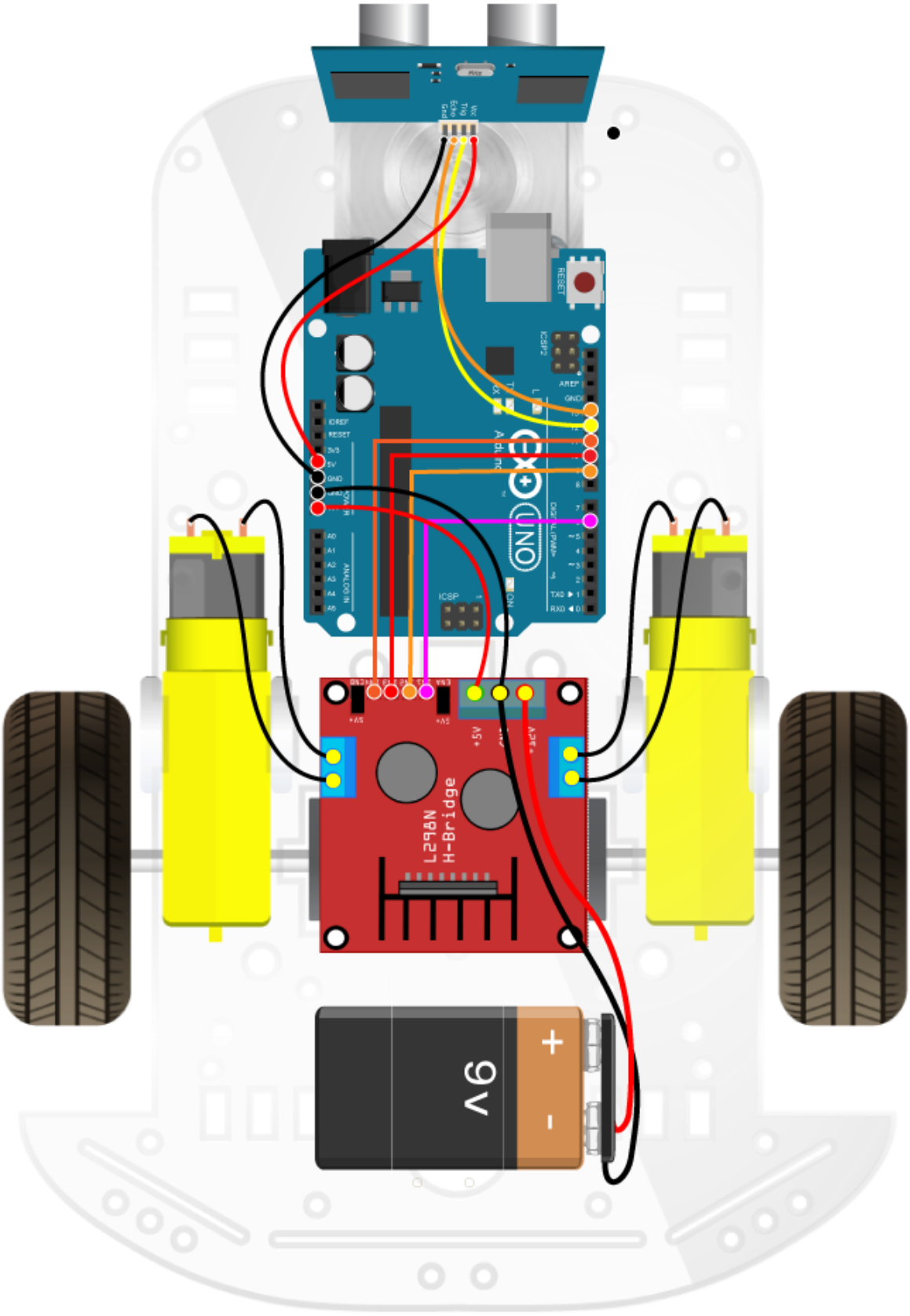
PROJECT 10b: **Obstacle avoider robot using ultrasonic sensor**

Obstacle avoider robot is an intelligent device which can navigate unknown environment without bumping into the obstacles. An ultrasonic sensor is mounted on the top of this robot which measures the distance of any obstacle using sound waves. If the sensor will detect any obstacle, the robot will stop and take a turn to avoid the collision.

Hardware:

- Arduino or Genuino Board - 1
- Ultrasonic HC-SR04 - 1
- DC geared Motors - 2
- Chassis – 1
- L298N/L293D Motor Driver – 1
- hook-up wires
- breadboard

Connections:



Program: Obstacle avoider robot using ultrasonic sensor

```
/*
 * OBSTACLE AVOIDER ROBOT
 * ----------------------------------
 * Avoids obstacles and prevents itself from collisions
 * Sensor used : HC-SR04
 *
 */

#define trigPin 12
#define echoPin 13

const int Lmotor1 = 6;            //constants to store motor pins
const int Lmotor2 = 9;
const int Rmotor1 = 10;
const int Rmotor2 = 11;

int reverseDelay = 1000;        //change delay values according to your
robot
int turnDelay = 500;

void setup()
{
  Serial.begin(9600);       //start serial communication
  pinMode(trigPin,OUTPUT);  //set trigger pin as output
  pinMode(echoPin,INPUT);   //set echo pin as input
}

void loop()
{
  int duration, distance;     // variables to store duration and distance
  digitalWrite(trigPin,HIGH); // turn on trigger pin
  delayMicroseconds(10);      // wait 10 seconds
  digitalWrite(trigPin,LOW);  // turn off trigger pin
  duration = pulseIn(echoPin,HIGH); //store the duration for echo
  distance= (duration/2)/29.1;      //calculate the distance
  Serial.print(distance);           //print distance on serial moitor
  Serial.println("cm");
  if (distance<=30)
  { movebackward();
    Serial.println("turning");
    delay(reverseDelay);             //give time to reverse
    moveright();                     //turn right
    delay(turnDelay);                //give time to turn
  }
  else
  {
     moveforward();
   }
}


void moveforward()
{ digitalWrite(Lmotor1,HIGH);     //move forward
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}
```

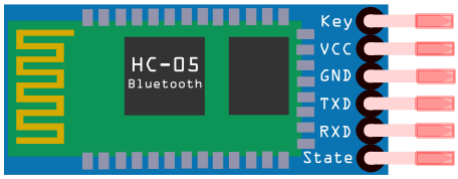
```
void moveright()
{ digitalWrite(Lmotor1,HIGH);      //take right turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,LOW);
}

void moveleft()
{ digitalWrite(Lmotor1,LOW);      //take left turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}

void movebackward()
{ digitalWrite(Lmotor1,LOW);      //reverse
  digitalWrite(Lmotor2,HIGH);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,HIGH);
}
```

17. Bluetooth

The wireless technology used to transmit data for short distances at high speed and using very little power using UHF radio waves. Bluetooth operates at frequencies between 2.402 and 2.480 GHz, or 2.400 and 2.4835 GHz including guard bands 2 MHz wide at the bottom end and 3.5 MHz wide at the top.

