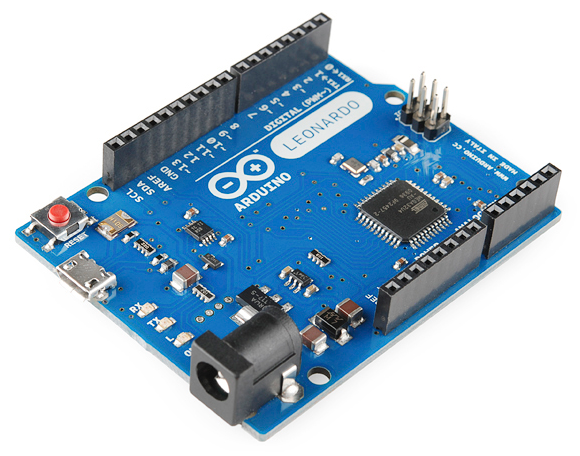
1. Arduino uno r3



# What is an Arduino?

## *Introduction*

[Arduino](http://arduino.cc/) is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a [microcontroller](http://en.wikipedia.org/wiki/Microcontroller)) and a piece of [software](http://arduino.cc/en/Main/Software), or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.



The Arduino UNO is a microcontroller board, based on the ATmega8/168/328, UNO R3 major in ATmega328.

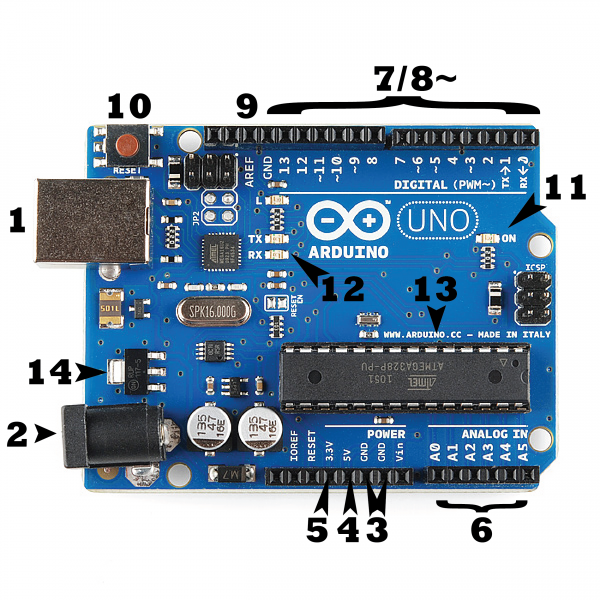
Arduino UNO R3 have 14 digital input/output pins (which include about 6 pins PWM output), 6 analog inputs and one 16MHz ceramic resonator, one USB connection and Power jack, an ICSP header (In Circuit Serial Programming) (like MAX232, RS232 programming).

TABLE:-

|  |  |
| --- | --- |
| Microcontroller | ATmega328 |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limits) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 40 Ma |
| DC Current for 3.3V Pin | 50 Ma |
| Flash Memory | 32 KB (ATmega328) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Clock Speed | 16 MHz |

## What's on the board?

There are many varieties of Arduino boards ([explained on the next page](https://learn.sparkfun.com/tutorials/what-is-an-arduino/the-arduino-family)) that can be used for different purposes. Some boards look a bit different from the one below, but most Arduinos have the majority of these components in common:



### Power (USB / Barrel Jack)

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply ([like this](https://www.sparkfun.com/products/8269)) that is terminated in a barrel jack. In the picture above the USB connection is labeled **(1)** and the barrel jack is labeled **(2)**.

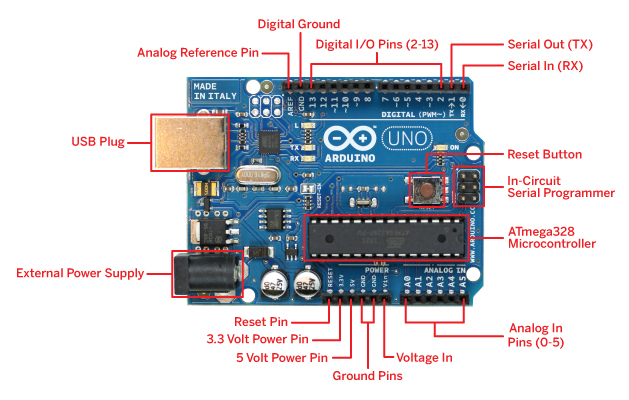
The USB connection is also how you will load code onto your Arduino board. More on how to program with Arduino can be found in our [Installing and Programming Arduino](https://learn.sparkfun.com/tutorials/installing-arduino-ide) tutorial.

**NOTE:** Do NOT use a power supply greater than 20 Volts as you will overpower (and thereby destroy) your Arduino. The recommended voltage for most Arduino models is between 6 and 12 Volts.

**Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF)**

The pins on your Arduino are the places where you connect wires to construct a circuit (probably in conjuction with a [breadboard](https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/) and some [wire](https://learn.sparkfun.com/tutorials/working-with-wire). They usually have black plastic ‘headers’ that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

* **GND (3)**: Short for ‘Ground’. There are several GND pins on the Arduino, any of which can be used to ground your circuit.
* **5V (4) & 3.3V (5)**: As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.
* **Analog (6)**: The area of pins under the ‘Analog In’ label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a [temperature sensor](https://www.sparkfun.com/products/10988)) and convert it into a digital value that we can read.
* **Digital (7)**: Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED).
* **PWM (8)**: You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM). We have [a tutorial on PWM](https://learn.sparkfun.com/tutorials/pulse-width-modulation), but for now, think of these pins as being able to simulate analog output (like fading an LED in and out).
* **AREF (9)**: Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

**Reset Button**

Just like the original Nintendo, the Arduino has a reset button **(10)**. Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if your code doesn’t repeat, but you want to test it multiple times. Unlike the original Nintendo however, blowing on the Arduino doesn’t usually fix any problems.

### Power LED Indicator

Just beneath and to the right of the word “UNO” on your circuit board, there’s a tiny LED next to the word ‘ON’ **(11)**. This LED should light up whenever you plug your Arduino into a power source. If this light doesn’t turn on, there’s a good chance something is wrong. Time to re-check your circuit!

### TX RX LEDs

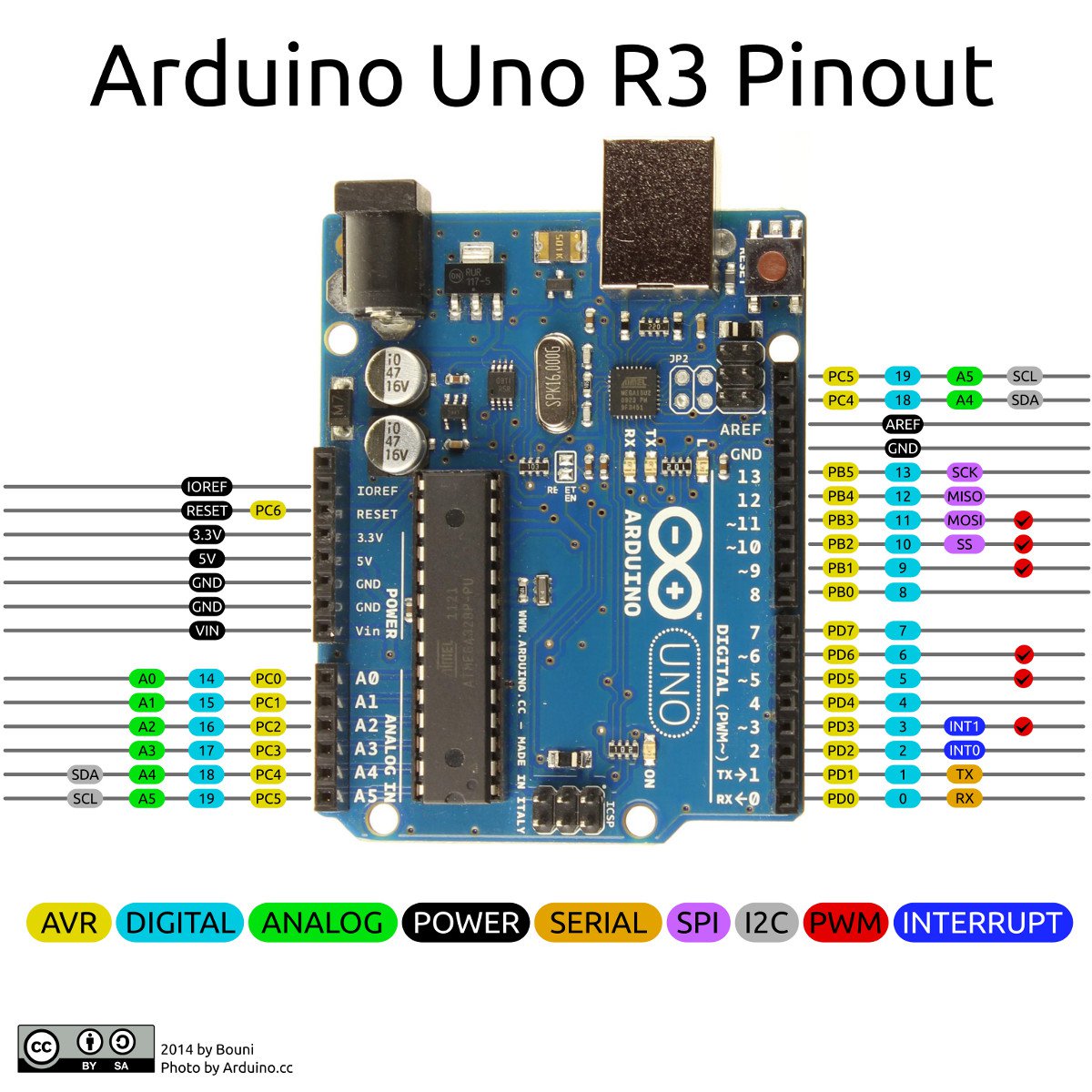
TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for [serial communication](https://learn.sparkfun.com/tutorials/serial-communication). In our case, there are two places on the Arduino UNO where TX and RX appear – once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs **(12)**. These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data (like when we’re loading a new program onto the board).

### Main IC

The black thing with all the metal legs is an IC, or Integrated Circuit **(13)**. Think of it as the brains of our Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC’s from the ATMEL company. This can be important, as you may need to know the IC type (along with your board type) before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC. If you want to know more about the difference between various IC’s, reading the datasheets is often a good idea.

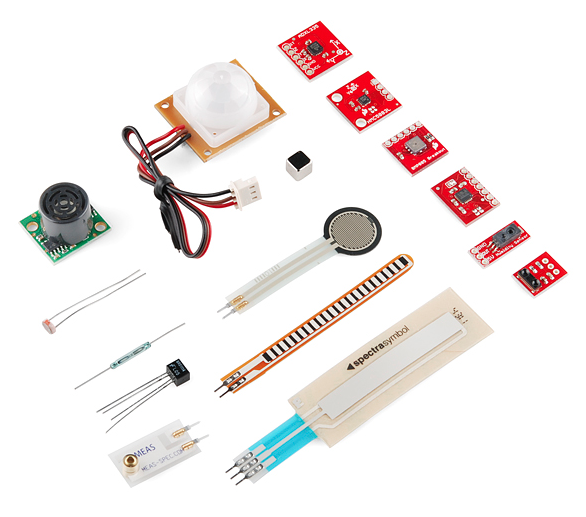
### Voltage Regulator

The voltage regulator **(14)** is not actually something you can (or should) interact with on the Arduino. But it is potentially useful to know that it is there and what it’s for. The voltage regulator does exactly what it says – it controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, so don’t hook up your Arduino to anything greater than 20 volts.



### Sensors

With some simple code, the Arduino can control and interact with a wide variety of **sensors** - things that can measure [light](https://www.sparkfun.com/products/9088), [temperature](https://www.sparkfun.com/products/10988), [degree of flex](https://www.sparkfun.com/products/8606), [pressure](https://www.sparkfun.com/products/11207), [proximity](https://www.sparkfun.com/products/242), [acceleration](https://www.sparkfun.com/products/9836), [carbon monoxide](https://www.sparkfun.com/products/9403), [radioactivity](https://www.sparkfun.com/products/11345), [humidity](https://www.sparkfun.com/products/9569), [barometric pressure](https://www.sparkfun.com/products/9721), [you name it](https://www.sparkfun.com/products/11574), [you can sense it](https://www.sparkfun.com/products/9964)!

[](https://cdn.sparkfun.com/assets/c/7/b/e/a/515b587fce395fec38000001.png)

Just a few of the sensors that are easily compatible with Arduino

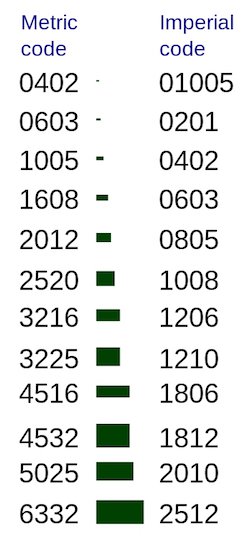
# Understanding Arduino UNO Hardware Design

# Components Overview

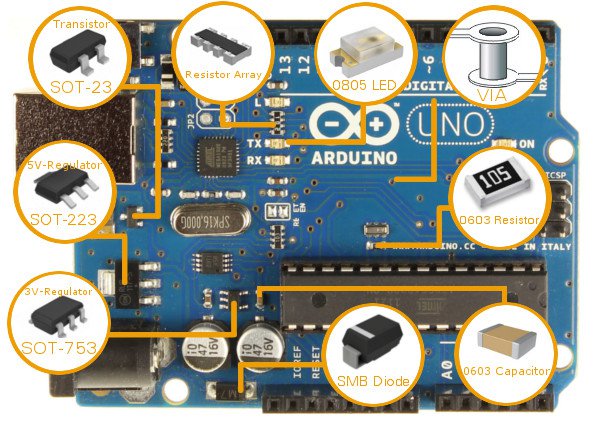
The PCB design of the Arduino UNO uses SMD (Surface Mount Device) components. I entered the SMD world years ago when I dug into Arduino PCB design while I was a part of a team redesigning a [DIY clone](https://upverter.com/atadiat/032374e685e8e488/SHAMDUINO/) for Arduino UNO.

