

A Report on
DTMF CONTROLLED ROBOT USING ARDUINO

INDEX

CONTENTS	page no.
Abstract	I
List of figures	II
List of tables	III
CHAPTER 1: INTRODUCTION	
1.1 overview	1
CHAPTER 2: PHYSICAL DESCRIPTION	
2.1 Block diagram	2
CHAPTER 3: HARDWARE DESCRIPTION	
3.1 Introduction	4
3.2 DTMF decoder	4
3.3 ARDUINO UNO	12
3.4 L293D motor driver	21
3.5 DC motors	22
CHAPTER 4: SCHEMATIC DIAGRAM	
4.1 Circuit diagram	26
CHAPTER 5: SOFTWARE DESCRIPTION	
5.1 ARDUINO IDE	27
5.2 Writing sketches	27

5.3	Uploading	27
5.4	Code	28
5.5	Arduino coding procedure	31

CHAPTER 6: RESULT

6.1	Output	45
-----	--------	----

CHAPTER 8: ADVANTAGES AND APPLICATIONS

7.1	Advantages	46
7.2	Applications	46

CHAPTER 9: CONCLUSION

7.1	Conclusion	47
-----	------------	----

CHAPTER 10: FUTURE SCOPE

8.1	Future Scope	48
-----	--------------	----

REFERENCE

LIST OF FIGURES

Figure No.	Name Of The Figure	Page No.
1.1	Dtmf robot	1
2.1	Diagram of DTMF Controlled Robot	2
3.2.2	MT8870 Module	5
3.2.3	Mobile Connected To Module	6
3.2.4	Power Supply to Board	6
3.2.5	Connect Output Pins	7
3.2.6	Make A Call	7
3.2.7	Accept the Call	8
3.2.8	Give Inputs	8
3.2.9	LED's On Particular Input	9
3.2.10	Decoder Circuit	11
3.3.1	Arduino Uno Board Description	13
3.3.2	Pin Description Of Atmega328p	18
3.4.1	L293D Pin Diagram	22
3.5.1	Left Hand Rule	24
3.5.2	Current Carrying Wire	24
3.5.3	Commutator	25
4.1	Connection Diagram	26
5.5.1	Arduino IDE	31
5.5.2	Plug It In	32
5.5.3	Settings	33
5.5.4	Sketch Running	34
5.5.5	Serial Monitor	37
5.5.6	Writing Code	37
6.1	DTMF Controlled Robot	48

LIST OF TABLES

Table No.	Name Of The Table	Page No.
3.2.1	Output Bits Of MT8870	5
3.3.1	Arduino Uno Features	12
3.3.2	Parallel Programming Mode	20
3.3.3	Serial Programming Mode	21

ABSTRACT

In this project, we can control the Robot using Dual Tone Multi Frequency (DTMF) technology. DTMF technology is most useful technique at present days. It is worked on to methods digital signal processing (DSP). Wireless-control of robots uses RF circuit that has the drawbacks of limited working range and limited control.

This DTMF gives advantage over the RF; it increases the range of working and also gives good results in case of motion and direction of robot using mobile phone through micro controller. This type of wireless communication gives the remote handling operation of Robot using DTMF.

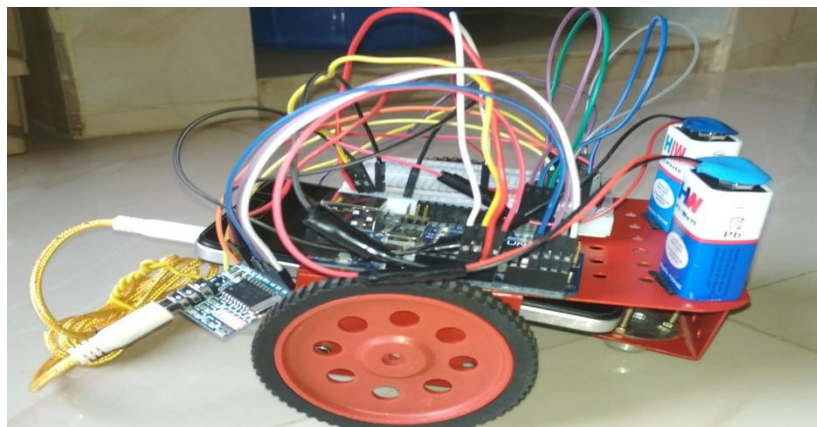
CHAPTER 1

INTRODUCTION

1.1 OVERVIEW:

A robot is electro-mechanical machine which is guided by computer, Mobile phone or programming, and is thus able to do tasks on its own. The Robot Institute of America define “A robot is a reprogrammable multifunctional manipulator -designed to move material parts, tools or specialized device through variable programmed motions for the performance of a variety of tasks.” Conventionally, wireless controlled robots use RF circuits, which have drawbacks of limited working range & frequency range, use of mobile phones can overcome this limitation. It provides the advantages of robust control, working range as large as the coverage area of the service provider, no interference with other controllers and up to twelve controls.

DTMF Mobile ROBOT is a machine that can be controlled with a mobile. In this project, the robot is controlled by a mobile phone that makes a call to the mobile phone attached to the robot. In the course of a call, if any button is pressed, a tone corresponding to the button pressed is heard at the other end of the call. This tone is called "Dual Tone Multiple-Frequency" (DTMF) tone. The robot perceives this DTMF tone with the help of the phone stacked on the robot. The received tone is processed by the Arduino microcontroller with the help of DTMF decoder MT8870 IC the decoder decodes the DTMF tone is to its equivalent binary digit and this binary number is send to the microcontroller, the microcontroller is pre-programmed to take decision for any give input and output its decision to motor drivers in order to drive the motors for forward or backward motion or a turn.



CHAPTER 2

BLOCK DIAGRAM



Fig 2.1 Block Diagram Of DTMF Controlled Robot

The block diagram of DTMF controlled robot consists of transmitter and receiver mobile phones, an DTMF decoder module (MT8870), Arduino UNO, L293D motor driver IC and DC motors.

Firstly, the mobile phone which acts as remote calls to the receiver mobile phone which is connected to the robot. When the call is received by the mobile phone connected to the robot accepts the call automatically (because of the auto answering mode enabled on receiver mobile phone). The inputs are given from remote mobile

phone or user mobile phone using the keypad and a combination of sine wave i.e. DTMF tone is generated and sent to the receiver mobile phone.

Now, the received DTMF tone is fed to the DTMF decoder module which converts this DTMF tone into binary equivalent i.e. 0's or 1's. This DTMF decoder module accepts the input signal and produces four binary outputs (Q1, Q2, Q3, Q4).

Now, these outputs are sent to the arduino which is programmed to accept the inputs from DTMF decoder and process them and produce the specified output. The output of arduino is taken from D3, D4, D5 and D6 pins and fed to the IN1, IN2, IN3 and IN4 pins of L293D motor driver IC. This L293D motor driver IC acts as interface between arduino and DC motors. These IC's are designed to control two DC motors simultaneously. Depending upon the values of the input and enable the motors will rotate in either clockwise or anticlockwise direction with full speed (when enable is HIGH) or with less speed (when enable is provided with PWM).

Let us assume for left motor when enable is HIGH and input 1 and input 2 is are HIGH and LOW respectively then the motor will move in clockwise direction.

CHAPTER 3

HARDWARE DESCRIPTION

3.1 INTRODUCTION:

DTMF controlled robot uses the below hardware components

- DTMF decoder
- ARDUINO UNO
- L293D motor driver
- DC motors

3.2 DTMF DECODER

DTMF means Dual-Tone-Multi-Frequency. DTMF signaling is used for telecommunication signaling over analog telephone lines in the voice-frequency band between telephone handsets and other communication devices and the switching centre. The DTMF system generally uses eight different frequency signals transmitted in pairs to represent sixteen different numbers, symbols and letters. When someone presses any key in the key pad of the handset, a DTMF signal is generate unique tone which consists of two different frequencies one each of higher frequency range ($>1\text{KHz}$) and lower frequency ($<1\text{KHz}$) range. The resultant tone is convolution of two frequencies. The frequencies and their corresponding frequency are shown in Table 3.1. Each of these tones is composed of two pure sine waves of the low and high frequencies superimposed on each other. These two frequencies explicitly represent one of the digits on the telephone keypad. Thus generated signal can be expressed mathematically as follows:

For more details you can see the datasheet of MT8870 IC.