Bluetooth Module HC-05

HC-05 is an easy to use SPP(Serial Port Protocol) device designed to transmit data between two points. It's a transceiver module, which means it is possible to transmit and receive the data using the same module.



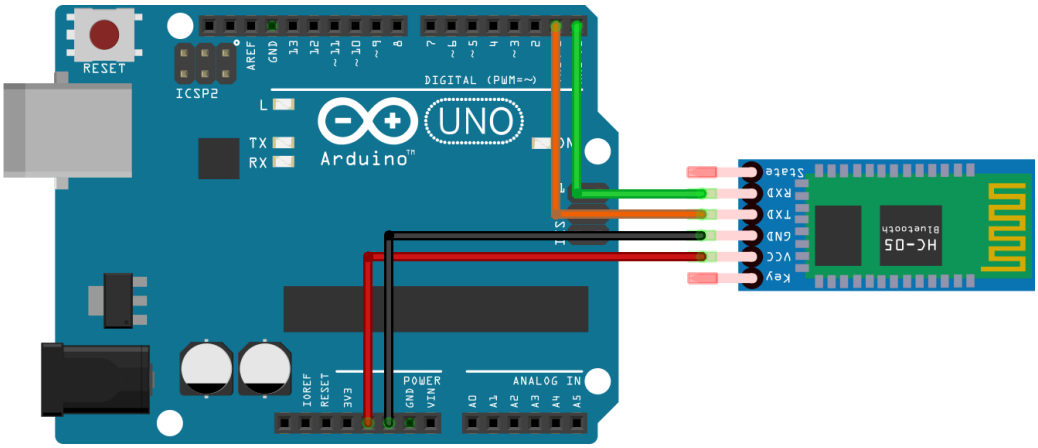
Getting Started with Bluetooth:

Features:

- Default Baud rate: 38400
- Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.
- Auto-connect to the last device on power as default.
- Auto-pairing PINCODE:"0000" or "1234" as default
- Low Power 1.8V Operation,1.8 to 3.6V I/O.
- UART interface with programmable baud rate
- With integrated antenna

Settings
AT Command Mode. AT command mode allows you to interrogate the Bluetooth module and to change some of the settings; things like the name, the baud rate, whether or not it operates in slave mode or master mode. When used as a master device AT commands allow you to connect to other Bluetooth slave devices

Step 1.
Upload a Blank Sketch to Arduino Board.
Step 2.
Now while Holding small Botton on the HC-05 Module, make the following connection:

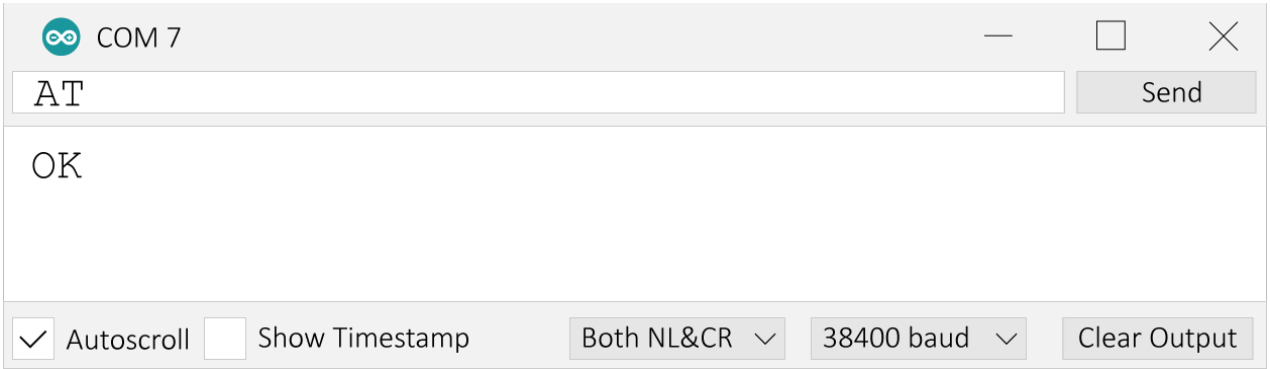


After making all the connections you can release the button.

Note: If there is no button on the Bluetooth module then instead of pressing the button you can connect EN pin to +5V and proceed with the same steps from step 2.

Step 3:
Open Arduino IDE and Press **ctrl+shift+M** to open the Serial Monitor

Step 5:
Make sure all the settings as shown below:
baud rate = **38400**.
Select **Both NL&CR** on third last box
Now Type "AT" and press "Send".

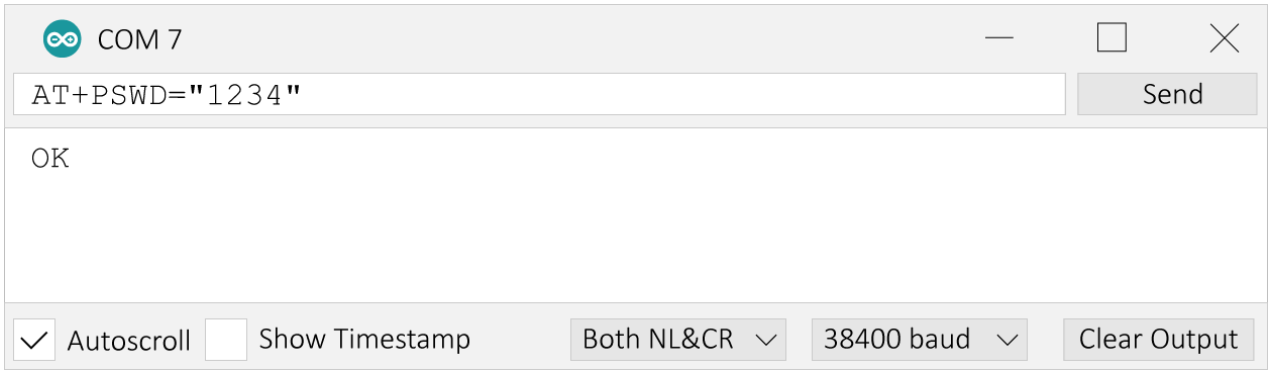


After you receive "OK" as a response. Now you can proceed with following commands

To change the name of the Bluetooth device:

To know the Address of Bluetooth Module

To Change the Password of Bluetooth Module



Restore to default

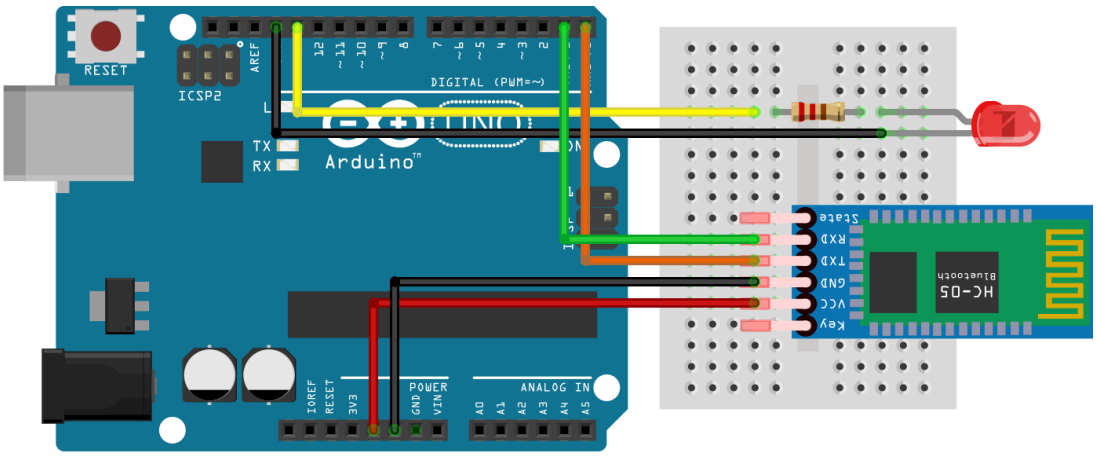
PROJECT 11a: **Controlling LED using the Android app**

Using Android App we can turn LED ON and OFF through Bluetooth. The app should be able to transmit data using the mobile's Bluetooth.