Integrated circuits use standardized packages, and there are families for packages.

The dimensions of many SMD resistors, capacitors, and LEDs are indicated by package codes such as the following:



Most packages are generic and can be used for different parts with different functionality. The SOT-223 package, for example, can contain a transistor or a regulator.

​

Arduino UNO System Overview

Before we can understand the UNO's hardware, we must have a general overview of the system first.

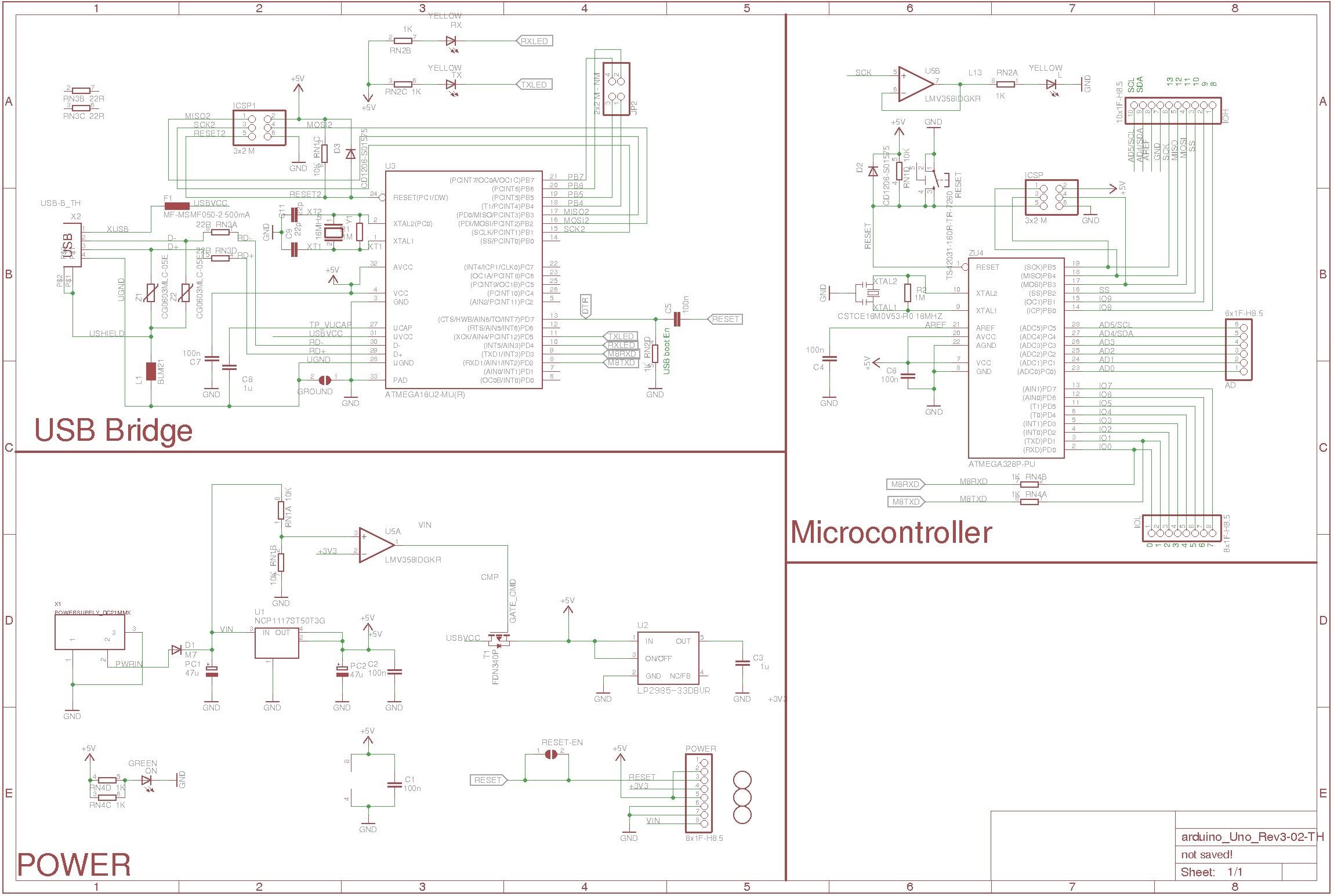
After your code is compiled using Arduino IDE, it should be uploaded to the main microcontroller of the Arduino UNO using a USB connection. Because the main microcontroller doesn’t have a USB transceiver, you need a bridge to convert signals between the serial interface (UART interface) of the microcontroller and the host USB signals.

The bridge in the latest revision is the ATmega16U2, which has a USB transceiver and also a serial interface (UART interface).

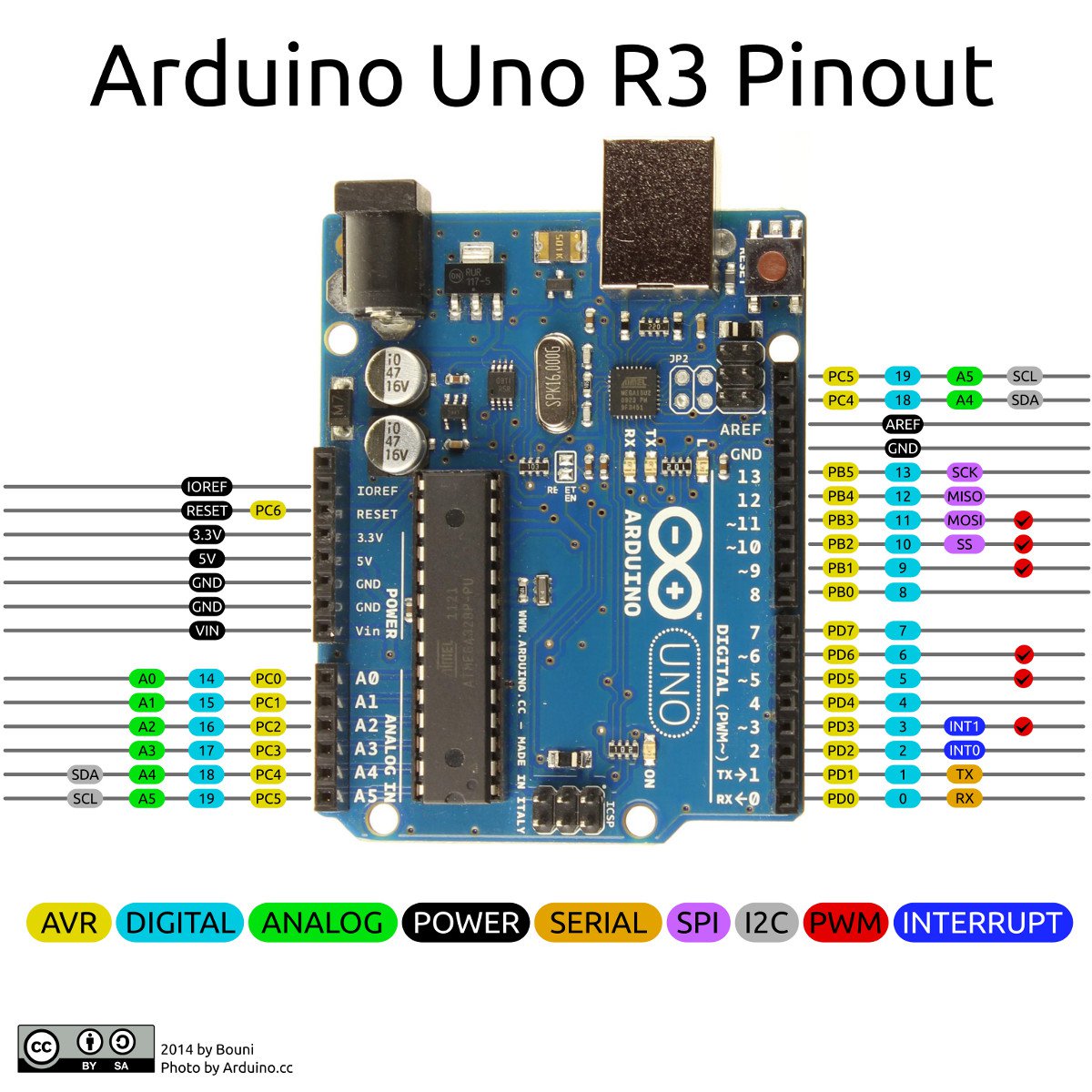
To power your Arduino board, you can use the USB as a power source. Another option is to use a DC jack. You may ask, “if I connect both a DC adapter and the USB, which will be the power source?” The answer will be discussed in the “Power Part” section from this article.

To reset your board, you should use a push button in the board. Another source of reset should be every time you open the serial monitor from Arduino IDE.

I redistributed the original Arduino UNO schematic to be more readable below. I advise you to download it and open the PCB and schematic using Eagle CAD while you are reading this article.

[](http://www.allaboutcircuits.com/uploads/articles/Arduino_schematic.png)

##### Image result for explain arduino uno r3



### The Microcontroller

It is important to understand that the Arduino board includes a microcontroller, and this microcontroller is what executes the instructions in your program. If you know this, you won't use the common nonsense phrase "Arduino is a microcontroller" ever again.

The ATmega328 microcontroller is the MCU used in Arduino UNO R3 as a main controller. ATmega328 is an MCU from the AVR family; it is an 8-bit device, which means that its data-bus architecture and internal registers are designed to handle 8 parallel data signals.

ATmega328 has three types of memory:

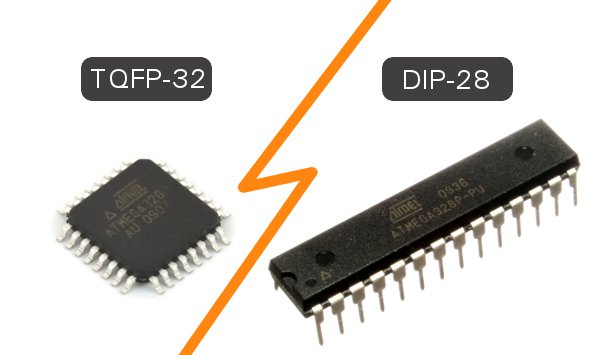
* **Flash memory:** 32KB nonvolatile memory. This is used for storing application, which explains why you don't need to upload your application every time you unplug arduino from its power source.
* **SRAM memory:** 2KB volatile memory. This is used for storing variables used by the application while it's running.
* **EEPROM memory:** 1KB nonvolatile memory. This can be used to store data that must be available even after the board is powered down and then powered up again.

Let us briefly go over some of this MCU's specs:

#### Packages:

This MCU is a DIP-28 package, which means that it has 28 pins in the dual in-line package. These pins include power and I/O pins. Most of the pins are multifunctional, which means that the same pin can be used in different modes based on how you configure it in the software. This reduces the necessary pin count, because the microcontroller does not require a separate pin for every function. It can also make your design more flexible, because one I/O connection can provide multiple types of functionality.

Other packages of ATmega328 are available like TQFP-32 SMD package (Surface Mount Device).



#### Power:

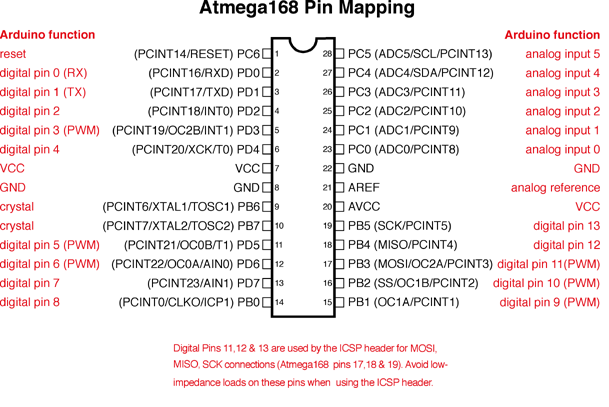
The MCU accepts supply voltages from 1.8 to 5.5 V. However, there are restrictions on the operating frequency; for example, if you want to use the maximum clock frequency (20 MHz), you need a supply voltage of at least 4.5 V.

#### Digital I/O:

This MCU has three ports: PORTC, PORTB, and PORTD. All pins of these ports can be used for general-purpose digital I/O or for the alternate functions indicated in the pinout below. For example, PORTC pin0 to pin5 can be ADC inputs instead of digital I/O.

There are also some pins that can be configured as PWM output. These pins are marked with “~” on the Arduino board.