F _{LOW}	F _{HIGH}	Key (ref.)	OE	Q4	Q3	Q2	Q1
697	1209	1	H	0	0	0	1
697	1336	2	H	0	0	1	0
697	1477	3	H	0	0	1	1
770	1209	4	H	0	1	0	0
770	1336	5	H	0	1	0	1
770	1477	6	H	0	1	1	0
852	1209	7	H	0	1	1	1
852	1336	8	H	1	0	0	0
852	1477	9	H	1	0	0	1
941	1336	0	H	1	0	1	0
941	1209	*	H	1	0	1	1
941	1477	#	H	1	1	0	0
697	1633	A	H	1	1	0	1
770	1633	B	H	1	1	1	0
852	1633	C	H	1	1	1	1
941	1633	D	H	0	0	0	0
ANY	ANY	ANY	L	Z	Z	Z	Z

L = logic low, H = logic high, Z = high impedance

Table 3.2.1 Output Bits Of MT8870 Corresponding To Different

This module lets you receive and decode DTMF signal and decode them by generating a 4-bit output. You can connect this module to a mobile phone and receive DTMF signal from another mobile phone far away.

Thus you can send DTMF signal from one mobile phone and receive the with another 4 phone connected to the module. The module receives the signal via mobile phone and generates a 4-Bit output. Now, if you connect 4 different loads with the output pins, you can switch on codes from your mobile phone no matter where you are.



Fig 3.2.2 MT8870 Module

MT8870: MT8870 is the core of this module.

Signal connector and DC jack: compatible DC jack. These connector and jack let you connect the module with your mobile phone.

Power pins: +5V and GND pins should be connected to +5V and GND pins of a DC power supply respectively.

Power switch: After connecting the power pins with the power source, the white switch should be pressed to give power to the board.

Power Indicator LED: The Red power indicator LED will glow as soon as the board gets power.

Signal indicator LED: The white signal indicator LED blinks once every time the connected mobile phone receives a signal from another mobile phone.

STD pin: This pin goes high every time a communication takes place.

Output pins: There are 4 output pins namely Output1-Output4. Here we will get the 4-bit output signal generated by the MT8870 DTMF decoder corresponding to the signal received by the mobile phone.

Let's do a simple experiment:

Step 1: At first connect the module with a mobile phone like this. Make sure that the mobile has a valid sim inside and the DC jack is supported by the mobile as shown in fig 3.2.3.



Fig 3.2.3 Mobile Connected To Module

Step 2: Give Power supply to the board. Here we have used USB breakout board for 5V power supply from a computer's USB port. You can use any 5V power supply. Press the white switch and the Red LED will get power.

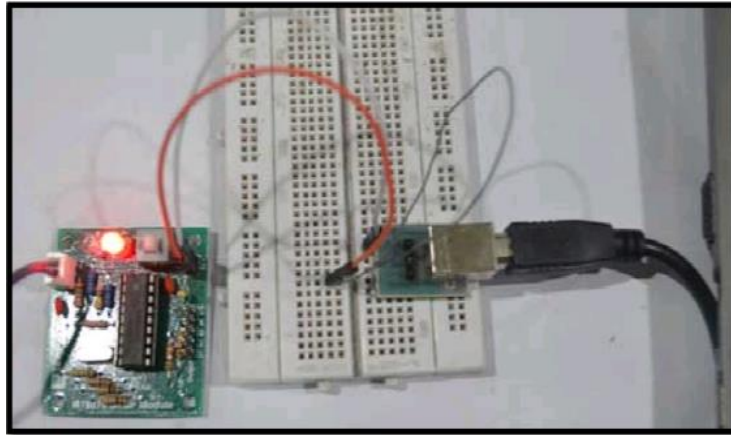


Fig 3.2.4 Power Supply To Board

Step 3: Connect the Output pins.

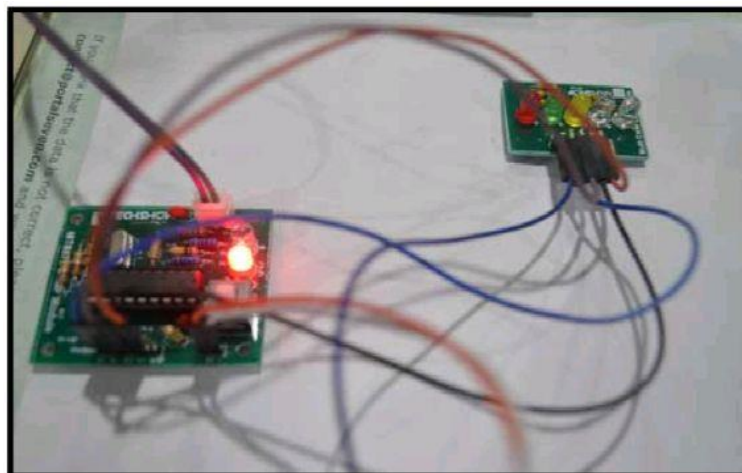


Fig 3.2.5 Connect Output Pins

Step 4: Now call the connected mobile number 017xxxxxxx from another mobile. Connect the Output pins(Q1-Q4) with four LEDs.



Fig 3.2.6 Make A Call

8Step 5: Receive the call.

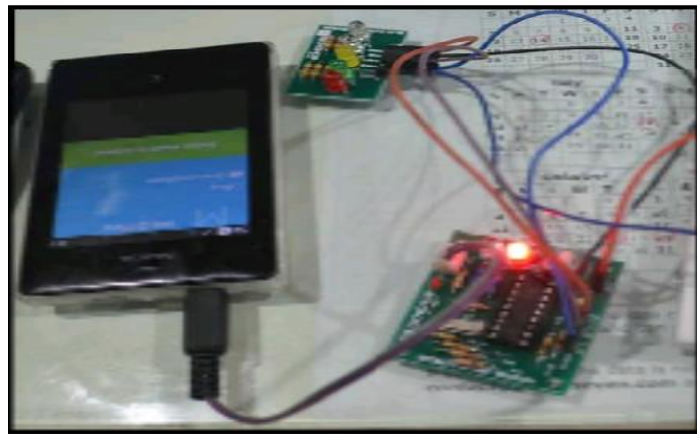


Fig 3.2.7 Accept The Call

Step 6: Press digits 1, 2, phone.

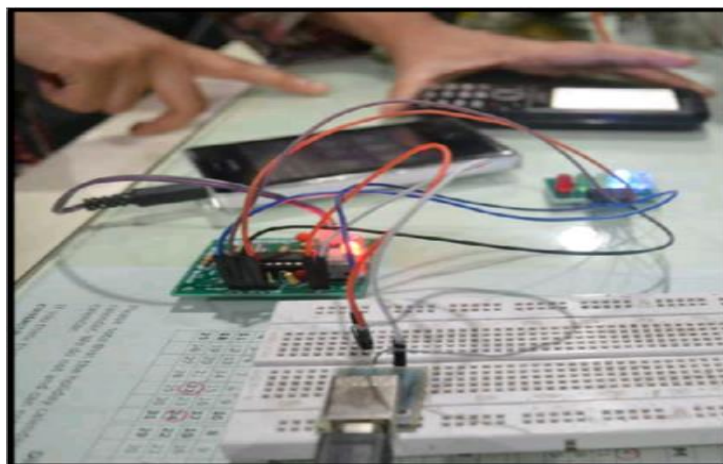


Fig 3.2.8 Give Inputs

Step 7: See the changes in LEDs corresponding to your sent signal. For example, if you send '1' only the LED connected to Q1 will glow. Send signals as you wish and match the results with chart 2.