Hardware:

- Arduino or Genuino Board - 1
- Bluetooth HC-05 - 1
- LED - 1
- 220ohm resistor – 1
- hook-up wires
- breadboard
- A smartphone

Connections:



Mobile App:

Scan the code to download the mobile app

Program: LED Blinking using Mobile App

```
/*
 * CONTROLLING LED USING BLUETOOTH
 * -----------------------------------
 * This example demonstrates how we can control a LED using Mobile App
 * with the help of Bluetooth HC-05
 */
const int LED = 13;                 //declare LED pin
char val;

void setup() {
Serial.begin(9600);                 //start serial communication
pinMode(LED,OUTPUT);                //set pinmode as output
}

void loop() {
if(Serial.available()>0)            //check if it is receiving data or not
 { val = Serial.read();             //read and store data inside val
   Serial.println(val);
   if(val=='1')                     //if 1 is received turn on the LED
   {
    digitalWrite(LED,HIGH);
    Serial.println("LED ON");
   }
   if(val=='0')                     //if 0 is received turn off the LED
   {
    digitalWrite(LED,LOW);
    Serial.println("LED OFF");
    }
  }
}
```

PROJECT 11b: **Controlling LED using Voice**

Using Android App we can turn LED ON and OFF

Hardware and Connections: Same as Project 11b.

Mobile App:

Scan the code to download the mobile app

Program: Control LED using voice Commands

```
/*
 * CONTROLLING LED USING Voice COMMANDS
 * ------------------------------------
 * Demonstrates how we can control a LED using voice commands
 * with the help of Bluetooth HC-05
 *
 */

const int LED = 13;                 //declare LED pin
String val;                         //variable to store data from
bluetooth
void setup() {
Serial.begin(9600);                //start serial communication
pinMode(LED,OUTPUT);               //set pinmode as output
}

void loop() {
if(Serial.available()>0)                //check if it is receiving data
or not
 { val = Serial.readStringUntil('\n');  //read and store data inside val
   Serial.println(val);
   if((val=="turn on the light")||(val=="turn on"))
   {
    digitalWrite(LED,HIGH);
    Serial.println("LED ON");
   }
   if((val=="turn of the light")||(val=="turn off"))
   {
    digitalWrite(LED,LOW);
    Serial.println("LED OFF");
    }
  }
}
```

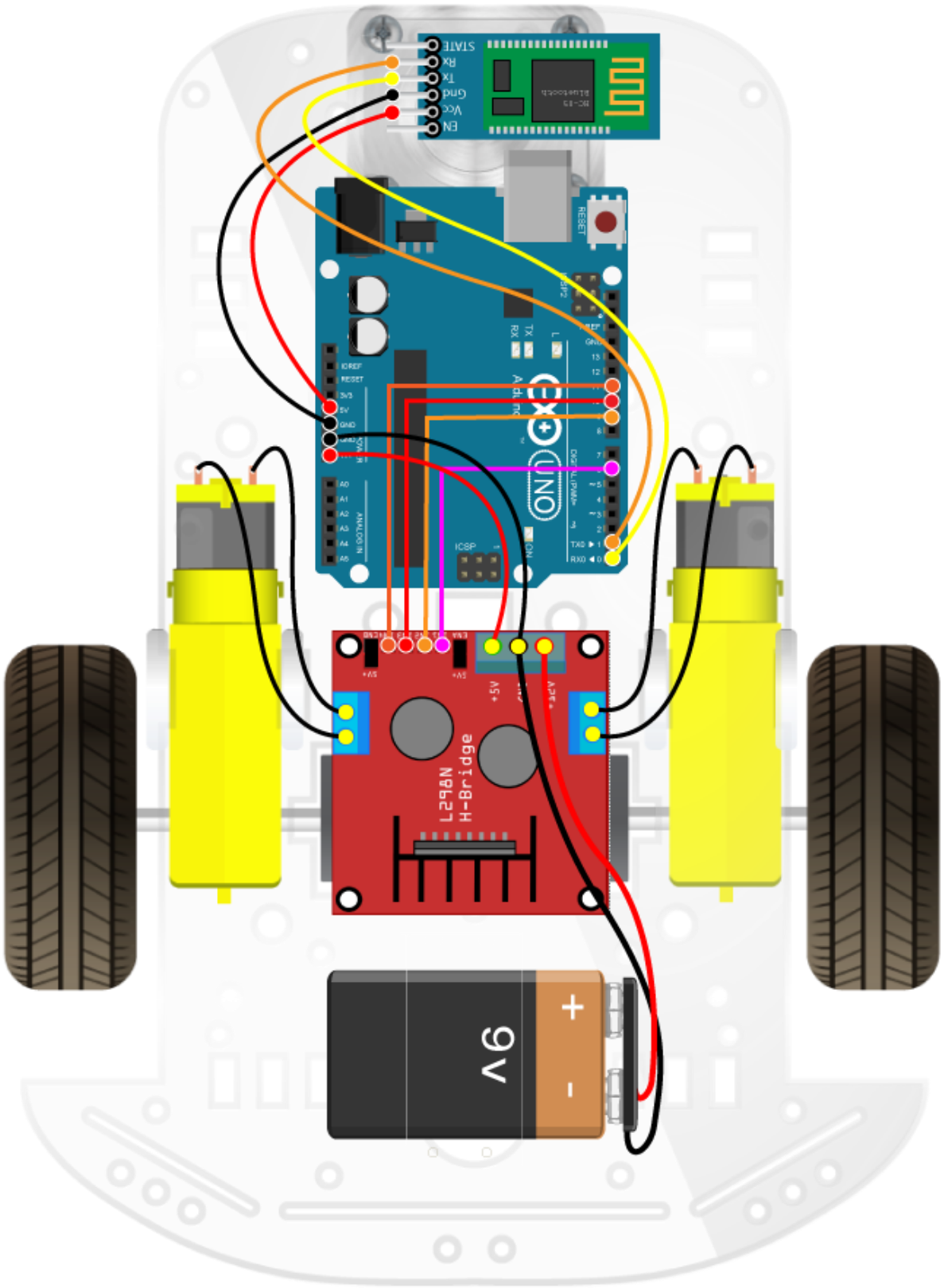
PROJECT 11c: **Remote controlled car using Bluetooth**

We can use an android app to control the movement of a robotic car either by using buttons or by accelerometer. There are various apps available on Googel playstore which can be used to transmit this data using Bluetooth.