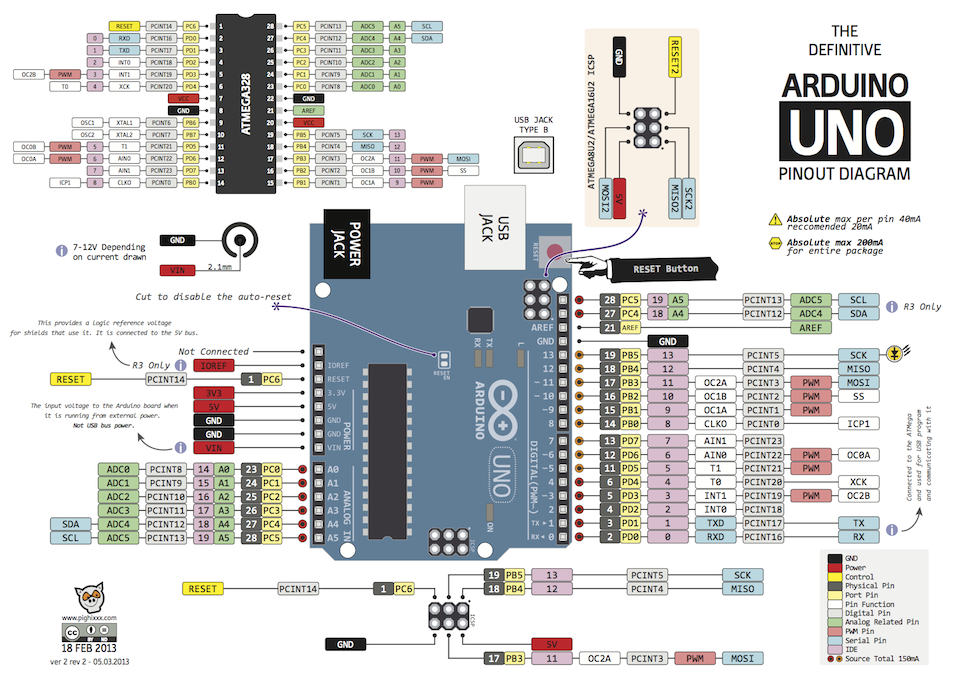
**Note**: The ATmega168 is almost identical to the ATmega328 and they are pin compatible. The difference is that the ATmega328 has more memory—32KB flash, 1KB EEPROM, and 2KB RAM compared to the ATmega168's 16KB flash, 512 bytes EEPROM, and 1KB RAM.

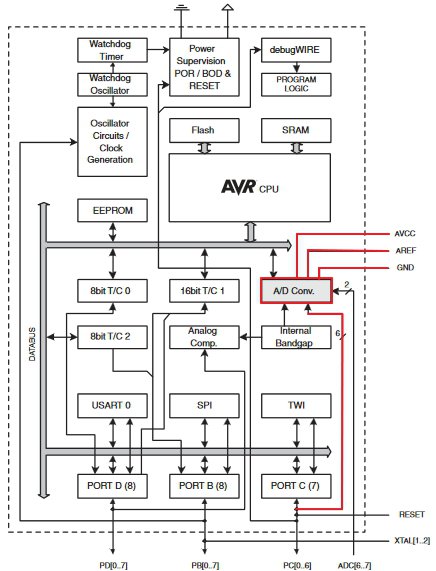


### Arduino UNO R3 Pinout

#### ADC Inputs:

This MCU has six channels—PORTC0 to PORTC5—with 10-bit resolution A/D converter. These pins are connected to the analog header on the Arduino board.



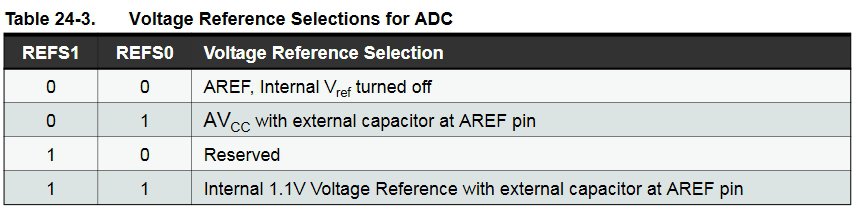


##### **ATmega328 block diagram.**

One common mistake is to think of analog input as dedicated input for A/D function only, as the header in the board states ”Analog”. The reality is that you can use them as digital I/O or A/D.

As shown in the diagram above (via the red traces), the pins related to the A/D unit are:

* AVCC: The power pin for the A/D unit.
* AREF: The input pin used optionally if you want to use an external voltage reference for ADC rather than the internal Vref. You can configure that using an internal register.



##### **Internal register settings for selecting the Vref source.**

#### UART Peripheral:

A UART (Universal Asynchronous Receiver/Transmitter) is a serial interface. The ATmega328 has only one UART module.

The pins (RX, TX) of the UART are connected to a USB-to-UART converter circuit and also connected to pin0 and pin1 in the digital header. You must avoid using the UART if you’re already using it to send/receive data over USB.

#### SPI Peripheral:

The SPI (Serial Peripheral Interface) is another serial interface. The ATmega328 has only one SPI module.

Besides using it as a serial interface, it can also be used to program the MCU using a standalone programmer. You can reach the SPI's pins from the header next to the MCU in the Arduino UNO board or from the digital header as below:  
11<->MOSI  
12<->MISO  
13<->SCK

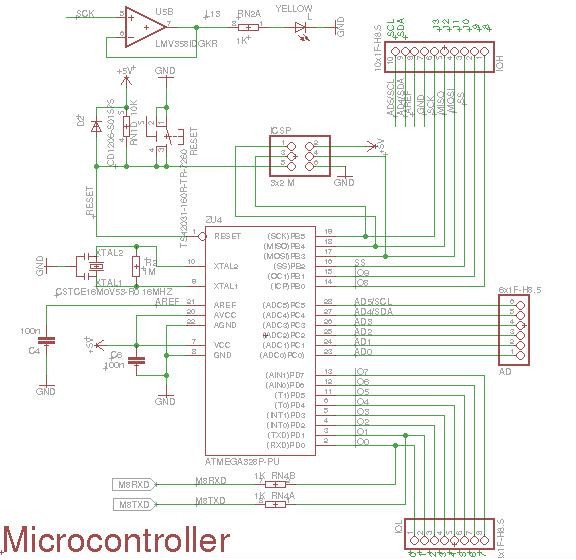
#### 

#### TWI:

The I2C or Two Wire Interface is an interface consisting of only two wires, serial data, and a serial clock: SDA, SCL.

You can reach these pins from the last two pins in the digital header or pin4 and pin5 in the analog header.

#### Other Functionality:Other functionality is included in the MCU, such as that offered by the timer/counter modules. You may not be aware of the functions that you don't use in your code. You can refer to the datasheet for more information.

[](http://www.allaboutcircuits.com/uploads/articles/Arduino_MCU_Part.jpg)**Arduino UNO R3 MCU part.**

 Returning to the electronic design, the microcontroller section has the following:

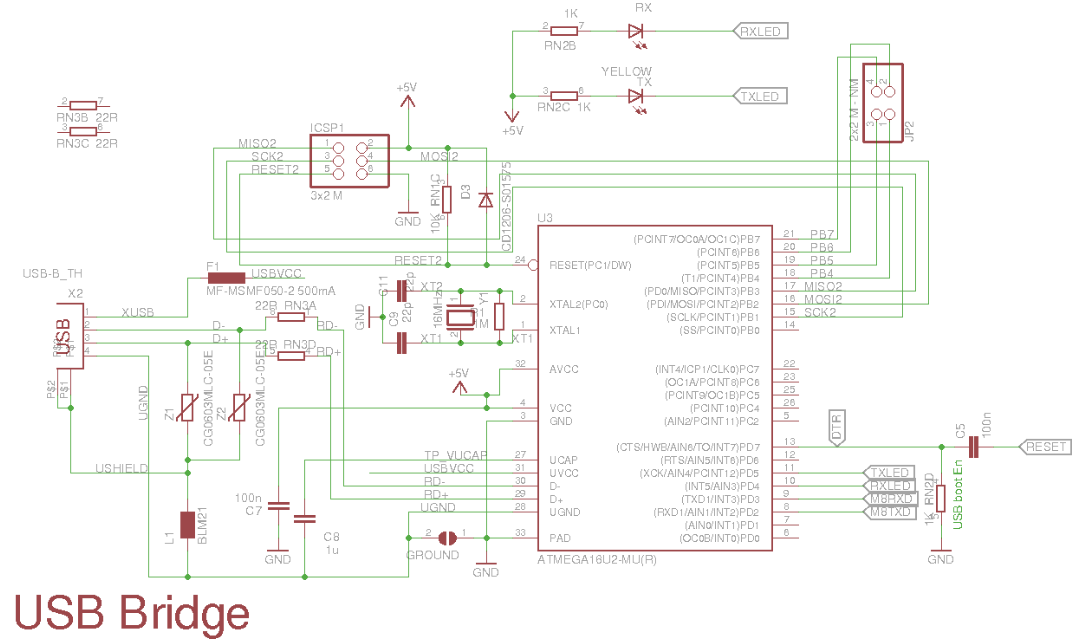
* **ATmega328-PU:** The MCU we just talked about.
* **IOL and IOH (Digital) Headers:** These headers are the digital header for pins 0 to 13 in addition to GND, AREF, SDA, and SCL. Note that RX and TX from the USB bridge are connected with pin0 and pin1.
* **AD Header:** The analog pins header.
* **16 MHz Ceramic Resonator (CSTCE16M0V53-R0):** Connected with XTAL2 and XTAL1 from the MCU.
* **Reset Pin:** This is pulled up with a 10K resistor to help prevent spurious resets in noisy environments; the pin has an internal pull-up resistor, but according to the AVR Hardware Design Considerations application note ([AVR042](http://www.atmel.com/Images/Atmel-2521-AVR-Hardware-Design-Considerations_ApplicationNote_AVR042.pdf)), “if the environment is noisy, it can be insufficient and reset may occur sporadically.” Reset occurs if the user presses the reset button or if a reset is issued from the USB bridge. You can also see the D2 diode. The role of this diode is described in the same app note: “If not using High Voltage Programming it is recommended to add an ESD protection diode from RESET to Vcc,  since this is not internally provided due to High Voltage Programming”.
* **C4 and C6 100nF Capacitors:** These are added to filter supply noise. The impedance of a capacitor decreases with frequency:

Xc = 1/2πfC

The capacitors give high-frequency noise signals a low-impedance path to ground. 100nF is the most common value.

* **PIN13:** This is connected to the SCK pin from the MCU and is also connected to an LED. The Arduino board uses a buffer (the LMV358) to drive the LED.
* **ICSP (In-Circuit Serial Programming) Header:** This is used to program the ATmega328 using an external programmer. It’s connected to the In-System Programming (ISP) interface (which uses the SPI pins). Usually, you don’t need to use this way of programming because bootloader handles the programming of the MCU from the UART interface which is connected using a bridge to the USB. This header is used when you need to flash the MCU, for example, with a bootloader for the first time in production.

### The USB-to-UART Bridge

[](http://www.allaboutcircuits.com/uploads/articles/Arduino_USBBridge_Part.png)

##### **Arduino USB bridge part. Click to enlarge.**

As we discussed in the “Arduino UNO System Overview” section, the role of the USB-to-UART bridge part is to convert the signals of USB interface to the UART interface, which the ATmega328 understands, using an ATmega16U2 with an internal USB transceiver. This is done using special firmware uploaded to the ATmega16U2.

From an electronic design perspective, this section is similar to microcontroller section. This MCU has an ICSP header, an external crystal with load capacitors (CL), and a Vcc filter capacitor.

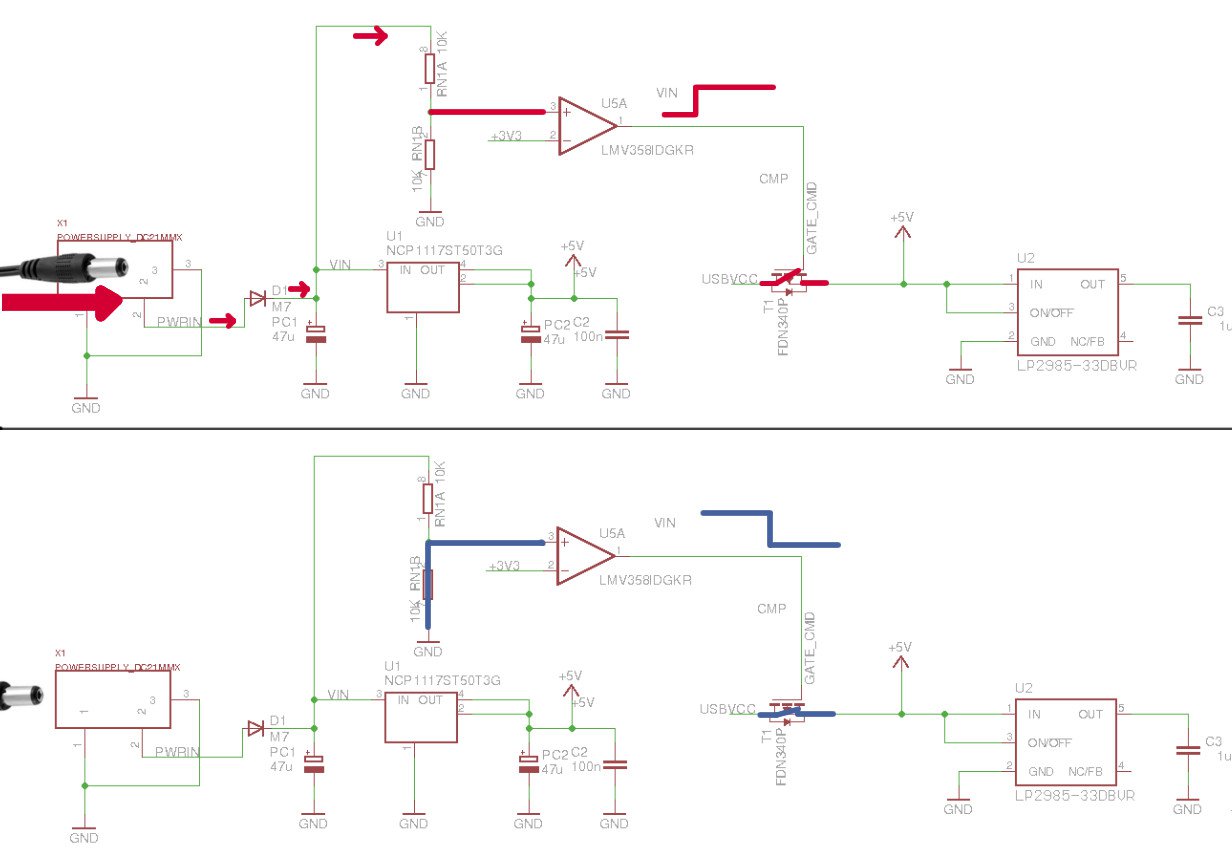
### The Power

For a power source, you have the option of using the USB or a DC jack. Now it’s time to answer the following question: “If I connect both a DC adapter and the USB, which will be the power source?”