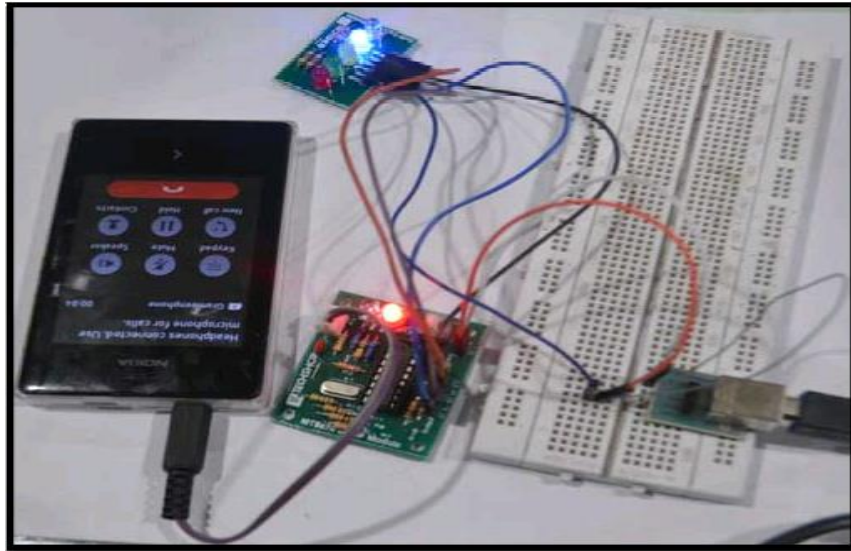


Fig 3.2.9 LED's Blinking On Particular Input

DTMF (Dual Tone Multi Frequency) decoder Circuit using M8870:

This DTMF (Dual Tone Multi Frequency) decoder circuit identifies the dial tone from the telephone line and decodes the key pressed on the remote telephone. Here for the detection of DTMF signaling, we are using the IC MT8870DE which is a touch tone decoder IC. It decodes the input DTMF to 5 digital outputs. The M-8870 DTMF (Dual Tone Multi Frequency) decoder IC uses a digital counting technique to determine the frequencies of the limited tones and to verify that they correspond to standard DTMF frequencies.

The DTMF tone is a form of one way communication between the dialer and the telephone exchange. The whole communication consists of the touch tone initiator and the tone decoder or detector. The decoded bits can be interfaced to a computer or microcontroller for further application (For example, Remote control of home/office electrical appliances using a telephone network, Cell Phone controlled home appliances, Mobile phone controlled robot, etc.).

COMPONENTS REQUIRED:

1. DTMF decoder IC (M-8870)
2. Resistors (100k Ω ; 70k Ω ; 390k Ω)
3. Capacitors (0.1 μ Fx 2)
4. Crystal oscillator (3.579545MHz)

In the premature days, our telephone systems were operated by human operators in a telephone exchange room. The caller will pick up the phone, giving instruction to the operator to connect their line to the destination. It is a kind of manual switching. As more and more people entered in the telephone technology as useful communication gear, manual switching becomes a time consuming tedious task.

The operation of DTMF method are as follows:

- Caller generates a dial tone consisting of two frequencies. It is transmitted via the telephone line (communication media).
- Telephone exchange consists of a DTMF decoder, which decodes the frequencies in to digital code.
- These codes are the address of destination subscriber; it is read and processed by a computer which connects caller to the destination subscriber.

Working of DTMF decoder circuit:

- DTMF keypads are employed in almost all landline and mobile handsets. Thus this technology is used in the telephone switching centers to identify the number dialed by the caller.
- The decoder distinguishes the DTMF tones and produces the binary sequence equivalent to key pressed in a DTMF (Dual Tone Multi Frequency) keypad.
- The circuit uses M-8870 DTMF decoder IC which decodes tone generated by the keypad of cell phone.
- DTMF signals can be tapped directly from the microphone pin of cell phone device. Cut the microphone wire and you will get two wires red and green. The red wire is the DTMF input to the circuit.

- The signals from the microphone wire are processed by the DTMF decoder IC which generates an equivalent binary sequence as a parallel output like Q1, Q2, Q3, and Q4.

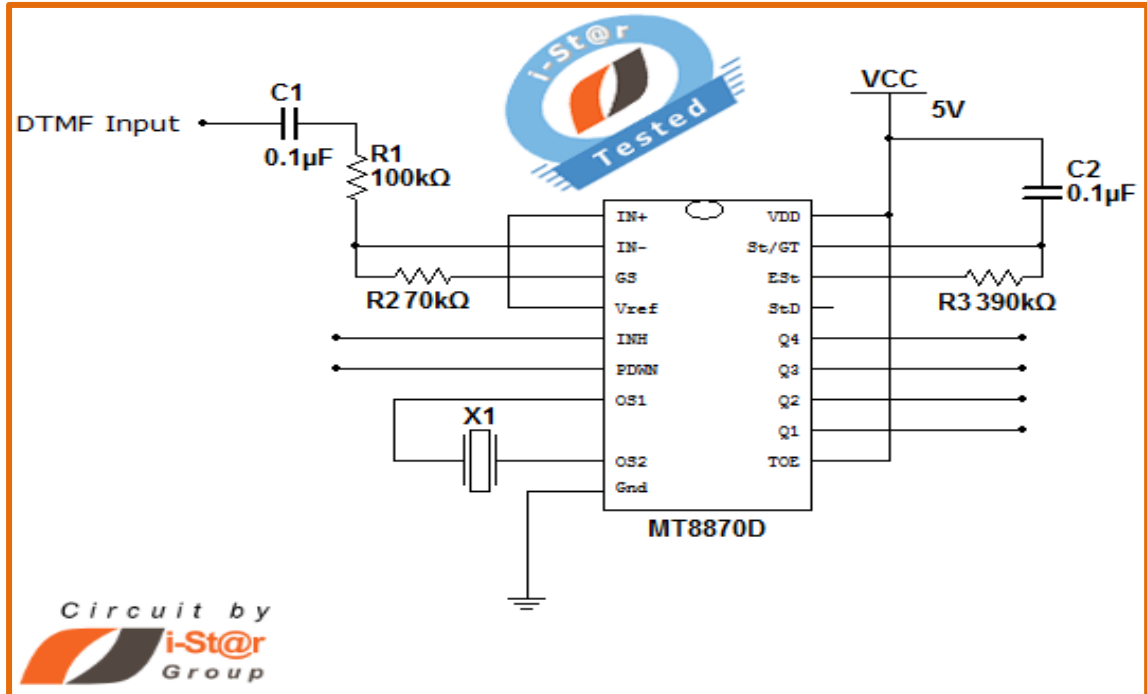


Fig 3.2.10 Decoder Circuit

Uses of other pins:

- The entire process from frequency detection to latching of the data, is controlled by steering control circuit consisting of St/GT, Est pins, resistor (390kΩ) and a capacitor (0.1µF).
- 5th Pin, INH is an active high pin, inhibits detection of A, B, C, D tones of character.
- 6th Pin, PWDN is an (active high), inhibits the working of oscillator thus stops the working of our circuit.
- The 10th pin 10; TOE is the output enable pin which is active high logic and enables the latching of the data on the data pins Q0, Q1, Q2, and Q3.

- 15th Pin StD is the Data valid pin, turn out to be high on detection of valid DTMF tone or else it remains low.
- Pins 7 (OS1) and 8 (OS2) are used to connect crystal oscillator. An oscillator of frequency 3.579545 MHz is used here.

3.3 ARDUINO UNO:

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm centre-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

Vin: The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

5V: The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.

3V3: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50mA.

GND: Ground pins.

Memory :

The ATmega328 has 32 KB (with 0.5 KB used for the boot loader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output :

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

Serial: 0 (RX) and 1 (TX): Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the `attachInterrupt()` function for details.

PWM: 3, 5, 6, 9, 10, and 11: Provide 8-bit PWM output with the `analogWrite()` function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK): These pins support SPI communication.

I 2C: 4 (SDA) and 5 (SCL): Support I2C (TWI) communication using the Wire library. There are a couple of other pins on the board:

AREF: Reference voltage for the analog inputs. Used with `analogReference()`.

Reset: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication:

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming :

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno" from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega8U2 firmware source code is available . The ATmega8U2 is loaded with a DFU bootloader, which can be activated by connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Automatic (Software) Reset:

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board

receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection:

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics:

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16") , not an even multiple of the 100 mil spacing of the other pins.