Hardware:

- Arduino or Genuino Board - 1
- Bluetooth HC-05 - 1
- DC geared Motors - 2
- Chassis – 1
- L298N/L293D Motor Driver – 1
- hook-up wires

Connections:



Mobile App:

Scan the code to download the mobile app

Program: Bluetooth controlled Robot

```
/*
 * BLUETOOTH CONTROLLED ROBOT
 * Control the robotic car using Mobile App.
 * Data transmission through Bluetooth Module HC-05
 *
 */

const int Lmotor1 = 6;              //constants to store motor pins
const int Lmotor2 = 9;
const int Rmotor1 = 10;
const int Rmotor2 = 11;
char incomingByte;

void setup() {
Serial.begin(9600);
pinMode(Lmotor1,OUTPUT);
pinMode(Lmotor2,OUTPUT);
pinMode(Rmotor1,OUTPUT);
pinMode(Rmotor2,OUTPUT);
}

void loop() {
 if (Serial.available()>0)
  {
    incomingByte = Serial.read();

    switch(incomingByte)
    {
      case 'u' : Serial.println("Forward");moveforward();break;
      case 'd' : Serial.println("Backward");movebackward();break;
      case 'r' : Serial.println("Right");moveright();break;
      case 'l' : Serial.println("Left");moveleft();break;
      case 'h' : Serial.println("forward");halt();break;
      default  : break;
    }
 }
}
```

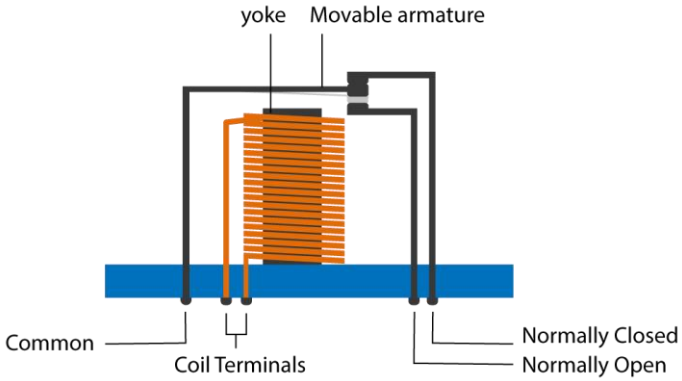
```
//functions

void moveforward()
{ digitalWrite(Lmotor1,HIGH);     //move forward
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}

void moveright()
{ digitalWrite(Lmotor1,HIGH);     //take right turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,LOW);
}

void moveleft()
{ digitalWrite(Lmotor1,LOW);     //take left turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}

void halt()
{
  digitalWrite(Lmotor1,LOW);     //stop
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,LOW);
}

void movebackward()
{ digitalWrite(Lmotor1,LOW);     //reverse
  digitalWrite(Lmotor2,HIGH);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,HIGH);
}
```

18. Home Automation using Bluetooth

Home Automation can be used to control lighting and electrical appliances in a smart home. We can control AC devices using an app on a mobile phone. As we know devices like Arduino works on 5 Volt dc power supply, whereas electrical appliances run on 220-250 Volts of ac power supply. So we need this device called relay which works on 5volts dc signal and controls devices using 220 volts of ac current.

What is Relay?

Relay is a switch which used to open and close a circuit electromechanically. The main operation of relay is to make or break contact without any human involvement in order to switch it ON and OFF. It is generally used to control high powered circuit using a low powered circuit. Generally, a DC signal is used to control circuit which is driven by high voltage like controlling AC home appliances with DC signals from microcontrollers.



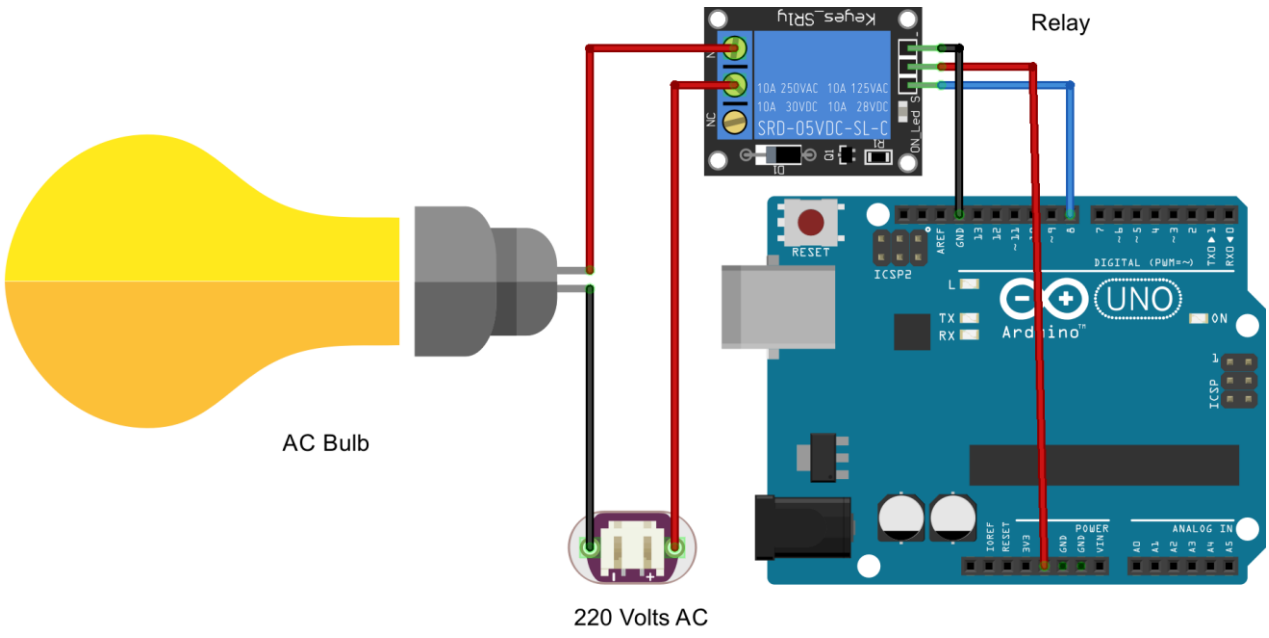
According to the diagram when we provide supply to coil terminals it acts as an electromagnet and attracts the movable armature and gets connected to the Normally open pin. we can say that when a coil is energized the armature is attracted and the switching action can be seen if the coil is de-energized it loses its magnetic property and the armature goes back to its initial position.