The 5V regulator is the NCP1117ST50T3G and the Vin of this regulator is connected via DC jack input through the M7 diode, the SMD version of the famous [1N4007 diode](http://www.mouser.com/ds/2/149/1N4007-888322.pdf) (PDF). This diode provides reverse-polarity protection.

The output of the 5V regulator is connected to the rest of 5V net in the circuit and also to the input of the 3.3V regulator, LP2985-33DBVR. You can access 5V directly from  the power header 5V pin.

Another source of 5V is USBVCC which is connected to the drain of an FDN340P, a P-channel MOSFET, and the source is connected to the 5V net. The gate of the transistor is connected to the output of an LMV358 op-amp used as a comparator. The comparison is between 3V3 and Vin/2. When Vin/2 is larger, this will produce a high output from the comparator and the P-channel MOSFET is off. If there is no Vin applied, the V+ of the comparator is pulled down to GND and \*Vout is low, such that the transistor is on and the USBVCC is connected to 5V.

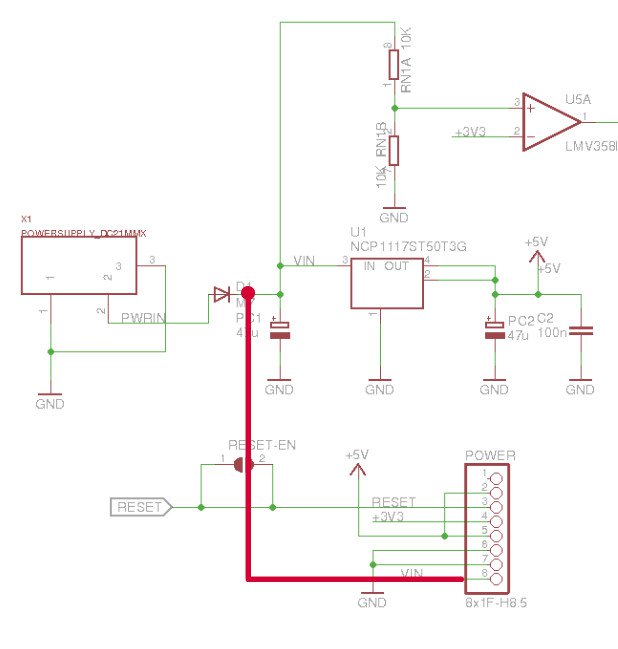
[](http://www.allaboutcircuits.com/uploads/articles/Arduino_Power_Switch.jpg)

##### **Power source switching mechanism. Click to enlarge.**

The LP2985-33DBVR is the 3V3 regulator. Both the 3V3 and 5V regulators are LDO (Low Dropout), which means that they can regulate voltage even if the input voltage is close to the output voltage. This is an improvement over older linear regulators, such as the [7805](https://www.fairchildsemi.com/datasheets/LM/LM7805.pdf).

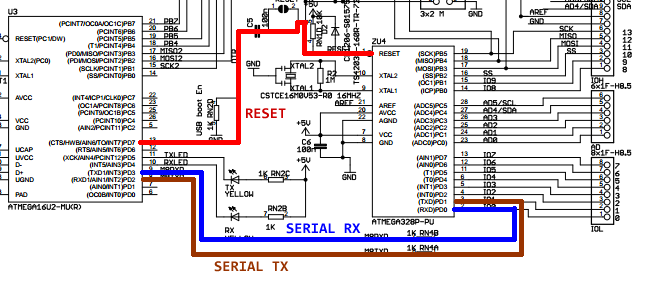
The last thing I'll talk about is the power protection that is provided in Arduino UNO.

As mentioned above, VIN from a DC jack is protected from reverse polarity by using a serial M7 diode in the input. Be aware that the VIN pin in the power header is not protected. This is because it is connected after the M7 diode. Personally, I don’t know why they decided to do that when they could connect it before the diode to provide the same protection.

[](http://www.allaboutcircuits.com/uploads/articles/Arduino_VIN.jpg)

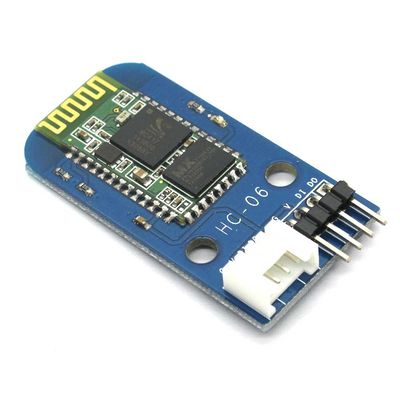
##### **VIN pin from power header. Click to enlarge.**

When you use USB as a power source, and to provide protection for your USB port, there is a PTC (positive temperature coefficient) fuse ([MF-MSMF050-2](http://www.mouser.com/ds/2/54/fmsmf-772347.pdf)) in series with the USBVCC. This provides protection from overcurrent, 500mA. When an overcurrent limit is reached, the PTC resistance increases a lot. Resistance decreases after the overcurrent is removed.

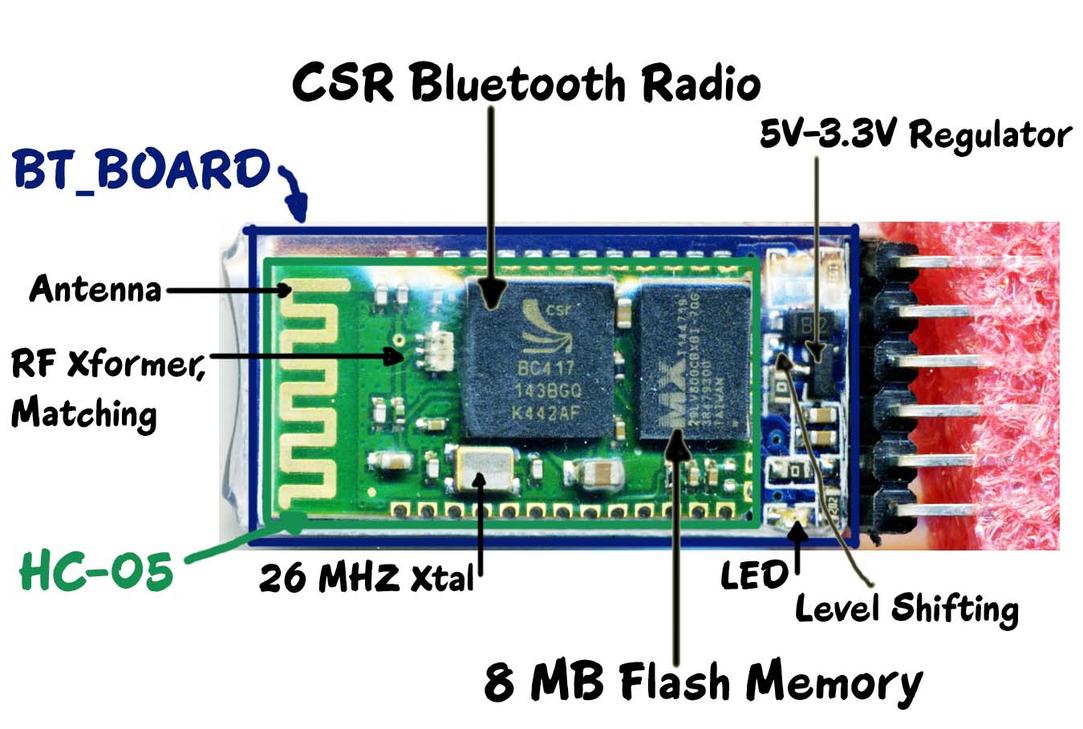


1. HC-06 BLUETOOTH MODULE

# BlueTooth-HC06-Modules-How-To

(arduino bluetooth tutorial)  
***Bluetooth***  is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs). Range is approximately 10 Meters (30 feet). 

These modules are based on the Cambridge Silicon Radio BC417 2.4 GHz BlueTooth Radio chip. This is a complex chip which uses an external 8 Mbit flash memory.   
  
These low-cost Bluetooth Sub-modules work well with Arduino and other Microcomputers. IMPORTANT Definitions:

* HC-06 is a more capable module that can be set to be either Master or Slave
* HC-06 is a Slave only device. (It looks physically just like the HC-06).(Note: Now HC-06 not cheaper)
* These small ( 3 cm long) modules run on 3.3V power with 3.3V signal levels, They have **no** pins and usually solder to a larger board. (See example below)
* The module has two modes of operation, Command Mode where we can send AT commands to it and Data Mode where it transmits and receives data to another bluetooth module.
* "Breakout" Boards that make these easy to use are available and recommended. These mount the sub-module like that shown on the right on a slightly larger board. NOTE: Sellers often label them "HC-06" or "HC-06", but they have some other model number on the reverse side. Most of these boards support operation at 5V power and interface to 5V Arduino signal levels with some technique of level shifting. A typical "breakout" board is shown next page.
* **HC-06 :**

## The Keyes / YourDuino BT Board Sections of the BT Board:



* The Green HC-06 sub-module is soldered on top of the Blue BT Board
* The HC-06 module includes the Radio and Memory chips, 26 MHz crystal, antenna and RF matching network.
* The right section of the BT Board has connection pins for power and signals as well as a 5V to 3.3V Regulator, LED, and level shifting.

### HC-06 PinOut (Right) :

* KEY: If brought HIGH **before** power is applied, forces AT Command Setup Mode. LED blinks slowly (2 seconds)
* VCC: +5 Power
* GND: System / Arduino Ground
* TXD: Transmit Serial Data from HC-06 to Arduino Serial Receive. NOTE: 3.3V HIGH level: OK for Arduino
* RXD: Receive Serial Data from Arduino Serial Transmit
* STATE: Tells if connected or not

### BlueToothCommandUtility:Arduino Sketch to send and receive Command Mode AT commands to HC-06 to change configuration.

### COMMAND and DATA TRANSFER MODES:

The module has two modes of operation, Command Mode where we can send AT commands to it and Data Mode where it transmits and receives data to another bluetooth module.  
  
The default mode is DATA Mode, and this is the default configuration, that may work fine for many applications:

* Baud Rate: 9600 bps, Data : 8 bits, Stop Bits: 1 bit, Parity : None, Handshake: None
* Passkey: 1234
* Device Name: HC-06

In some cases you may want to change some of the configuration setup values. There are two ways to get into Command Mode:

1. Connect the KEY pin high **before** applying power to the module. This will put the module into command mode at 38400 baud. This is commonly used, and needed if you don't know the baud rate the module is set to. You can use the BlueToothCommandUtility  for this.
2. Apply power to the module **then** pull the KEY pin high. This will enter command mode at the currently configured baud rate. This is useful if you want to send AT commands from a microcontroller as the KEY pin can be controlled from one of the microcontroller pins. BUT you need to know the currently configured Baud Rate.

Commands are sent to the module in UPPERCASE and are terminated with a CR/LF pair.

### Command Mode Commands:

The format of commands is:

* Always starts with "AT"
* Then "+" followed by <ParameterName>
* Then either:
  + ? (returns current value of parameter)
  + = (New Value of parameter)

A few examples:

* AT (AT Test command. Should respond with OK)
* AT+VERSION? (show the firmware version)
* AT+UART=9600,0,0 (Set baud rate to 9600, 1 stop bit, no parity)

### Slave Mode:

The HC-06 bluetooth module can also act as a slave. There are fewer commands to set this up:

* AT+ORGL Reset to defaults
* AT+RMAAD Clear any paired devices
* AT+ROLE=0 Set mode to SLAVE
* AT+ADDR Display SLAVE address

**SERIAL BEGIN PROGRAM OF BLUETOOTH**

#include <SoftwareSerial.h>

SoftwareSerial mySerial(4, 2); // RX, TX

String command = ""; // Stores response of the HC-06 Bluetooth device

void setup() {

// Open serial communications:

Serial.begin(9600);

Serial.println("Type AT commands!");

// The HC-06 defaults to 9600 according to the datasheet.

mySerial.begin(9600);

}

void loop() {

// Read device output if available.

if (mySerial.available()) {

while(mySerial.available()) { // While there is more to be read, keep reading.

command += (char)mySerial.read();

}

Serial.println(command);

command = ""; // No repeats

}

// Read user input if available.

if (Serial.available()){

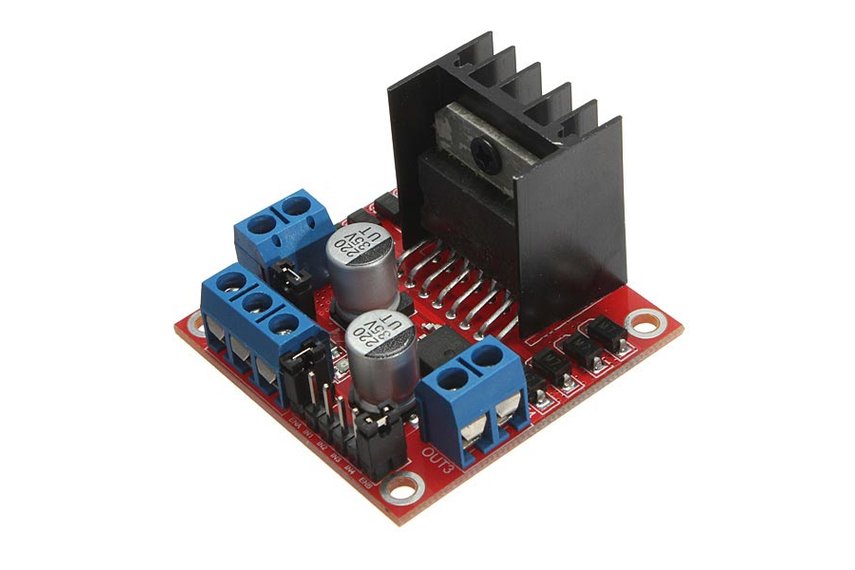
delay(10); // The delay is necessary to get this working!

mySerial.write(Serial.read());

}

}

1. L298 dual h-bridge motor driveR



# Motor Driver 2A Dual L298 H-Bridge

**Description:** This dual bidirectional motor driver, it is based on the very popular [L298](http://www.sparkfun.com/commerce/product_info.php?products_id=9479) Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently control two motors of up to 2A each in both directions.