ATmega328P :

The ATmega328 is a single chip microcontroller created by Atmel in the megaAVR family. The Atmel AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega328P provides the following features: 32Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 1Kbytes EEPROM, 2Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers,

Real Time Counter (RTC), three flexible Timer/Counters with compare modes and PWM, 1 serial programmable USARTs , 1 byte-oriented 2-wire Serial Interface (I2C), a 6- channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages) , a programmable Watchdog Timer with internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main oscillator and the asynchronous timer continue to run.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega328/P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega328/P is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, and Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non volatile memory programmer, or by an On-chip Boot program running on the AVR core.

Pin Description:

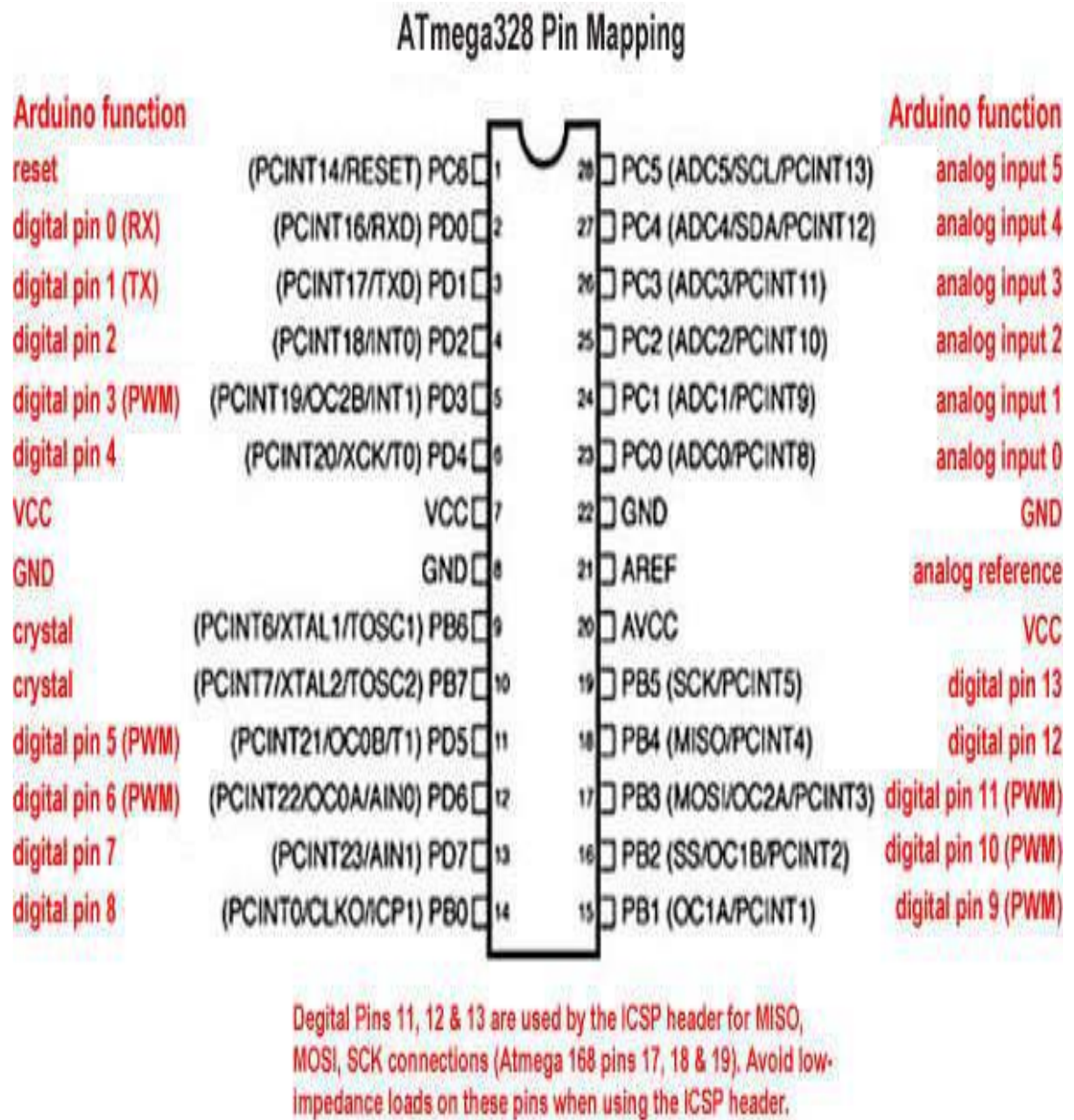


Fig 3.3.2 Pin Description

VCC: Digital supply voltage.

GND: Ground.

Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2 :

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). As inputs, Port B pins that are externally pulled low will

source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

Port C (PC[5:0]):

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

PC6/RESET:

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in the Alternate Functions of Port C section.

Port D (PD[7:0]) :

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability.

AVCC: AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC[6:4] use digital supply voltage, VCC.

AREF: AREF is the analog reference pin for the A/D Converter.

ADC[7:6] (TQFP and VFQFN Package Only): In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

Programming:

Programming mode is entered when PAGEL (PD7), XA1 (PD6), XA0 (PD5), BS1 (PD4) is set to zero. RESET pin to 0V and VCC to 0V. VCC is set to 4.5 - 5.5V. Wait 60 μ s, and RESET is set to 11.5 - 12.5 V. Wait more than 310 μ s.[2] Set XA1:XA0:BS1: DATA = 100 1000 0000, pulse XTAL1 for at least 150 ns, pulse WR to zero. This starts the Chip Erase. Wait until RDY/BSY (PD1) goes high. XA1:XA0:BS1: DATA = 100 0001 0000, XTAL1 pulse, pulse WR to zero. This is the Flash write command. And so on..

Parallel Programming mode:

Programming signal	Pin Name	I/O	Function
RDY/BSY	PD1	O	High means the MCU is ready for a new command, otherwise busy.
OE	PD2	I	Output Enable (Active low)
WR	PD3	I	Write Pulse (Active low)
BS1	PD4	I	Byte Select 1 ("0" = Low byte, "1" = High byte)
XA0	PD5	I	XTAL Action bit 0
XA1	PD6	I	XTAL Action bit 1
PAGEL	PD7	I	Program memory and EEPROM Data Page Load
BS2	PC2	I	Byte Select 2 ("0" = Low byte, "1" = 2nd High byte)
DATA	PC[1:0]:PB[5:0]	I/O	Bi-directional data bus (Output when OE is low)

Table 3.3.2 Parallel Programming Mode

Serial Programming mode:

Serial data to the MCU is clocked on the rising edge and data from the MCU is clocked on the falling edge. Power is applied to VCC while RESET and SCK are set to zero. Wait for at least 20 ms and then the Programming Enable serial instruction 0xAC, 0x53, 0x00, 0x00 is sent to the MOSI pin. The second byte (0x53) will be echoed back by the MCU.

Symbol	Pins	I/O	Description
MOSI	PB3	I	Serial data in
MISO	PB4	O	Serial Data out
SCK	PB5	I	Serial Clock

Table 3.3.3 Serial Programming Mode

3.4 L293D MOTOR DRIVER:

L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.

L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive(L293D) supply.

APPLICATIONS:

Stepper Motor Drivers

DC Motor Drivers

Latching Relay Drivers

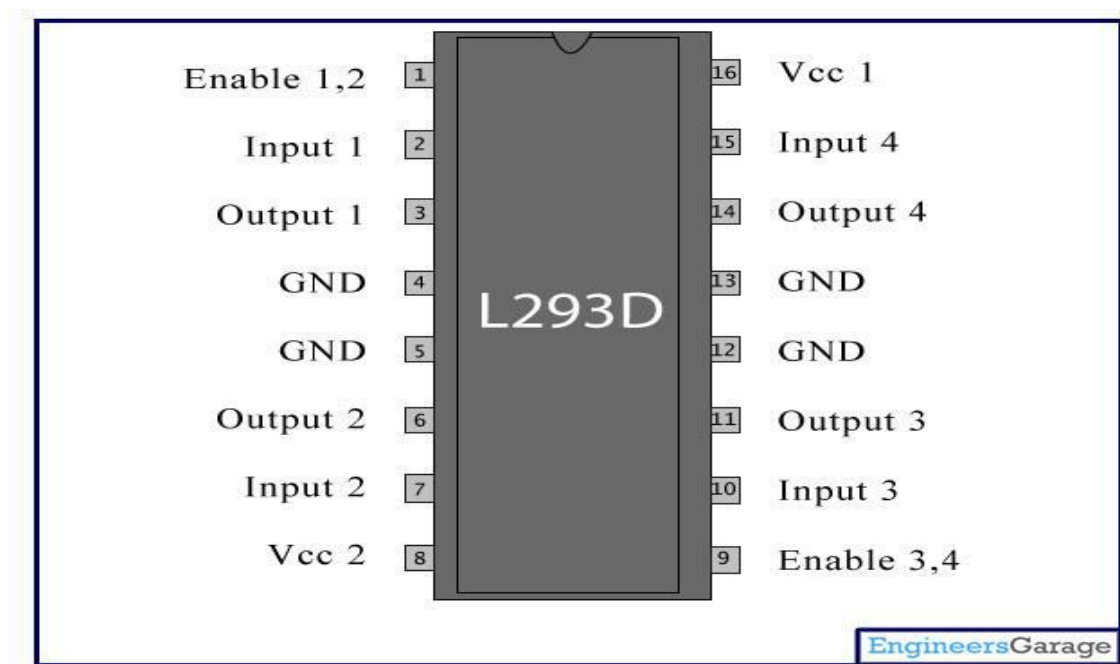


Fig 3.4.1 L293D Pin Diagram

FEATURES:

1. Wide supply-voltage range: 4.5V to 36V
2. Separate input- logic supply
3. Internal ESD protection
4. Thermal shutdown
5. High-Noise-Immunity input
6. Functional Replacements for SGS L293 and SGS L293D
7. Output current 1A per channel (600 mA for L293D)
8. Peak output current 2 A per channel (1.2 A for L293D)
9. Output clamp diodes for Inductive Transient Suppression (L293D).

3.5 DC MOTOR:

This page describes how DC motors work, and how we can use them to build the traction system of a robot. It covers both permanent magnet motors, and series wound motors (such as car starter motors). If you are interested in converting a starter motor for use in a robot, see the separate page [Converting starter motors](#).

Centre Shaft Economy Series DC Motor is high quality low cost DC geared motor. It has steel gears and pinions to ensure longer life and better wear and tear properties. The gears are fixed on hardened steel spindles polished to a mirror finish. The output shaft rotates in a plastic bushing. The whole assembly is covered with a plastic ring. Gearbox is sealed and lubricated with lithium grease and require no maintenance. The motor is screwed to the gear box from inside.

Although motor gives 200 RPM at 12V but motor runs smoothly from 4V to 12V and gives wide range of RPM, and torque. Tables below gives fairly good idea of the motor performance in terms of RPM and no load current as a function of voltage and stall torque, stall current as a function of voltage. For compatible wheels refer to [Wheels and Accessories](#) product category.

You can also mount this motor on the chassis using [Motor Mount for Centre Shaft Economy Series DC Motor](#). For adding Position Encoder, refer to [Encoder Kit for Centre Shaft Economy Series DC Motor](#).

Technical Specifications:

Metal Gears

Operating Voltage: 6-12V

RPM: 300

Central Shaft

Shaft Length: 25mm

Shaft Diameter: 5mm

Wire Length: ~25cm

Weight: ~180 grams

Motor principles:

All motors require two magnetic fields, one produced by the stationary part of the motor (the stator, or field), and one by the rotating part (the rotor, or armature). These are produced either by a winding of coils carrying a current, or by permanent magnets. If the field is a coil of wire, this may be connected in a variety of ways, which produces different motor characteristics.

The basic law of a motor, the reason why they rotate, is governed by Fleming's left hand rule (see figure below). This tells you the direction of the force on a wire that is carrying current when it is in a magnetic field.

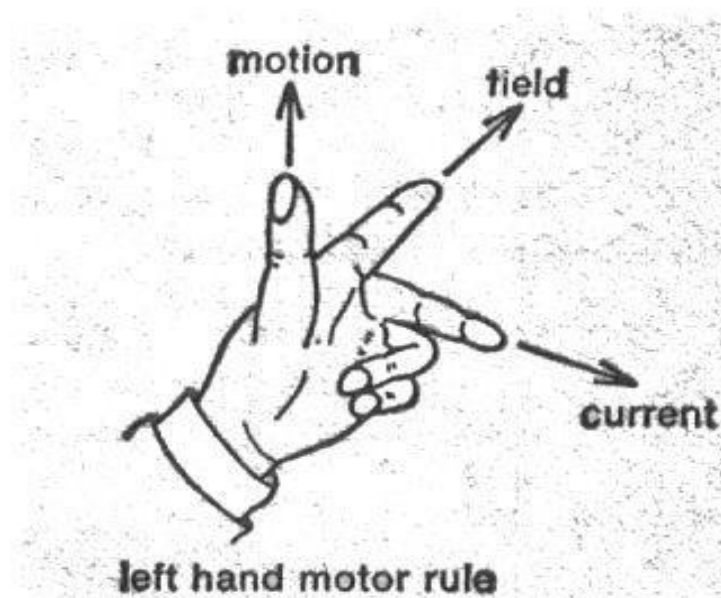


Fig 3.5.1 Left Hand Rule

The next diagram shows the force acting on a wire carrying current, obeying the left hand rule:

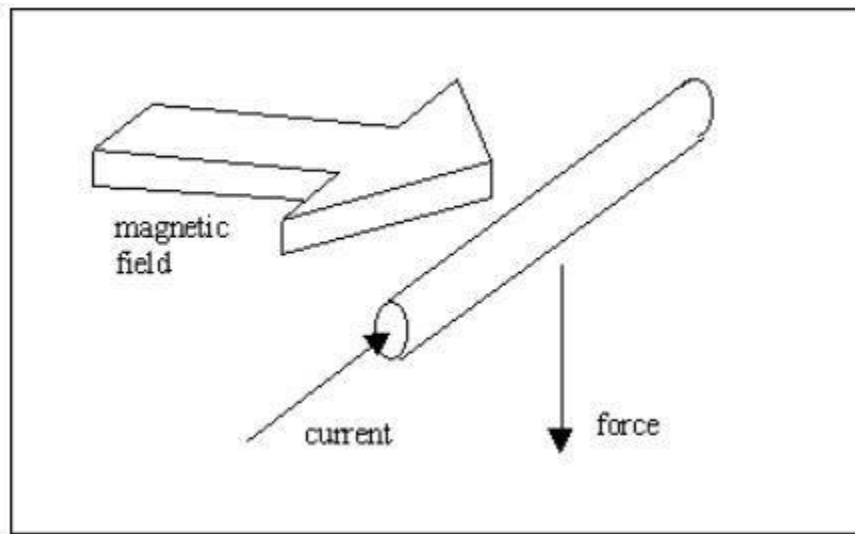


Fig 3.5.2 Current Carrying Wire

If we now bend the wire round in a loop, and place it in a magnetic field caused by two permanent magnets, we have the situation shown in the diagram below. Here, both sides of the wire loop will have a force on them, trying to make the wire loop rotate. The current is applied to the loop through the commutator, which is shown as two pieces of metal formed into a ring in the figure. Current is applied to the commutator by stationary graphite blocks, called brushes, which rub against the commutator ring.