PROJECT 12: **Turning a bulb ON and OFF using Arduino**

Hardware:

- Arduino or Genuino Board – 1
- Relay Module - 1
- Power Cord – 1
- Bulb Holder
- AC Bulb = 1
- hook-up wires

Connections:



Program: For turning this Bulb ON and OFF we just have to use our first program of LED Blinking.

```
/*
 * Turns on and off a light bulb connected to digital
 * pin 8.
 */

void setup() {
  pinMode(8, OUTPUT);     // sets the digital pin 8 as output
}

void loop() {
  digitalWrite(8, HIGH); // sets the digital pin 8 on
  delay(1000);            // waits for a second
  digitalWrite(8, LOW);  // sets the digital pin 8 off
  delay(1000);            // waits for a second
}
```

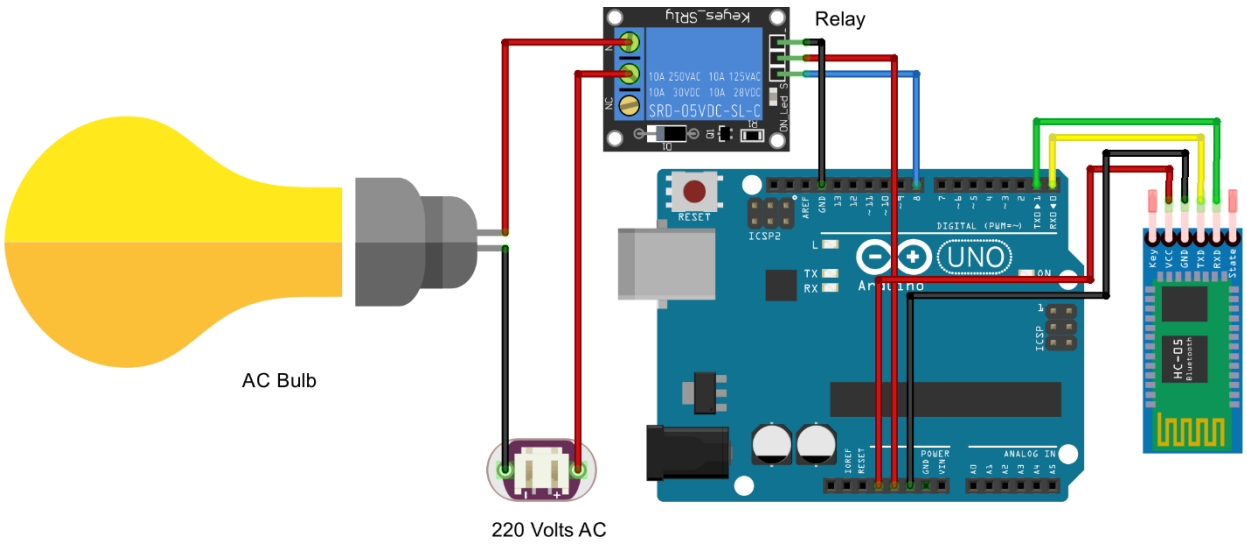
PROJECT 13: Home Automation using Mobile App and Bluetooth

We can control our home appliances using Android app via HC-05 Bluetooth Module.

Hardware:

- Arduino or Genuino Board – 1
- Relay Module – 1
- Bluetooth Module HC-05 - 1
- Power Cord – 1
- Bulb Holder - 1
- AC Bulb - 1
- hook-up wires

Connections:



Mobile App:

Scan the code to download the mobile app

Program:

```
/*
 * CONTROLLING BULB USING BLUETOOTH
 * ------------------------------------
 * This example demonstrates how we can control a Bulb using Mobile App
 * with the help of Bluetooth HC-05
 *
 */

const int BULB = 8;                //declare relay control pin
char val;

void setup() {

Serial.begin(9600);                //start serial communication

pinMode(BULB,OUTPUT);              //set pinmode as output

}

void loop() {
if(Serial.available()>0)           //check if it is receiving data or not
{ val = Serial.read();            //read and store data inside val

Serial.println(val);

if(val=='a')                     //if 1 is received turn on the BULB
    {
     digitalWrite(BULB,HIGH);
     Serial.println("BULB ON");
    }

if(val=='b')                     //if 0 is received turn off the BULB
    {
     digitalWrite(BULB,LOW);
     Serial.println("BULB OFF");
      }

   }
}
```

19. Soil Moisture Sensor

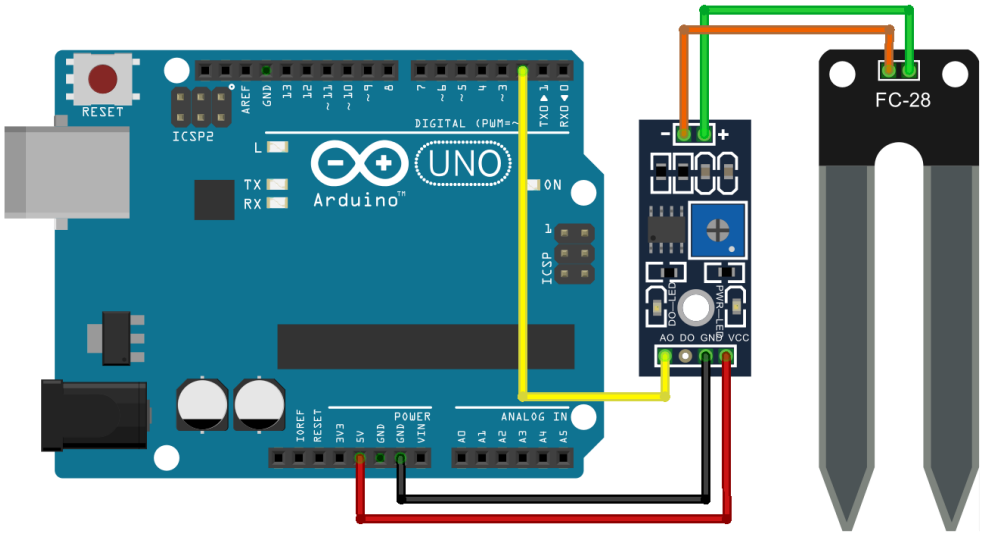
The Soil Moisture Sensor uses capacitance to measure dielectric permittivity of the surrounding medium. In soil, dielectric permittivity is a function of the water content. The sensor creates a voltage proportional to the dielectric permittivity, and therefore the water content of the soil. The sensor averages the water content over the entire length of the sensor.