It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc.

The circuit incorporates 4 direction LEDs (2 per motor), a heat sink, screw-terminals, as well as eight Schottky EMF-protection diodes. Two high-power current sense resistors are also incorporated which allow monitoring of the current drawn on each motor through your microcontroller.

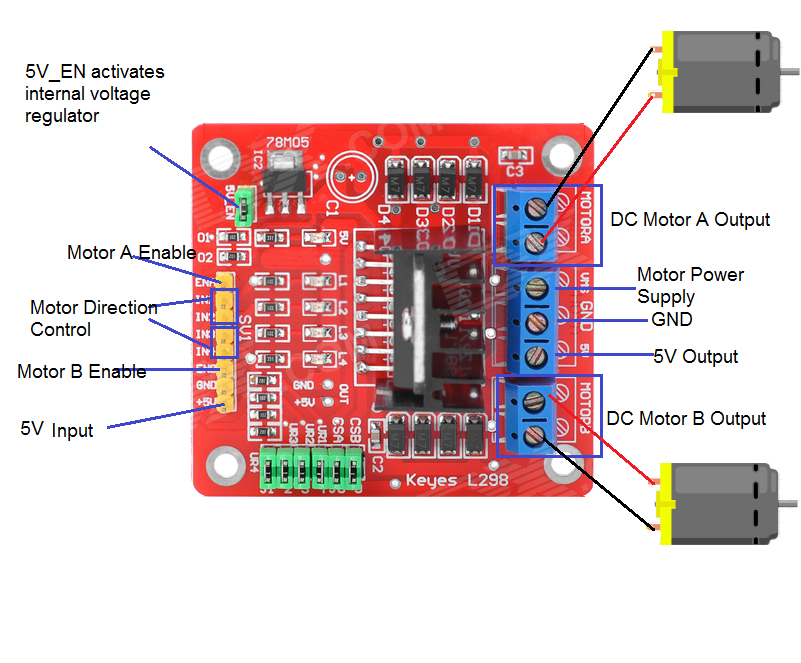
An on-board user-accessible 5V regulator is also incorporated which can also be used to supply any additional circuits requiring a regulated 5V DC supply of up to about 1A.

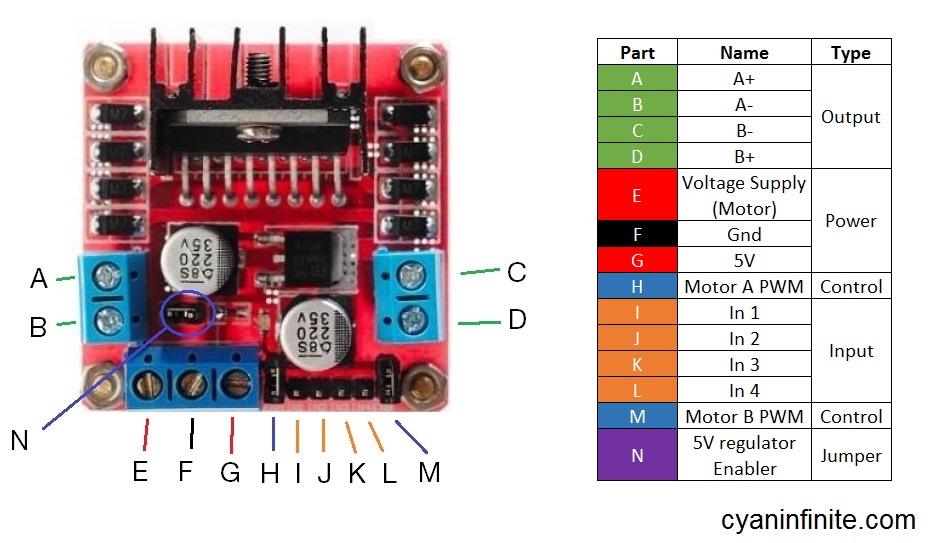
The circuit also offers a bridged mode of operation allowing bidirectional control of a single motor of up to about 4A.

**Features:**

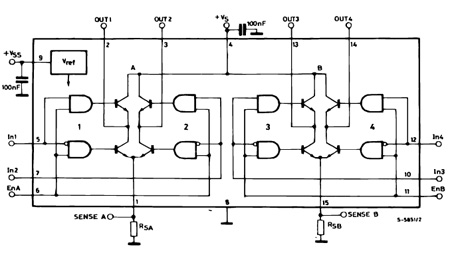
* Motor supply: 6 to 35 VDC
* Control Logic: Standard TTL Logic Level
* Output Power: Up to 2 A each
* Current Sense Outputs
* Onboard Power Resistors Provided for Current Limit
* Enable and Direction Control Pins
* External Diode Bridge Provided for Output
* Heatsink for IC
* Power-On LED indicator
* 4 Direction LED indicators

**PIN DESCRIPTION:-**





## CIRCUIT DIAGRAM



**Usage:**

H-Bridge's are typically used in controlling motors speed and direction, but can be used for other projects such as driving the brightness of certain lighting projects such as high powered LED arrays.

**How it works:**

An H-Bridge is a circuit that can drive a current in either polarity and be controlled by \*Pulse Width Modulation (PWM).

\* Pulse Width Modulation is a means in controlling the duration of an electronic pulse. In motors try to imagine the brush as a water wheel and electrons as a the flowing droplets of water. The voltage would be the water flowing over the wheel at a constant rate, the more water flowing the higher the voltage. Motors are rated at certain voltages and can be damaged if the voltage is applied to heavily or if it is dropped quickly to slow the motor down. Thus PWM. Take the water wheel analogy and think of the water hitting it in pulses but at a constant flow. The longer the pulses the faster the wheel will turn, the shorter the pulses, the slower the water wheel will turn. Motors will last much longer and be more reliable if controlled through PWM.

**Pins:**

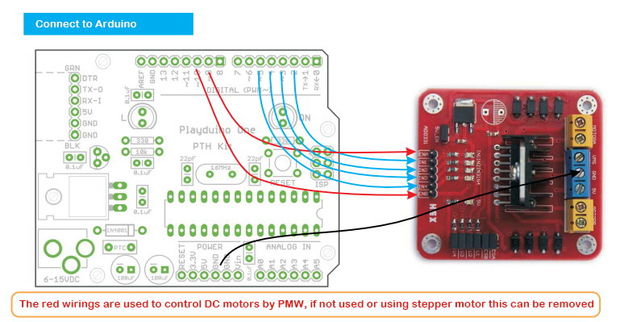
* Out 1: Motor A lead out
* Out 2: Motor A lead out
* Out 3: Motor B lead out
* Out 4: Mo (*Can actually be from 5v-35v, just marked as 12v*)
* GND: Ground
* 5v: 5v input (*unnecessary if your power source is 7v-35v, if the power source is 7v-35v then it can act as a 5v out*)
* EnA: Enables PWM signal for Motor A (Please see the "Arduino Sketch Considerations" section)
* In1: Enable Motor A
* In2: Enable Motor A
* In3: Enable Motor B
* In4: Enable Motor B
* EnB: Enables PWM signal for Motor B (Please see the "Arduino Sketch Considerations" section)

**Specifications:**

* Double H bridge Drive Chip:*L298N*
* Logical voltage: *5V Drive voltage: 5V-35V*
* Logical current: *0-36mA Drive current: 2A (MAX single bridge)*
* Max power: *25W*
* Dimensions: *43 x 43 x 26mm*
* Weight: *26g*

*\*Built-in 5v power supply, when the driving voltage is 7v-35v*

## Wiring to an Arduino:



There are several different models of these L298N Dual H-Bridge Motor Controllers. The generic wiring schematic above should do the trick for most.

**Two things to mention**

* Make sure you have all of your grounds tied together; Arduino, Power source, and the Motor controller.
* The PWM Pins are unnecessary if you do not want to control PWM features.

**CODE:-**

//L293D

//Motor A

const int motorPin1 = 9; // Pin 14 of L293

const int motorPin2 = 10; // Pin 10 of L293

//Motor B

const int motorPin3 = 6; // Pin 7 of L293

const int motorPin4 = 5; // Pin 2 of L293

void setup()

{

//Set pins as outputs

pinMode(motorPin1, OUTPUT);

pinMode(motorPin2, OUTPUT);

pinMode(motorPin3, OUTPUT);

pinMode(motorPin4, OUTPUT);

}

//This will run only one time.

void loop(){

//Motor Control - Motor A: motorPin1,motorpin2 & Motor B: motorpin3,motorpin4

//This code will turn Motor A clockwise for 2 sec.

analogWrite(motorPin1, 180);

analogWrite(motorPin2, 0);

analogWrite(motorPin3, 180);

analogWrite(motorPin4, 0);

delay(5000);

//This code will turn Motor A counter-clockwise for 2 sec.

analogWrite(motorPin1, 0);

analogWrite(motorPin2, 180);

analogWrite(motorPin3, 0);

analogWrite(motorPin4, 180);

delay(5000);

//This code will turn Motor B clockwise for 2 sec.

analogWrite(motorPin1, 0);

analogWrite(motorPin2, 180);

analogWrite(motorPin3, 180);

analogWrite(motorPin4, 0);

delay(1000);

//This code will turn Motor B counter-clockwise for 2 sec.

analogWrite(motorPin1, 180);

analogWrite(motorPin2, 0);

analogWrite(motorPin3, 0);

analogWrite(motorPin4, 180);

delay(1000);

//And this code will stop motors

analogWrite(motorPin1, 0);

analogWrite(motorPin2, 0);

analogWrite(motorPin3, 0);

analogWrite(motorPin4, 0);

}

1. ULTRASONIC RANGING MODULE HC-SR04



**Description**

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. From 2cm to 400 cm or 1” to 13 feet. It operation is not affected by sunlight or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

**Features**

* Power Supply :+5V DC
* Quiescent Current : <2mA
* Working Current: 15mA
* Effectual Angle: <15°
* Ranging Distance : 2cm – 400 cm/1″ – 13ft
* Resolution : 0.3 cm
* Measuring Angle: 30 degree
* Trigger Input Pulse width: 10uS
* Dimension: 45mm x 20mm x 15mm

**Sensor**



**Pins**

* VCC: +5VDC
* Trig : Trigger (INPUT)
* Echo: Echo (OUTPUT)
* GND: GND

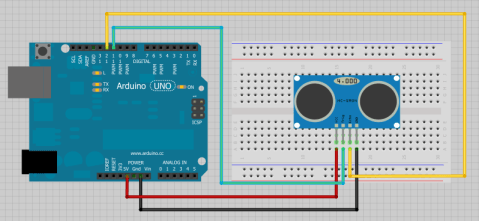
## Arduino with HC – SR04 Sensor

This sensor is really cool and popular among the Arduino Tinkerers. So I’ve decided to post a project example using this sensor. In this project the ultrasonic sensor read and write the distance in the serial monitor. It’s really simple.

My goal is to help you understand how this sensor works and then you can use this example in your own projects.

**Note:** There’s an Arduino library called [NewPing](http://playground.arduino.cc/Code/NewPing) that can make your life easier when using this sensor.

## Schematics



## Source code

TESTING :-

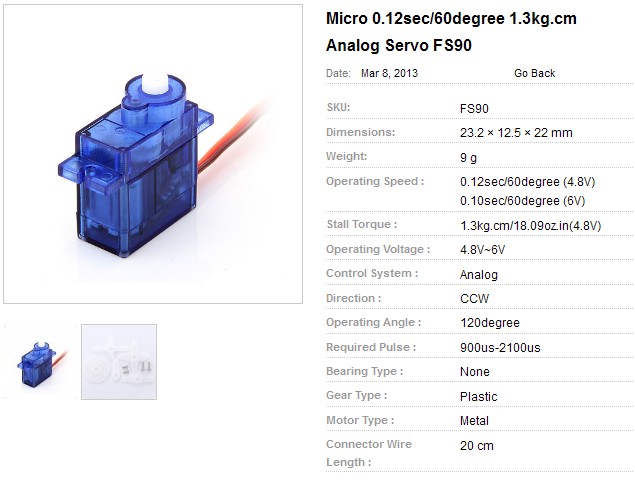
/\* Complete Guide for Ultrasonic Sensor HC-SR04  
 \*  
    Ultrasonic sensor Pins:  
        VCC: +5VDC  
        Trig : Trigger (INPUT) - Pin11  
        Echo: Echo (OUTPUT) - Pin 12  
        GND: GND  
 \*/  
int trigPin = 11;    //Trig - green Jumper  
int echoPin = 12;    //Echo - yellow Jumper  
long duration, cm, inches;  
   
void setup() {  
  //Serial Port begin  
  Serial.begin (9600);  
  //Define inputs and outputs  
  pinMode(trigPin, OUTPUT);  
  pinMode(echoPin, INPUT);  
}  
   
void loop()  
{   
   
  // The sensor is triggered by a HIGH pulse of 10 or more microseconds.  
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:  
  digitalWrite(trigPin, LOW);  
  delayMicroseconds(5);  
  digitalWrite(trigPin, HIGH);  
  delayMicroseconds(10);  
  digitalWrite(trigPin, LOW);  
   
  // Read the signal from the sensor: a HIGH pulse whose  
  // duration is the time (in microseconds) from the sending  
  // of the ping to the reception of its echo off of an object.  
  pinMode(echoPin, INPUT);  
  duration = pulseIn(echoPin, HIGH);  
   
  // convert the time into a distance  
  cm = (duration/2) / 29.1;  
  inches = (duration/2) / 74;   
    
  Serial.print(inches);  
  Serial.print("in, ");  
  Serial.print(cm);  
  Serial.print("cm");  
  Serial.println();  
    
  delay(250);  
}

Source code with NewPing

#include <NewPing.h>  
#define TRIGGER\_PIN 11  
#define ECHO\_PIN 12  
#define MAX\_DISTANCE 200  
NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE); // NewPing setup of pins and maximum distance.  
   
void setup() {  
   Serial.begin(9600);  
}  
void loop() {  
   delay(50);  
   unsigned int uS = sonar.ping\_cm();  
   Serial.print(uS);  
   Serial.println(“cm”);  
}

1. SERVO MOTOR





**What ARE Servo Motors ?**

A servo motor is a linear or rotary actuator that provides fast precision position control for closed-loop position control applications. Unlike large industrial motors, a servo motor is not used for continuous energy conversion.