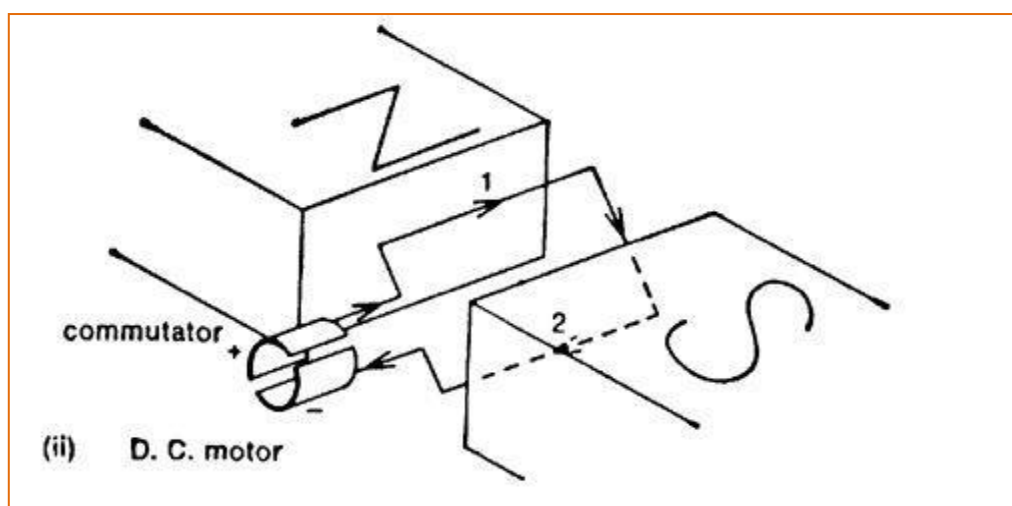


Fig 3.5.3 Commutator

The loop will continue to rotate anticlockwise (as we see it in the figure) until it is vertical. At this point, the stationary brushes won't be applying current around the loop any more because they will be contacting the gap between the commutator segments, but the inertia of the loop keeps it going a little more, until the DC supply reconnects to the commutator segments, and the current then goes around the loop in the opposite direction. The force though is still in the same direction, and the loop continues to rotate.

This is how DC motors work. In a real motor, there are many wire loops (windings) all at varying angles around a solid iron core. Each loop has its own pair of commutator segments. This block of core and wire loops is called the rotor because it rotates, or the armature.

CHAPTER 4

SCHEMATIC DIAGRAM

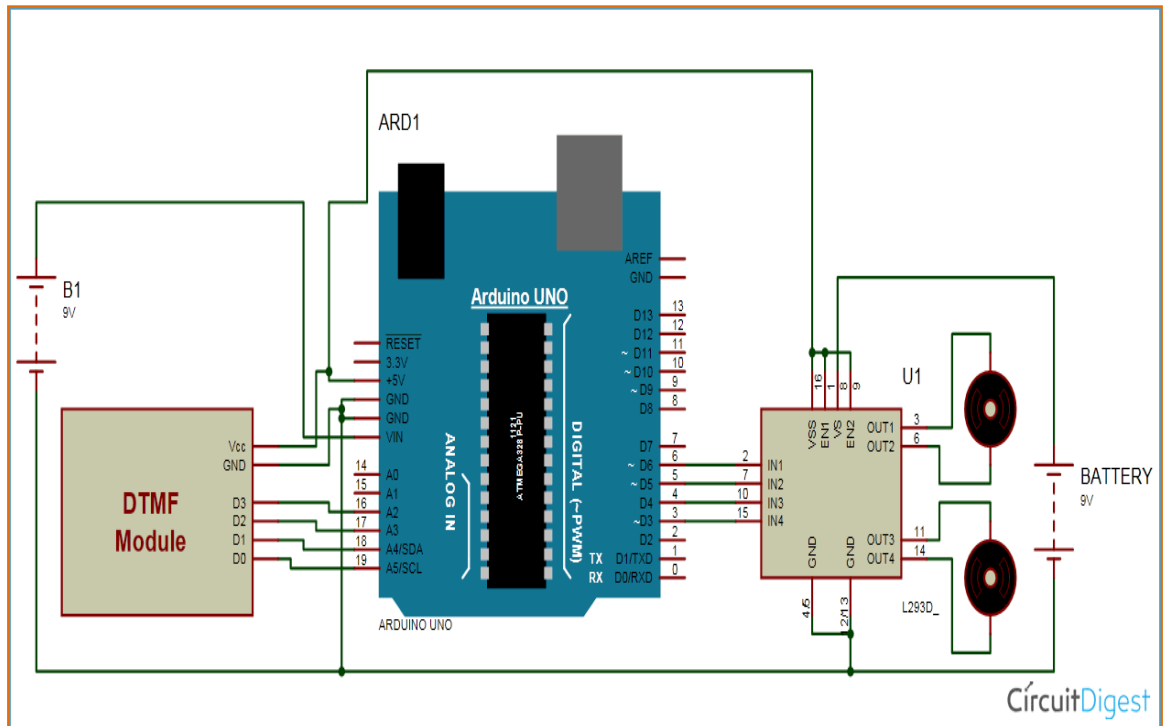


Fig 4.1 Connection Diagram

DTMF decoder module – pin D0 – Arduino UNO A5

DTMF decoder module – pin D1 – Arduino UNO A4

DTMF decoder module – pin D2 – Arduino UNO A3

DTMF decoder module – pin D3 – Arduino UNO A2

Arduino UNO - D3 – L293D – IN4

Arduino UNO - D2 – L293D – IN3

Arduino UNO - D1 – L293D – IN2

Arduino UNO - D0 – L293D – IN1

L293D – OUT1 – MOTOR m11

L293D – OUT2 – MOTOR m12

L293D – OUT3 – MOTOR m21

L293D – OUT4 – MOTOR m22

BATTERY 1 (+) – Arduino Vin

BATTERY 1 (-) – Arduino

GND

BATTERY 2 (+) - L293D pin 8, DTMF module Vin

BATTERY 2 (-) – L293D

GND

CHAPTER 5

SOFTWARE DESCRIPTION

5.1 ARDUINO IDE:

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

5.2 WRITING SKETCHES:

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

5.3 UPLOADING:

Before uploading your sketch, you need to select the correct items from the Tools > Board and Tools > Port menus. The boards are described below. On the Mac, the serial port is probably something like /dev/tty.usbmodem241 (for an Uno or Mega2560 or Leonardo) or /dev/tty.usbserial-1B1 (for a Duemilanove or earlier USB board), or /dev/tty.USA19QW1b1P1.1 (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be /dev/ttyACMx , /dev/ttyUSBx or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the Upload item from the Sketch menu.

With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

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).

5.4 CODE:

```
#define m11 3

#define m12 4

#define m21 5

#define m22 6

#define D0 19

#define D1 18

#define D2 17

#define D3 16

void forward()

{

    digitalWrite(m11, HIGH);

    digitalWrite(m12, LOW);

    digitalWrite(m21, HIGH);

    digitalWrite(m22, LOW);

}
```

```
void backward()

{

    digitalWrite(m11, LOW);

    digitalWrite(m12, HIGH);

    digitalWrite(m21, LOW);

    digitalWrite(m22, HIGH);

}
```

```
void left()

{

    digitalWrite(m11, HIGH);

    digitalWrite(m12, LOW);

    digitalWrite(m21, LOW);

    digitalWrite(m22, LOW);

}
```

```
void right()

{

    digitalWrite(m11, LOW);

    digitalWrite(m12, LOW);

    digitalWrite(m21, HIGH);

    digitalWrite(m22, LOW);

}
```

```
void Stop()

{
```

```
    digitalWrite(m11, LOW);

    digitalWrite(m12, LOW);

    digitalWrite(m21, LOW);

    digitalWrite(m22, LOW);

}

void setup()

{

    pinMode(D0, INPUT);

    pinMode(D1, INPUT);

    pinMode(D2, INPUT);

    pinMode(D3, INPUT);

    pinMode(m11, OUTPUT);

    pinMode(m12, OUTPUT);

    pinMode(m21, OUTPUT);

    pinMode(m22, OUTPUT);

}

void loop()

{

    int temp1=digitalRead(D0);

    int temp2=digitalRead(D1);

    int temp3=digitalRead(D2);

    int temp4=digitalRead(D3);

    if(temp1==0 && temp2==1 && temp3==0 && temp4==0)
```

```
forward();

else if(temp1==0 && temp2==0 && temp3==1 && temp4==0)

left();

else if(temp1==0 && temp2==1 && temp3==1 && temp4==0)

right();

else if(temp1==0 && temp2==0 && temp3==0 && temp4==1)

backward();

else if(temp1==1 && temp2==0 && temp3==1 && temp4==0)

Stop();

}
```