PROJECT 14: **To measure the Moisture of soil and display readings on Serial Monitor**

Hardware:

- Arduino or Genuino Board – 1
- Soil Moisture Sensor - 1
- hook-up wires

Connections:



Program:

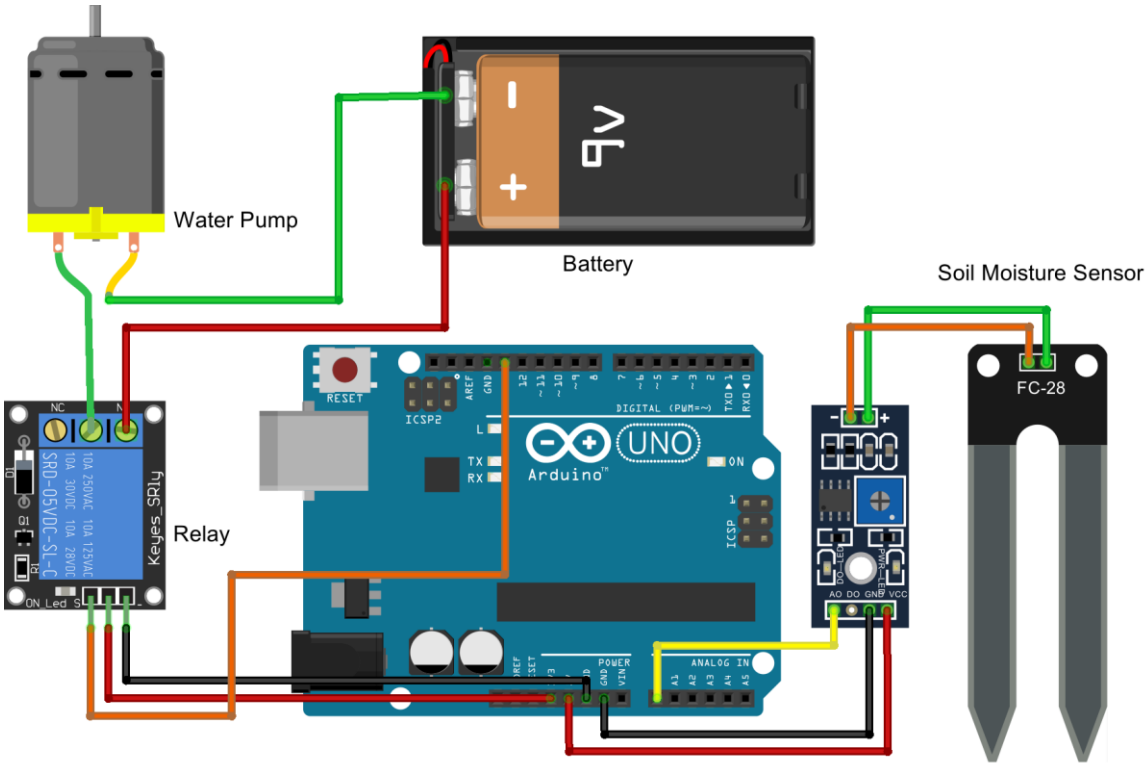
```
/*
 * SOIL MOISTURE SENSOR
 * ---------------------------
 * This program reads the readings of a soil moisture sensor
 * and display them on serial monitor
 */

int sensor_pin = A0;       //name of the sensor pin to use
int val = 0;               //variable to store sensor readings
void setup() {
Serial.begin(9600);
}

void loop() {
val = analogRead(A0);      //start reading values fro the sensor
Serial.println(val);       //print these values on Serial Monitor
}
```

PROJECT 15: **Automatic Irrigation System using Soil Moisture Sensor**

Hardware:

- Arduino or Genuino Board – 1
- Soil Moisture Sensor – 1
- Water Pump - 1
- 9V Battery - 1
- hook-up wires

Connections:



Program:

```
/*
 * IRRIGATION SYSTEM USING SOIL MOISTURE SENSOR
 * ---------------------------------------------
 * This program reads the readings of a soil moisture sensor
 * and moisture of the soil is not as expected, it will turn on the
 * water pump
 *
 */

int sensor_pin = A0;          //name of the sensor pin to use
int pump = 13;                //name of the water pump pin

int val = 0;                  //variable to store sensor readings
```

```
int requiredMoisture = 100; //Enter value for required moisture

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

void loop() {
val = analogRead(A0);        //start reading values fro the sensor
Serial.println(val);         //print these values on Serial Monitor

if(val<requiredMoisture)     //if moisture is less than requred value
{digitalWrite(pump,HIGH);}     //Turn on the water pump

}
```

20. Accelerometer

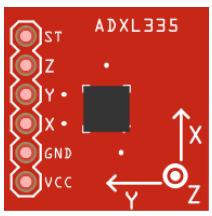
Acceleration is a measure of how fast velocity changes. It is the name we give to any process where the velocity changes. Since velocity is a speed and a direction, there are only two ways for you to accelerate: change speed or change direction—or change both.

In mathematical terms, acceleration is the change of velocity divided by the change of time.

What is an accelerometer?

An accelerometer is an electromechanical device used to measure acceleration forces. For example, when your mobile phone automatically changes screen orientation when you hold it vertically or horizontally is because of this sensor.

We are going to use **ADXL335** which is easily available in the market

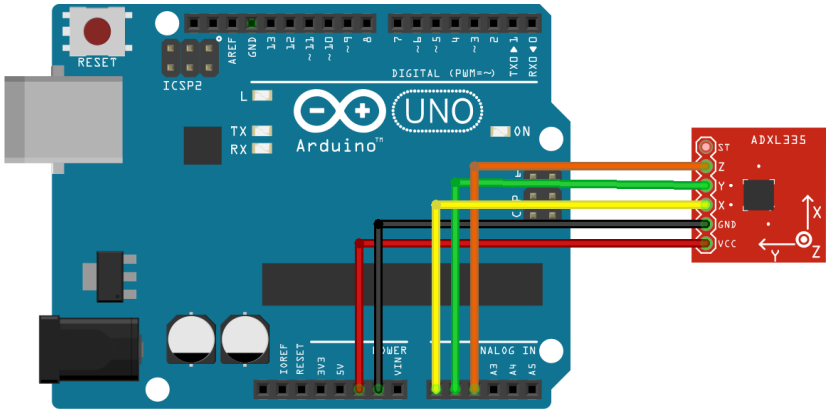


PROJECT 16a: **Reading Values from accelerometer and display on Serial Monitor**

Hardware:

- Arduino or Genuino Board – 1
- Accelerometer ADXL335 -1
- hook-up wires

Connections:



Program:

```
/*
 * INTERFACE ACCELEROMETER WITH ARDUINO
 * ------------------------------------------------
 * This program reads values of X, Y and Z axis of the accelerometer
 * and display them on Serial Monitor
 *
 */

int xPin = A0;            //define X,y and Z axis pins
int yPin = A1;
int zPin = A2;

int xVal;
int yVal;
int zVal;   //variables to store data of x, y and z-axis

void setup() {

Serial.begin(9600);     //Start Serial monitor
}

void loop() {

xVal=analogRead(A0);    //read value from x axis
yVal=analogRead(A1);    //read value from y axis
zVal=analogRead(A2);    //read value from z axis

//print all the values on Serial Monitor

Serial.print("x: ");Serial.print(xVal);
Serial.print('\t');
Serial.print("y: ");Serial.print(yVal);
Serial.print('\t');
Serial.print("z: ");Serial.print(zVal);
Serial.println("");

}
```

PROJECT 16b: **Accelerometer controlled car using ADXL335 and HC-05 Bluetooth Module.**