Servo motors have a high speed response due to low inertia and are designed with small diameter and long rotor length. Then how do servo motors work?

Servo motors work on servo mechanism that uses position feedback to control the speed and final position of the motor. Internally, a servo motor combines a motor, feedback circuit, controller and other electronic circuit.

It uses encoder or speed sensor to provide speed feedback and position. This feedback signal is compared with input command position (desired position of the motor corresponding to a load), and produces the error signal (if there exist a difference between them).

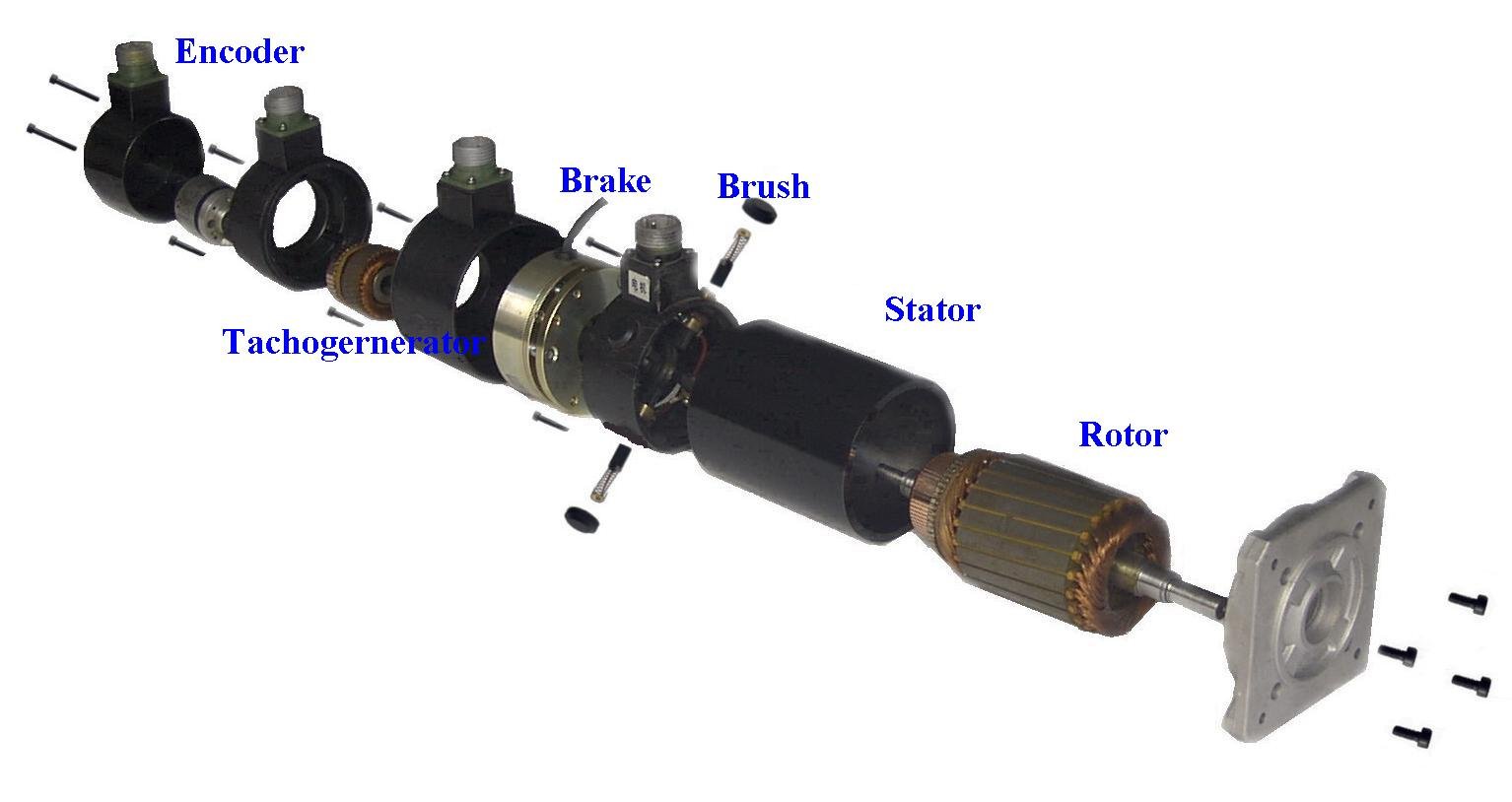
The error signal available at the output of error detector is not enough to drive the motor. So the error detector followed by a servo amplifier raises the voltage and power level of the error signal and then turns the shaft of the motor to desired position.

### Types of Servo Motors

Basically, servo motors are classified into AC and DC servo motors depending upon the nature of supply used for its operation. Brushed permanent magnet DC servo motors are used for simple applications owing to their cost, efficiency and simplicity. These are best suited for smaller applications. With the advancement of microprocessor and power transistor, AC servo motors are used more often due to their high accuracy control.

DC Servo Motors

A DC servo motor consists of a small DC motor, feedback potentiometer, gearbox, motor drive electronic circuit and electronic feedback control loop. It is more or less similar to the normal DC motor. The stator of the motor consists of a cylindrical frame and the magnet is attached to the inside of the frame.



[**DC Servo Motor**](http://img.alibaba.com/img/pb/844/623/242/1271406736072_hz_fileserver2_207572.jpg)

The rotor consists of brush and shaft. A commutator and a rotor metal supporting frame are attached to the outside of the shaft and the armature winding is coiled in the rotor metal supporting frame.

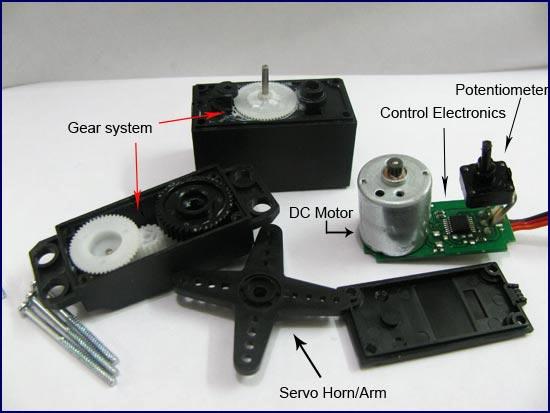
A brush is built with an armature coil that supplies the current to the commutator. At the back of the shaft, a detector is built into the rotor in order to detect the rotation speed.

With this construction, it is simple to design a controller using simple circuitry because the torque is proportional to the amount of current flow through the armature.

And also the instantaneous polarity of the control voltage decides the direction of torque developed by the motor. Types of DC servo motors include series motors, shunt control motor, split series motor, and permanent magnet shunt motor.

#### Working Principle of DC Servo Motor

A DC servo motor is an assembly of four major components, namely a DC motor, a position sensing device, a gear assembly, and a control circuit. The below figure shows the parts that consisting in RC servo motors in which small DC motor is employed for driving the loads at precise speed and position.



[**Internal diagram**](http://www.engineersgarage.com/sites/default/files/imagecache/Original/wysiwyg_imageupload/1/Servo%20Components.jpg)

A DC reference voltage is set to the value corresponding to the desired output. This voltage can be applied by using another potentiometer, control pulse width to voltage converter, or through timers depending on the control circuitry.

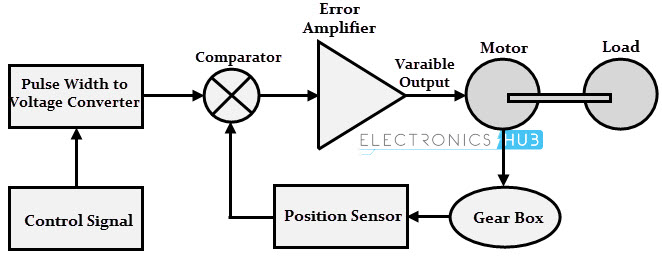
The dial on the potentiometer produces a corresponding voltage which is then applied as one of the inputs to error amplifier.

In some circuits, a control pulse is used to produce DC reference voltage corresponding to desired position or speed of the motor and it is applied to a pulse width to voltage converter.

In this converter, the capacitor starts charging at a constant rate when the pulse high. Then the charge on the capacitor is fed to the buffer amplifier when the pulse is low and this charge is further applied to the error amplifier.

So the length of the pulse decides the voltage applied at the error amplifier as a desired voltage to produce the desired speed or position.

In digital control, microprocessor or microcontroller are used for generating the PWM pluses in terms of duty cycles to produce more accurate control signals.

[](http://www.electronicshub.org/wp-content/uploads/2015/10/Working-Principle-of-DC-Servo-Motor.jpg)

The feedback signal corresponding to the present position of the load is obtained by using a position sensor. This sensor is normally a potentiometer that produces the voltage corresponding to the absolute angle of the motor shaft through gear mechanism. Then the feedback voltage value is applied at the input of error amplifier (comparator).

The error amplifier is a negative feedback amplifier and it reduces the difference between its inputs. It compares the voltage related to current position of the motor (obtained by potentiometer) with desired voltage related to desired position of the motor (obtained by pulse width to voltage converter), and produces the error either a positive or negative voltage.

This error voltage is applied to the armature of the motor. If the error is more, the more output is applied to the motor armature.

As long as error exists, the amplifier amplifies the error voltage and correspondingly powers the armature. The motor rotates till the error becomes zero. If the error is negative, the armature voltage reverses and hence the armature rotates in the opposite direction.

**ARDUINO SERVO MOTOR**

****

Servo motors are great devices that can turn to a specified position.

Usually, they have a servo arm that can turn 180 degrees. Using the Arduino, we can tell a servo to go to a specified position and it will go there. As simple as that!

Servo motors were first used in the Remote Control (RC) world, usually to control the steering of RC cars or the flaps on a RC plane. With time, they found their uses in robotics, automation, and of course, the Arduino world.

Here we will see how to connect a servo motor and then how to turn it to different positions.

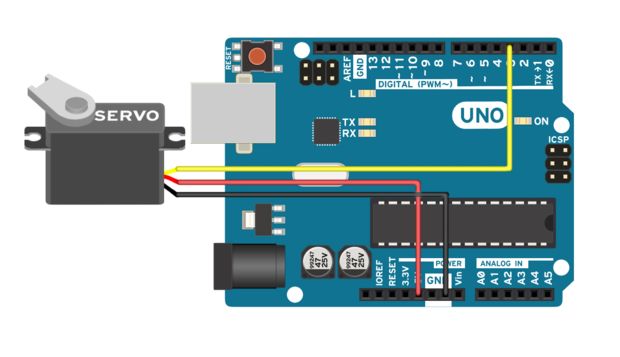
The first motor I ever connected to an Arduino, seven years ago, was a Servo motor. Nostalgic moment over, back to work!

We will need the following things:

1. An Arduino board connected to a computer via USB
2. A servo motor
3. Jumper wires

There are few big names in the servo motor world. Hitec and Futaba are the leading RC servo manufacturers. Good places to buy them are [Servocity](https://www.servocity.com/), [Sparkfun](https://www.sparkfun.com/categories/245), and [Hobbyking](http://www.hobbyking.com/).

## Step 1: How to Connect Them



A servo motor has everything built in: a motor, a feedback circuit, and most important, a motor driver. It just needs one power line, one ground, and one control pin.

Following are the steps to connect a servo motor to the Arduino:

1. The servo motor has a female connector with three pins. The darkest or even black one is usually the ground. Connect this to the Arduino GND.
2. Connect the power cable that in all standards should be red to 5V on the Arduino.
3. Connect the remaining line on the servo connector to a digital pin on the Arduino.

Check the image for a view of the servo connected to the Arduino.

## Step 2: Code

The following code will turn a servo motor to 0 degrees, wait 1 second, then turn it to 90, wait one more second, turn it to 180, and then go back.

// Include the Servo library   
#include <Servo.h>

// Declare the Servo pin

int servoPin = 3;

// Create a servo object

Servo Servo1;

void setup() {

// We need to attach the servo to the used pin number

Servo1.attach(servoPin);

}

void loop(){

// Make servo go to 0 degrees

Servo1.write(0);

delay(1000);

// Make servo go to 90 degrees

Servo1.write(90);

delay(1000);

// Make servo go to 180 degrees

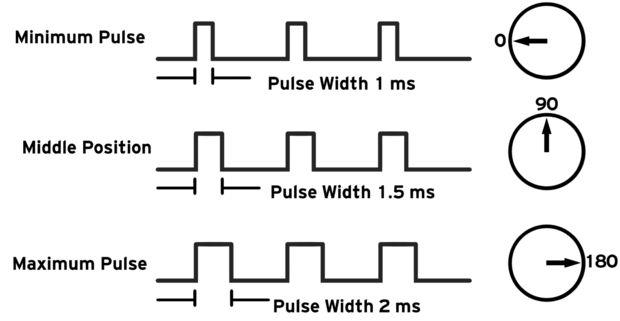
Servo1.write(180);

delay(1000);

}

If the servo motor is connected on another digital pin, simply change the value of servoPin to the value of the digital pin that has been used.