5.5 ARDUINO CODING PROCEDURE:

Step 1: Arduino IDE

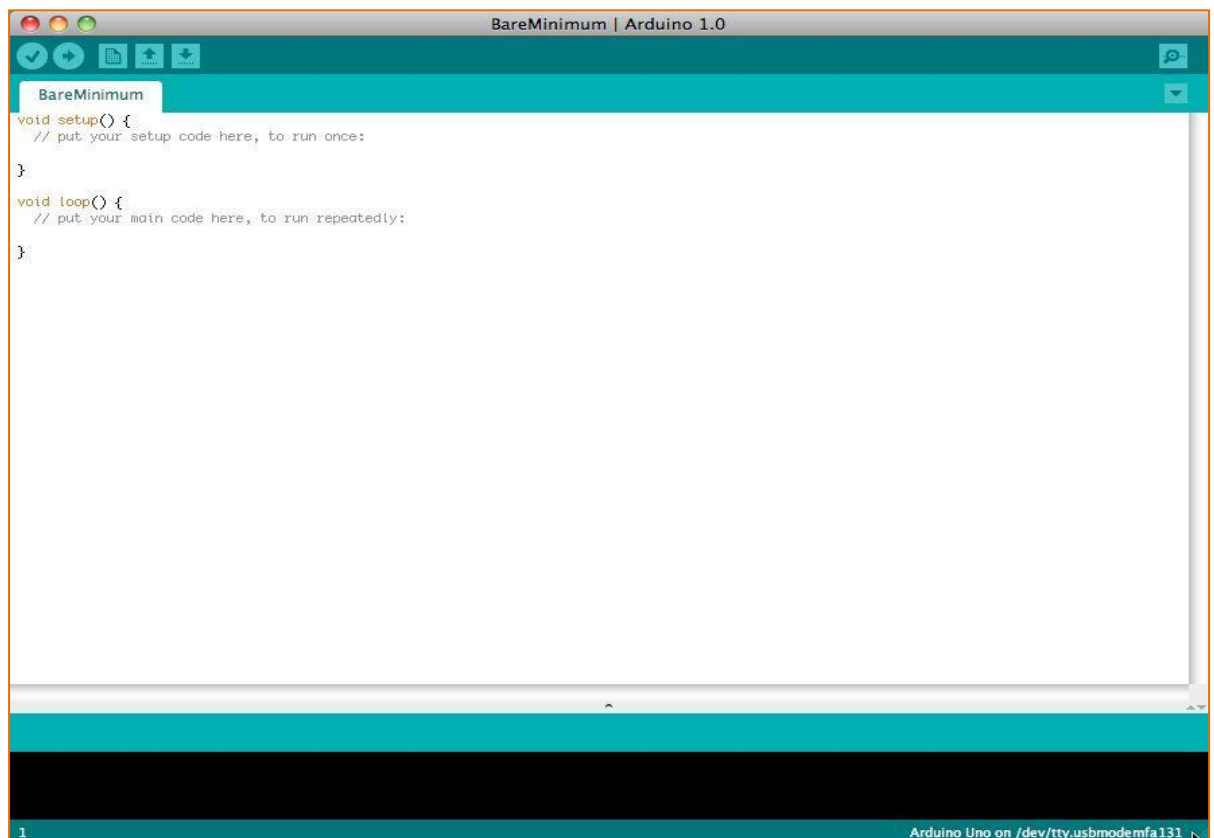


Fig 5.5.1 Arduino IDE

Before you can start doing anything with the Arduino, you need to download and install the Arduino IDE (integrated development environment). From this point on we will be referring to the Arduino IDE as the Arduino Programmer.

The Arduino Programmer is based on the Processing IDE and uses a variation of the C and C++ programming languages.

Step 2: Plug It In



Fig 5.5.2 Plug It In

Connect the Arduino to your computer's USB port.

Please note that although the Arduino plugs into your computer, it is not a true USB device. The board has a special chip that allows it to show up on your computer as a virtual serial port when it is plugged into a USB port. This is why it is important to plug the board in. When the board is not plugged in, the virtual serial port that the Arduino operates upon will not be present (since all of the information about it lives on the Arduino board).

It is also good to know that every single Arduino has a unique virtual serial port address. This means that every time you plug in a different Arduino board into your computer, you will need to reconfigure the serial port that is in use.

Step 3: Settings

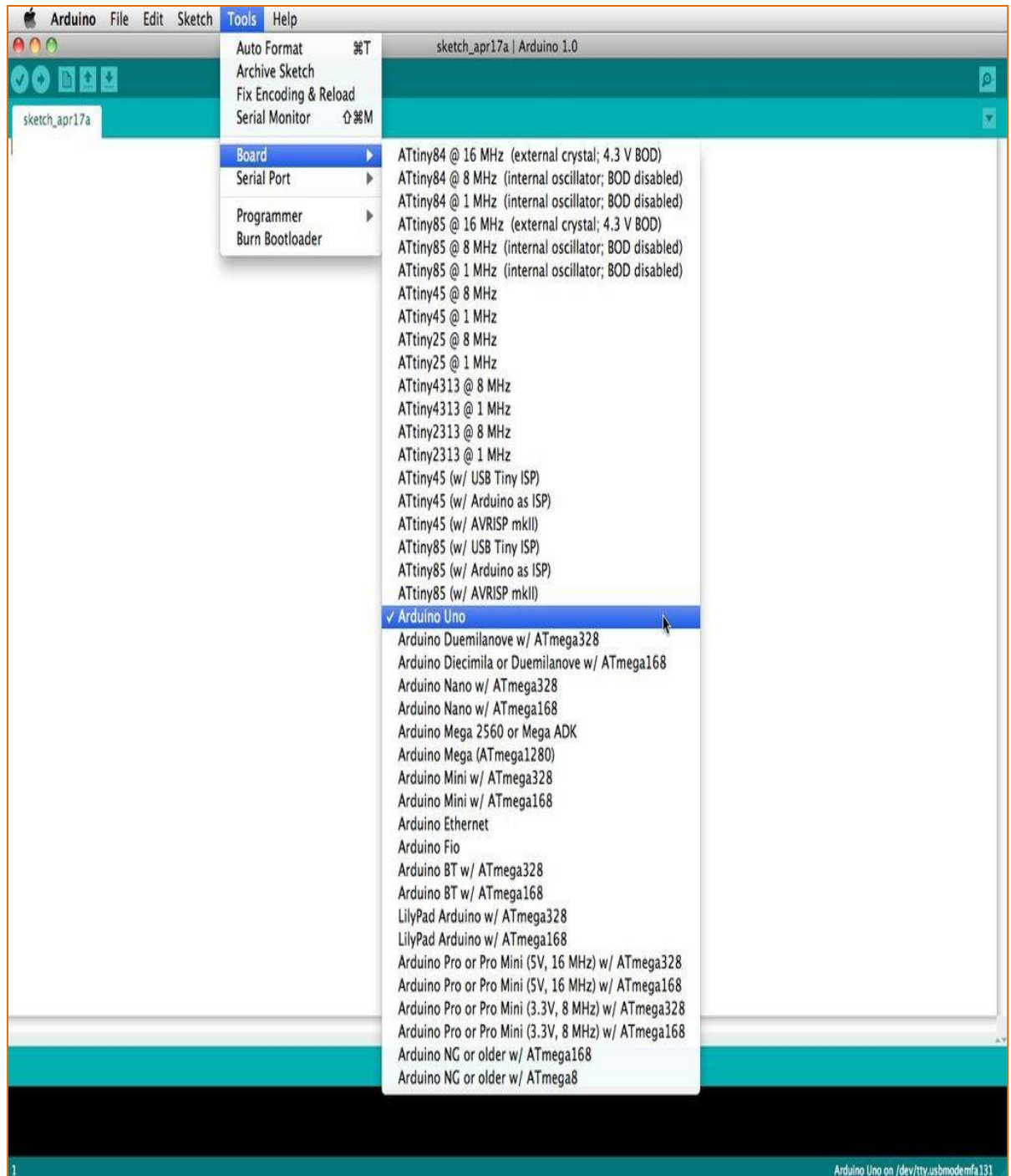


Fig 5.5.3 Settings

Before you can start doing anything in the Arduino programmer, you must set the board-type and serial port.

To set the board, go to the following:

Tools --> Boards

Select the version of board that you are using. Since I have an Arduino Uno plugged in, I obviously selected "Arduino Uno."