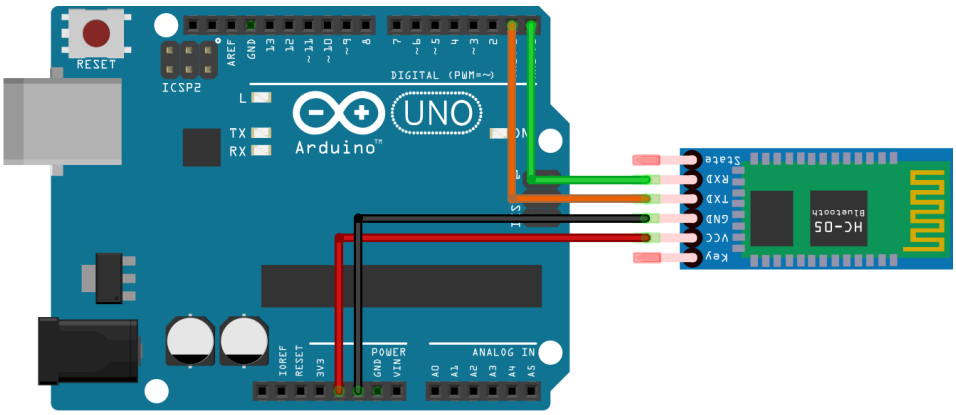
Hardware:

- Arduino Boards – 2 *(preferred: Arduino Uno - 1, Arduino Nano - 1)*
- Accelerometer ADXL335 -1
- HC-05 Bluetooth - 2
- hook-up wires

Theory:

Before starting we have to use one of the Bluetooth in Master Mode and the other one in slave mode. To do so you have to follow these steps:

- Upload Blank Sketch on Arduino Board.
- Connect the First HC-05 Bluetooth Module with Arduino Uno as shown below and follow the steps below:



SLAVE MODE

a. Remove Ground (GND) wire.
b. Press and Hold Enable button on HC-05 Bluetooth Module.
c. Plug Ground Wire back to its pin.
d. Release Enable button on HC-05 Bluetooth Module.
e. Open Serial Monitor and follow the steps below.

f. After sending “AT” you should receive “OK”.
g. Type “AT+UART?” to check the baud rate.

h. Type “AT+ROLE=0” to use this module as **SLAVE**

i. Type “AT+ADDR?” to know the address of the **SLAVE**.

- Now connect second HC-05 Module with Arduino Board to configure master mode.

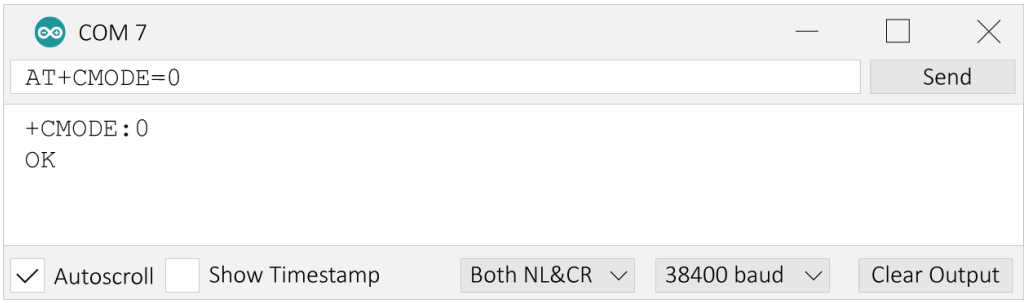
MASTER MODE

a. Remove Ground (GND) wire.
b. Press and Hold Enable button on HC-05 Bluetooth Module.
c. Plug Ground Wire back to its pin.
d. Release Enable button on HC-05 Bluetooth Module.
e. Open Serial Monitor follow the steps below.

f. Type “AT+UART?” to check the baud rate.

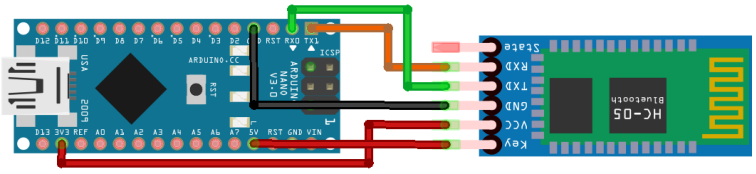
g. Type “AT+ROLE=1” to use this module as **MASTER.**

h. Type “AT+CMODE=0” which means it will only connect to a fixed address.

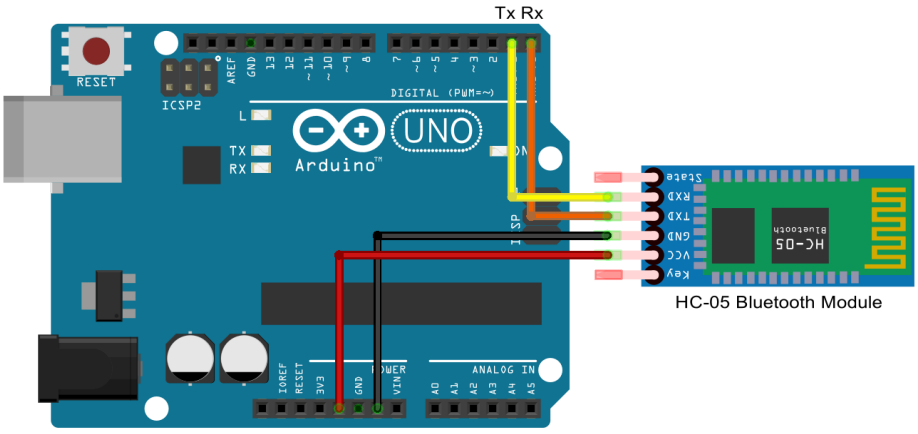


i. Type “AT+BIND=19,8,350CBD” to bind it with Slave’s address.

- After finishing all the steps for both **Master** and **Slave** mode reconnect both the modules as shown below:



Master Side



Slave Side

- Now we will upload a test program on both devices to check the connectivity
- **Test Program for Master**

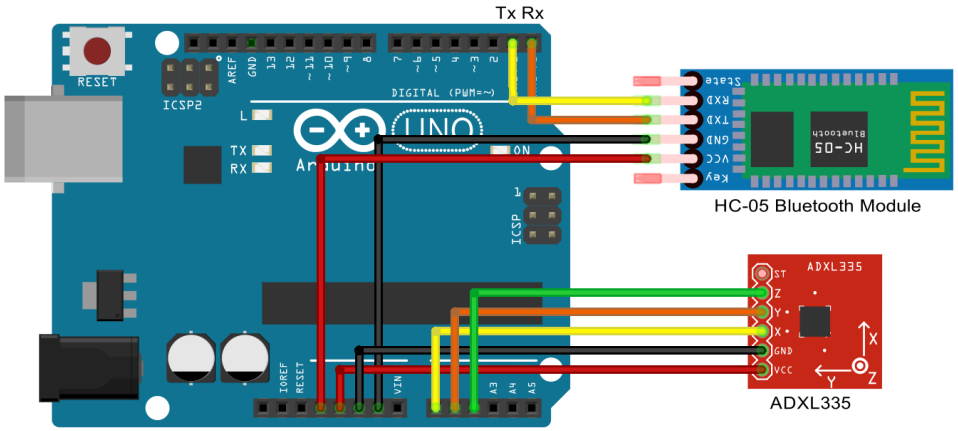
```
//Test Program for Master
void setup() {
Serial.begin(9600); //start serial communication
}
void loop() {
Serial.write("a");  //Transmit alphabet 'a'
}
```