## Step 3: How It Works



Servos are clever devices. Using just one input pin, they receive the position from the Arduino and they go there. Internally, they have a motor driver and a feedback circuit that makes sure that the servo arm reaches the desired position. But what kind of signal do they receive on the input pin?

It is a square wave similar to PWM. Each cycle in the signal lasts for 20 milliseconds and for most of the time, the value is LOW. At the beginning of each cycle, the signal is HIGH for a time between 1 and 2 milliseconds. At 1 millisecond it represents 0 degrees and at 2 milliseconds it represents 180 degrees. In between, it represents the value from 0–180. This is a very good and reliable method. The graphic makes it a little easier to understand.

Remember that using the Servo library automatically disables PWM functionality on PWM pins 9 and 10 on the Arduino UNO and similar boards.

### Code breakdown

The code simply declares the servo object and then initializes the servo by using the servo.attach() function. We shouldn't forget to include the servo library. In the loop(), we set the servo to 0 degrees, wait, then set it to 90, and later to 180 degrees.

## Step 4: More Things About Servos

Controlling servos is easy, and here are a few more tricks we can use:

### Controlling the exact pulse time

Arduino has a built-in function servo.write(degrees) that simplifies the control of servos. However, not all servos respect the same timings for all positions. Usually, 1 millisecond means 0 degrees, 1.5 milliseconds mean 90 degrees, and, of course, 2 milliseconds mean 180 degrees. Some servos have smaller or larger ranges.

For better control, we can use the servo.writeMicroseconds(us) function, which takes the exact number of microseconds as a parameter. Remember, 1 millisecond equals 1,000 microseconds.

### More servos

In order to use more than one servo, we need to declare multiple servo objects, attach different pins to each one, and address each servo individually. First, we need to declare the servo objects—as many as we need:

// Create servo objects

Servo Servo1, Servo2, Servo3;

Then we need to attach each object to one servo motor. Remember, every servo motor uses an individual pin:

Servo1.attach(servoPin1);

Servo2.attach(servoPin2);

Servo3.attach(servoPin3);

In the end, we just have to address each servo object individually:

Servo1.write(0); // Set Servo 1 to 0 degrees

Servo2.write(90); // Set Servo 2 to 90 degrees

Connection-wise, the grounds from the servos go to GND on the Arduino, the servo power to 5V or VIN (depending on the power input), and in the end, each signal line has to be connected to a different digital pin. Contrary to popular belief, servos don't need to be controlled by PWM pins—any digital pin will work.

### Continuous rotation servos

There is a special breed of servos labelled as **continuous rotation servos**. While a normal servo goes to a specific position depending on the input signal, a continuous rotation servo either rotates clockwise or counter-clockwise at a speed proportional to the signal. For example, the Servo1.write(0) function will make the servomotor spin counter-clockwise at full speed. The Servo1.write(90) function will stop the motor and Servo1.write(180) will turn the motor clockwise at full speed. There are multiple uses for such servos; however, they are really slow. If you are building a microwave and need a motor to turn the food, this is your choice. But be careful, microwaves are dangerous!

**SOME DETAIL OF SERVO MOTOR**



Item Code: 77011635

TowerPro SG-90

• Stall Torque: 1.6kg/cm @ 6.0V

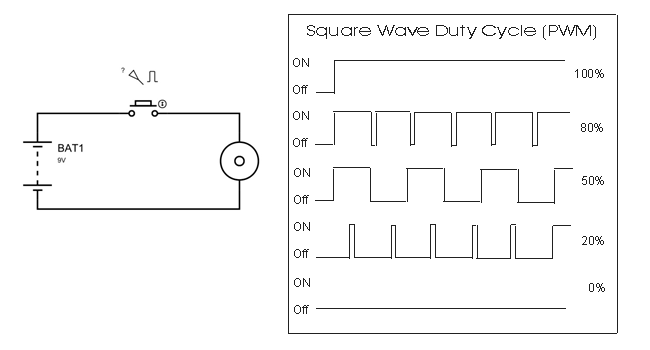
• Speed: 0.12 seconds/60deg. @ 4.8V

• Dimensions: 21x12x22 mm

• Temperature Range: 0c - 55c

• Operating Voltage: 3.0V - 6.0V

• Weight: 9 grams Teflon Bushing coreless motor

1. PWM MOTOR

# Duty cycle

A **duty cycle** is the fraction of one [period](https://en.wikipedia.org/wiki/Frequency) in which a signal or system is active. Duty cycle is commonly expressed as a percentage or a ratio. A period is the time it takes for a signal to complete an on-and-off [cycle](https://en.wikipedia.org/wiki/Turn_(geometry)). As a formula, a duty cycle (%) may be expressed as:

D=(PW/T)\*100%

Equally, a duty cycle (ratio) may be expressed as:

D=(PW/T)\*100

where {\displaystyle D}is the duty cycle, {\displaystyle PW}is the pulse width (pulse active time), and {\displaystyle T}is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.

Duty cycles can be used to describe the percent time of an active signal in an electrical device such as the power switch in a [switching power supply](https://en.wikipedia.org/wiki/Switched-mode_power_supply) or the firing of [action potentials](https://en.wikipedia.org/wiki/Action_potentials) by a living system such as a [neuron](https://en.wikipedia.org/wiki/Neuron).

The **duty factor** for periodic signal expresses the same notion, but is usually scaled to a maximum of one rather than 100%.

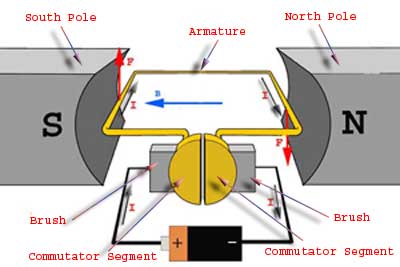
In electronics, duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform. It is generally used to represent time duration of a pulse when it is high (1). In digital electronics, signals are used in rectangular waveform which are represented by logic 1 and logic 0. Logic 1 stands for presence of an electric pulse and 0 for absence of an electric pulse. For example, a signal (10101010) has 50% duty cycle, because the pulse remains high for 1/2 of the period or low for 1/2 of the period. Similarly, for pulse (10001000) the duty cycle will be 25% because the pulse remains high only for 1/4 of the period and remains low for 3/4 of the period. Electrical motors typically use less than a 100% duty cycle. For example, if a [motor](https://en.wikipedia.org/wiki/Electric_motor) runs for one out of 100 seconds, or 1/100 of the time, then, its duty cycle is 1/100, or 1 percent.

[Pulse-width modulation](https://en.wikipedia.org/wiki/Pulse-width_modulation) (PWM) is used in a variety of electronic situations, such as power delivery and voltage regulation.

Working or Operating Principle of DC Motor

A [DC motor](https://www.electrical4u.com/dc-motor-or-direct-current-motor/) in simple words is a device that converts electrical energy (direct current system) into mechanical energy. It is of vital importance for the industry today, and is equally important for engineers to look into the **working principle of DC motor** in details that has been discussed in this article.

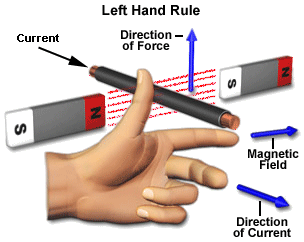
In order to understand the **operating principle of DC motor** we need to first look into its constructional feature.



The very basic [construction of a DC motor](https://www.electrical4u.com/construction-of-dc-motor-yoke-poles-armature-field-winding-commutator-brushes-of-dc-motor/) contains a [current](https://www.electrical4u.com/electric-current-and-theory-of-electricity/) carrying armature which is connected to the supply end through commutator segments and brushes.

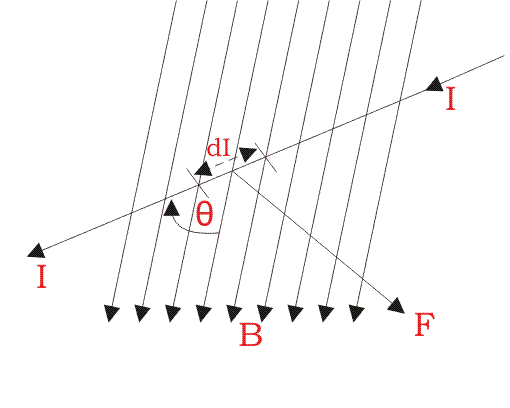
The armature is placed in between north south poles of a permanent or an electromagnet as shown in the diagram above.

As soon as we supply direct current in the armature, a mechanical force acts on it due to electromagnetic effect of the magnet. Now to go into the details of the **operating principle of DC motor** its important that we have a clear understanding of [Fleming’s left hand rule](https://www.electrical4u.com/fleming-left-hand-rule-and-fleming-right-hand-rule/) to determine the direction of force acting on the armature conductors of DC motor.



If a current carrying [conductor](https://www.electrical4u.com/electrical-conductor/) is placed in a [magnetic field](https://www.electrical4u.com/what-is-magnetic-field/) perpendicularly, then the conductor experiences a force in the direction mutually perpendicular to both the direction of field and the current carrying conductor. Fleming’s left hand rule says that if we extend the index finger, middle finger and thumb of our left hand perpendicular to each other, in such a way that the middle finger is along the direction of current in the conductor, and index finger is along the direction of magnetic field i.e. north to south pole, then thumb indicates the direction of created mechanical force.

For clear understanding the **principle of DC motor** we have to determine the magnitude of the force, by considering the diagram below.



We know that when an infinitely small charge dq is made to flow at a velocity ‘v’ under the influence of an electric field E, and a magnetic field B, then the Lorentz Force dF experienced by the charge is given by:-

dF=dq(E+vB)

For the **operation of DC motor**, considering E = 0

dF=dq\*v\*B

i.e. it’s the cross product of dq v and magnetic field B.

dF=dq(dq/dt)B [V=dL/dt]

Where dL is the length of the conductor carrying charge q.

dF=dq(dL/dt)\*B

or,

[since,current I=dq/dt]

dF=IdL\*B

or,

F=IL\*B=ILBsin

Or,

F=BILsin

From the 1st diagram we can see that the construction of a DC motor is such that the direction of current through the armature conductor at all instance is perpendicular to the field. Hence the force acts on the armature conductor in the direction perpendicular to the both uniform field and current is constant.

i.e

So if we take the current in the left hand side of the armature conductor to be I, and current at right hand side of the armature conductor to be − I, because they are flowing in the opposite direction with respect to each other. Then the force on the left hand side armature conductor,

Fi=BILsin90=BIL

Similarly force on the right hand side conductor

Fr=B(-I)Lsin90=-BIL

Therefore, we can see that at that position the force on either side is equal in magnitude but opposite in direction. And since the two conductors are separated by some distance w = width of the armature turn, the two opposite forces produces a rotational force or a torque that results in the rotation of the armature conductor. Now let's examine the expression of torque when the armature turn crate an angle of α (alpha) with its initial position.

The torque produced is given by,

Torque = (force,tangential to the direction of armature rotation)\*(distance)

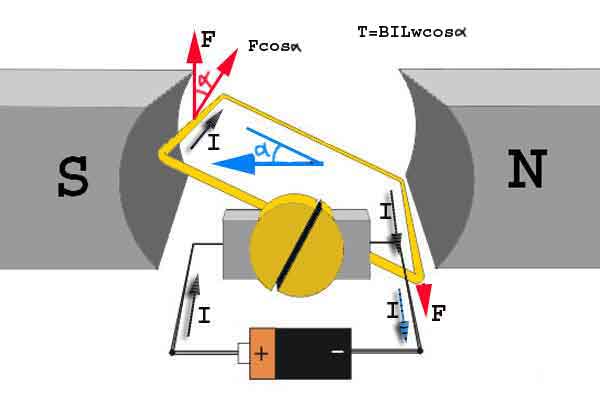
Or,

\*

Or,

Where, α (alpha) is the angle between the plane of the armature turn and the plane of reference or the initial position of the armature which is here along the direction of magnetic field. The presence of the term cosα in the torque equation very well signifies that unlike force the torque at all position is not the same. It in fact varies with the variation of the angle α (alpha).

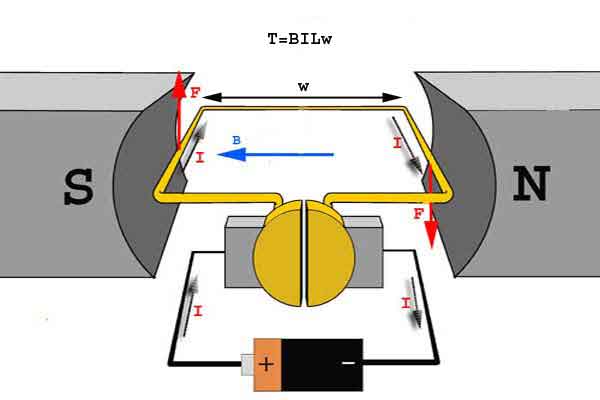
To explain the variation of torque and the principle behind rotation of the motor let us do a step wise analysis.



**Step 1:** Initially considering the armature is in its starting point or reference position where the angle α = 0.

So

Since, α = 0, the term cos α = 1, or the maximum value, hence torque at this position is maximum given by τ = BILw. This high starting torque helps in overcoming the initial inertia of rest of the armature and sets it into rotation.

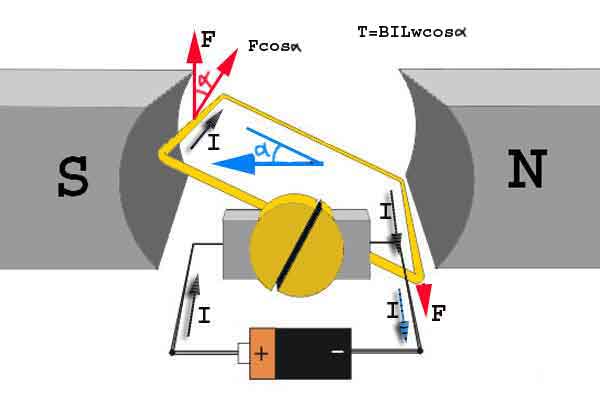


Step 2: Once the armature is set in motion, the angle α between the actual position of the armature and its reference initial position goes on increasing in the path of its rotation until it becomes 90° from its initial position. Consequently the term cosα decreases and also the value of torque.