- **Test Program for Slave**

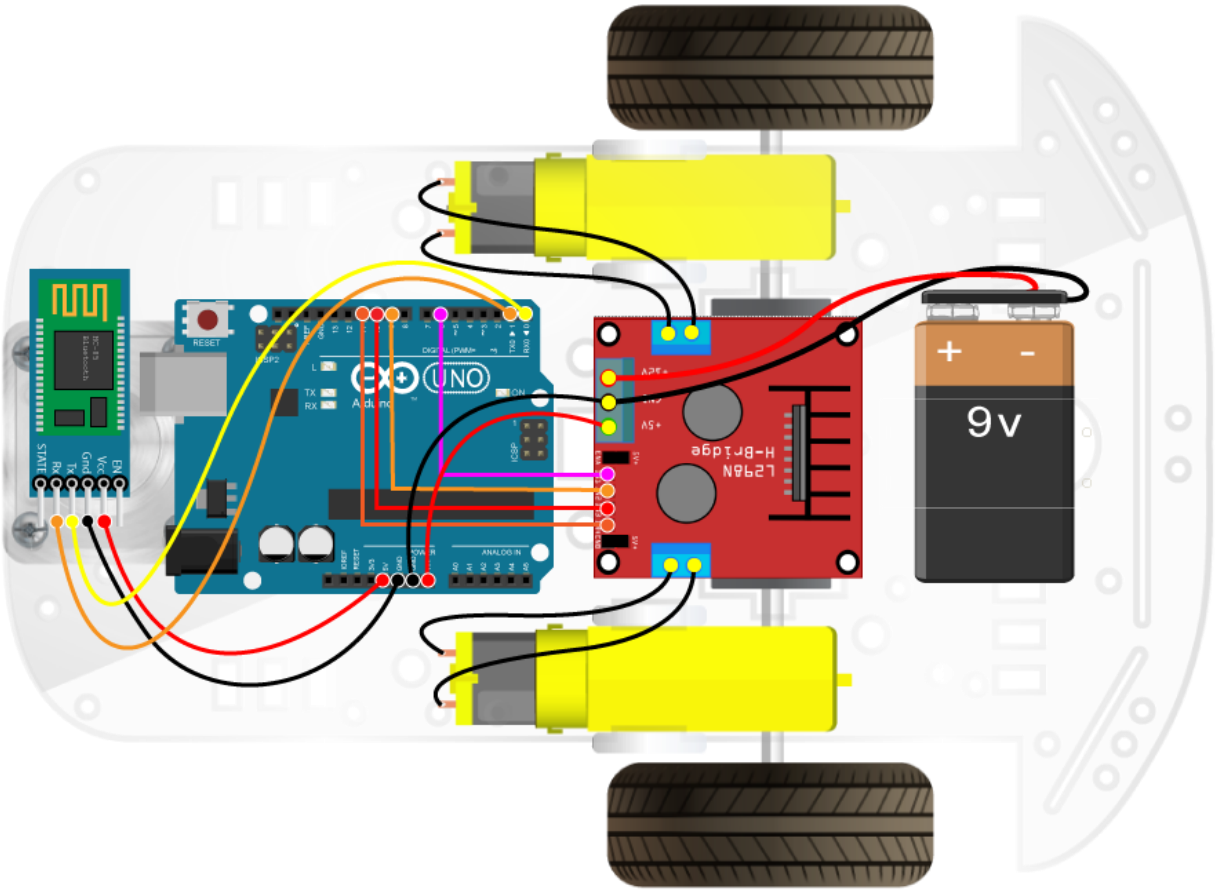
```
// Test program for Slave
void setup() {
Serial.begin(9600);   //Start Serial Communication @9600 baud
}
void loop() {
if(Serial.available()>0)
{
 char val=Serial.read();  //read Serial data and store in val
 Serial.println(val);     //display data on Serial Monitor
}
}
```

Connections for Accelerometer Controlled Robotic Car:

Transmitter Side:



Receiver Side:



Program for Transmitter:

```
/*
 * ACCELEROMETER CONTROLLED ROBOT
 * This robot receives data over bluetooth and moves the robot in
 * particula direction accordingly
 */

int xPin=A0;      //variable to store x axis
int yPin=A1;      //variable to store y axis


void setup(){
  Serial.begin(9600);   //start serial port
}


void loop()
{
  int xval=analogRead(xPin);  //read value from x axis
  int yval=analogRead(yPin);  //read value from y axis

  if ((xval>325 && xval<330) && (yval>320 && yval<335)) //stop
  {
    //Serial.println("stop");
    Serial.write("s");
  }

  else
  {
    if ((xval>330 && xval<342) && (yval>320 && yval<335)) //forward
    {
     //Serial.println("forward");
     Serial.write("f");
    }

    if ((xval>267 && xval<320) && (yval>320 && yval<335)) //backward
    {
     //Serial.println("backward");
     Serial.write("b");
    }

    if ((xval>332 && xval<334) && (yval>335 && yval<385)) //left
    {
     //Serial.println("left");
     Serial.write("l");
    }

    if ((xval>332 && xval<334) && (yval>273 && yval<325))//right
    {
     // Serial.println("right");
     Serial.write("r");
    }
  }
  Serial.flush();
}
```

Program for Receiver:

```
/*
 * BLUETOOTH CONTROLLED ROBOT
 * Control the robotic car using Mobile App.
 * Data transmission through Bluetooth Module HC-05
 */
const int Lmotor1 = 6;            //constants to store motor pins
const int Lmotor2 = 9;
const int Rmotor1 = 10;
const int Rmotor2 = 11;
char incomingByte;

void setup() {
Serial.begin(9600);
pinMode(Lmotor1,OUTPUT);
pinMode(Lmotor2,OUTPUT);
pinMode(Rmotor1,OUTPUT);
pinMode(Rmotor2,OUTPUT);
}

void loop() {
 if (Serial.available()>0)
  {
    incomingByte = Serial.read();

    switch(incomingByte)
    {
      case 'f' : Serial.println("Forward");moveforward();break;
      case 'b' : Serial.println("Backward");movebackward();break;
      case 'r' : Serial.println("Right");moveright();break;
      case 'l' : Serial.println("Left");moveleft();break;
      case 's' : Serial.println("forward");halt();break;
      default  : break;
    }
 }
}
//functions

void moveforward()
{ digitalWrite(Lmotor1,HIGH);     //move forward
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}

void moveright()
{ digitalWrite(Lmotor1,HIGH);     //take right turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,LOW);
}

void moveleft()
{ digitalWrite(Lmotor1,LOW);     //take left turn
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,HIGH);
  digitalWrite(Rmotor2,LOW);
}
```

```
void halt()
{
  digitalWrite(Lmotor1,LOW);      //stop
  digitalWrite(Lmotor2,LOW);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,LOW);
}

void movebackward()
{ digitalWrite(Lmotor1,LOW);      //reverse
  digitalWrite(Lmotor2,HIGH);
  digitalWrite(Rmotor1,LOW);
  digitalWrite(Rmotor2,HIGH);
}
```