The torque in this case is given by

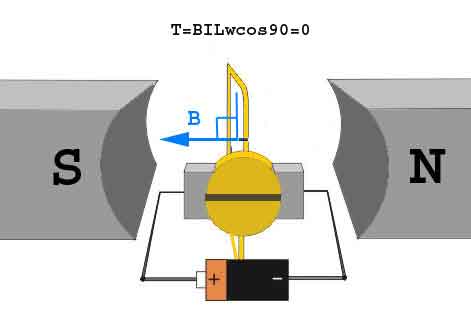
τ = BILwcosα

which is less than BIL w when α is greater than 0°.



Step 3: In the path of the rotation of the armature a point is reached where the actual position of the rotor is exactly perpendicular to its initial position, i.e. α = 90°, and as a result the term cosα = 0. The torque acting on the conductor at this position is given by,

So,



i.e. virtually no rotating torque acts on the armature at this instance. But still the armature does not come to a standstill, this is because of the fact that the operation of DC motor has been engineered in such a way that the inertia of motion at this point is just enough to overcome this point of null torque. Once the rotor crosses over this position the angle between the actual position of the armature and the initial plane again decreases and torque starts acting on it again.

1. ARDUINO PROGRAMMING

Arduino Programming Language Components

1. Structure

* Basic syntax
* Arithmetic operators
* Control structures
* Comparison Operators
* Boolean Operators

1. Variables

* Constants
* Data types
* Scope

1. Functions

* Digital I/O
* Analog I/O
* Math
* Serial communication
* Defining your own
  + - 1. OBSTACLE AVOIDING PROGRAM

//Obstacle avoiding program

#include <NewPing.h>

#include <Servo.h>

#define TRIG\_PIN A0

#define ECHO\_PIN A1

#define MAX\_DISTANCE 200

NewPing sonar(TRIG\_PIN, ECHO\_PIN, MAX\_DISTANCE);

Servo myservo;

boolean goesForward=false;

int distance = 100;

int speedSet = 0;

const int motorPin1 =7;

const int motorPin2 =6;

//Motor B

const int motorPin3 =5;

const int motorPin4 =4;

void setup(){

myservo.attach(9);

myservo.write(115);

delay(2000);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

}

void loop (){

int distanceR = 0;

int distanceL = 0;

delay(40);

if(distance<=30)

{

moveStop();

delay(100);

moveBackward();

delay(300);

moveStop();

delay(200);

distanceR = lookRight();

delay(200);

distanceL = lookLeft();

delay(200);

if(distanceR>distanceL)

{

turnRight();

moveStop();

}

else

{

turnLeft();

moveStop();

}

}

else

{

moveForward();

}

distance = readPing();

}

int lookRight()

{

myservo.write(50);

delay(500);

int distance = readPing();

delay(100);

myservo.write(115);

return distance;

}

int lookLeft()

{

myservo.write(170);

delay(500);

int distance = readPing();

delay(100);

myservo.write(115);

return distance;

delay(100);

}

int readPing(){

delay(70);

int cm = sonar.ping\_cm();

if(cm==0)

{

cm=250;

}

return cm;

}

void moveStop(){

analogWrite(motorPin1,0);

analogWrite(motorPin2,0);

analogWrite(motorPin3,0);

analogWrite(motorPin4,0);

}

void moveForward(){

analogWrite(motorPin1,180);

analogWrite(motorPin2,0);

analogWrite(motorPin3,180);

analogWrite(motorPin4,0);

}

void moveBackward(){

analogWrite(motorPin1,0);

analogWrite(motorPin2,180);

analogWrite(motorPin3,0);

analogWrite(motorPin4,180);

}

void turnRight(){

analogWrite(motorPin1,180);

analogWrite(motorPin2,0);

analogWrite(motorPin3,0);

analogWrite(motorPin4,180);

delay(300);

moveForward();

}

void turnLeft(){

analogWrite(motorPin1,0);

analogWrite(motorPin2,180);

analogWrite(motorPin3,180);

analogWrite(motorPin4,0);

delay(300);

moveForward();

}

* + 1. PROGRAM CONTROL BY BLUETOOTH

int izqA = 7;

int izqB = 6;

int derA = 5;

int derB = 4;

int vel = 200; // Velocidad de los motores (0-255)

int estado = 'g'; // inicia detenido

void setup() {

Serial.begin(9600); // inicia el puerto serial para comunicacion con el Bluetooth

pinMode(derA, OUTPUT);

pinMode(derB, OUTPUT);

pinMode(izqA, OUTPUT);

pinMode(izqB, OUTPUT);

}

void loop() {

if(Serial.available()>0){ // lee el bluetooth y almacena en estado

estado = Serial.read();

}

if(estado=='a'){ //Forward

Serial.println(estado);

digitalWrite(derB, 0);

digitalWrite(izqB, 0);

digitalWrite(derA, vel);

digitalWrite(izqA, vel);

}

if(estado=='d'){ // Right

Serial.println(estado);

digitalWrite(derB, vel);

digitalWrite(izqB, 0);

digitalWrite(derA, 0);

digitalWrite(izqA, vel);

}

if(estado=='c'){ // Stop

Serial.println(estado);

digitalWrite(derB, 0);

digitalWrite(izqB, 0);

digitalWrite(derA, 0);

digitalWrite(izqA, 0);

}

if(estado=='b'){ // left

Serial.println(estado);

digitalWrite(derB, 0);

digitalWrite(izqB, vel);

digitalWrite(izqA, 0);

digitalWrite(derA, vel);

}

if(estado=='e'){ // Reverse

Serial.println(estado);

digitalWrite(derA, 0);

digitalWrite(izqA, 0);

digitalWrite(derB, vel);

digitalWrite(izqB, vel);

}

if(estado=='f'){ // Boton ON se mueve sensando distancia

digitalWrite(izqB, vel);

digitalWrite(izqA, vel);

digitalWrite(derA, vel);

digitalWrite(derB, vel);

}

if(estado=='g'){ // Boton OFF, deteine los motores no hace nada

digitalWrite(izqB, 0);

digitalWrite(izqB, 0);

digitalWrite(derB, 0);

digitalWrite(derB, 0);

}

}

* + 1. COMBINED SENSOR OR BLUETOOTH MODULE PROGRAM

#include <NewPing.h>

#include <Servo.h>

#define TRIG\_PIN A4

#define ECHO\_PIN A5

#define MAX\_DISTANCE 200

NewPing sonar(TRIG\_PIN, ECHO\_PIN, MAX\_DISTANCE);

Servo myservo;

boolean goesForward=false;

int distance = 100;

int speedSet = 0;

int pluto = 'a';

const int motorPin1 =7;

const int motorPin2 =6;

//Motor B

const int motorPin3 =5;

const int motorPin4 =4;

void setup(){

Serial.begin(9600);

myservo.attach(9);

myservo.write(115);

delay(200);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

}

void loop (){

int distanceR =0;

int distanceL =0;

delay(40);

if(Serial.available()>0)

pluto = Serial.read();

{

switch (pluto)

{

case 'c':

moveStop();

return moveStop();

case 'a':

break;

}

}

if(distance<=30)

{

moveStop();

delay(100);

moveBackward();

delay(300);

moveStop();

delay(100);

distanceR = lookRight();

delay(200);

distanceL = lookLeft();

delay(200);

if(distanceR>distanceL)

{

turnRight();

turnRight();

moveStop();

}

else

{

turnLeft();

turnLeft();

turnLeft();

moveStop();

}

}

else

{

moveForward();

}

distance = readPing();

}

int lookRight()

{

myservo.write(50);

delay(500);

int distance = readPing();

delay(100);

myservo.write(115);

return distance;

}

int lookLeft()

{

myservo.write(170);

delay(500);

int distance = readPing();

delay(100);

myservo.write(115);

return distance;

delay(100);

}

int readPing(){

delay(70);

int cm = sonar.ping\_cm();

if(cm==0)

{

cm=250;

}

return cm;

}

void moveStop(){

analogWrite(motorPin1,0);

analogWrite(motorPin2,0);

analogWrite(motorPin3,0);

analogWrite(motorPin4,0);

}

void moveForward(){

analogWrite(motorPin1,150);

analogWrite(motorPin2,0);

analogWrite(motorPin3,150);

analogWrite(motorPin4,0);

}

void moveBackward(){

analogWrite(motorPin1,0);

analogWrite(motorPin2,150);

analogWrite(motorPin3,0);

analogWrite(motorPin4,150);

}

void turnRight(){

analogWrite(motorPin1,150);

analogWrite(motorPin2,0);

analogWrite(motorPin3,0);

analogWrite(motorPin4,150);

delay(300);

moveForward();

}

void turnLeft(){

analogWrite(motorPin1,0);

analogWrite(motorPin2,150);

analogWrite(motorPin3,150);

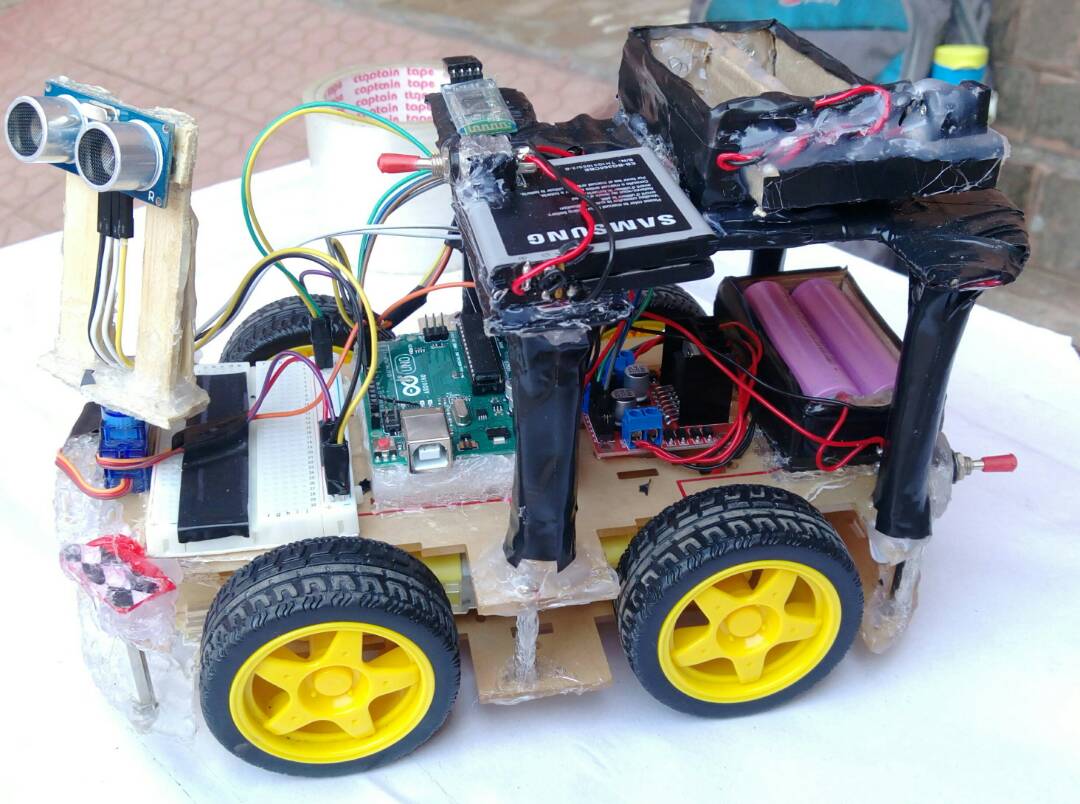
analogWrite(motorPin4,0);

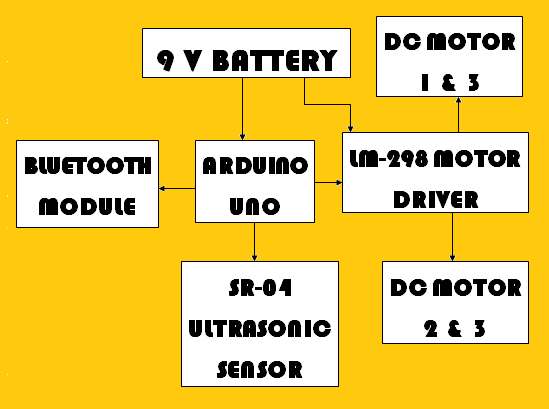
delay(300);

moveForward();

}

8.PROJECT EXPLANATION





This car works on two different platforms. First one is

blurtooth controlled via android phone and other one with ultrasonic sensor.

Bluetooth Controlled -

You just need to download the Arduino Car Control application in your android phone( link provided at the end )

and you are good to go! First pair your bluetooth device with the bluetooth module used in the car, then connect using the app.

Now just press the control buttons similar to any game controller and is easy to understand.

Ultrasonic Sensor -

It is very simple in working. Just turn on the switch and let the sensor

take control. It detects any obstacle within its provided range

and decides itself whether to take a right or left turn.

One more feature added to this is the combining of bluetooth module.

Now, to turn the robot off again, there is no need to catch the

car or run behind it for the switch. Just connect with the bluetooth module using the car

control app and press the button programmed to stop the car

and you could start it again with the other button progreammed

to start the car again!

1. Applications and Advantages:
   * + The robot is small in size so can be used for spying.
     + With few additions and modifications, this robot can be used in the borders for detecting and disposing hidden land mines.
     + The robot can be used for reconnaissance or surveillance.
2. Future Development:
   * + We can interface sensors to this robot so that it can monitor some parameters.
     + We can add wireless camera to this robot.