To set the serial port, go to the following:

Tools-->SerialPort

Select the serial port that looks like:

/dev/tty.usbmodem [random numbers]

Step 4: Run a Sketch

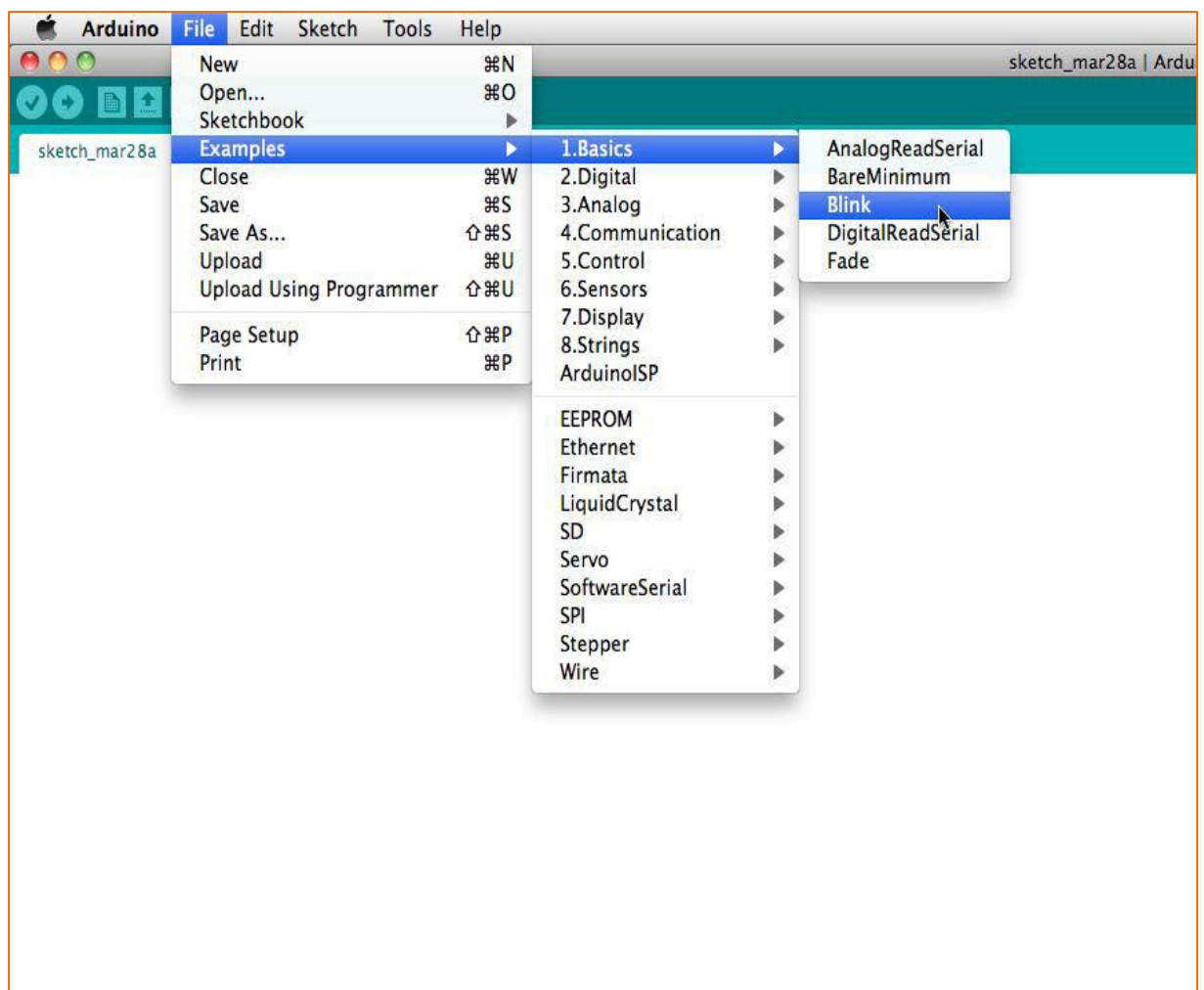


Fig 5.2.4 Sketch Running

Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something. To get the LED tied to digital pin 13 to blink on and off, let's load the blink example. The blink example can be found here:

Files --> Examples --> Basics --> Blink

The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.

Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.

Step 5: **Serial Monitor**

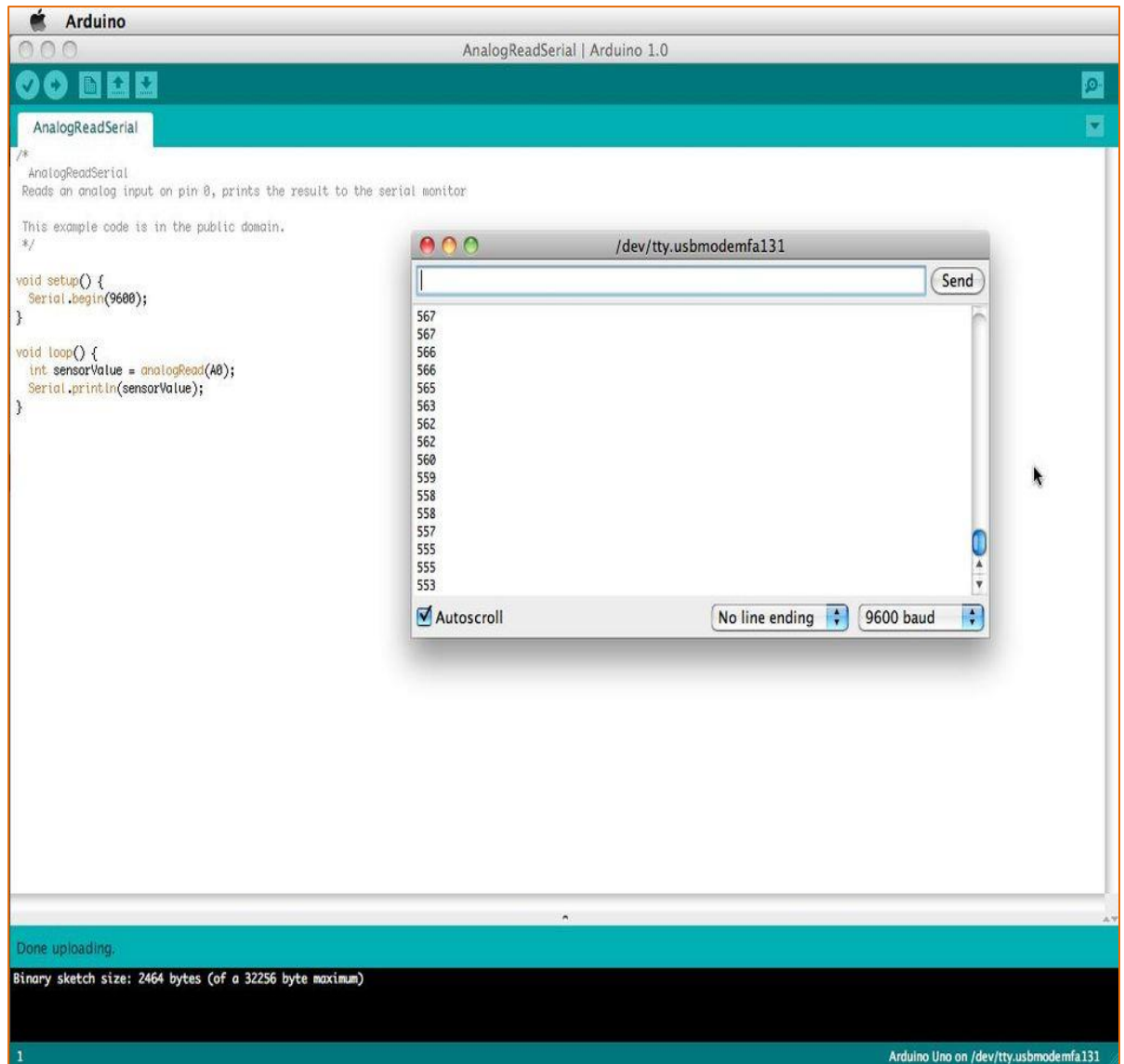


Fig 5.5.5 Serial Monitor

The serial monitor allows your computer to connect serially with the Arduino. This is important because it takes data that your Arduino is receiving from sensors and other devices and displays it in real-time on your computer. Having this ability is invaluable to debug your code and understand what number values the chip is actually receiving.

For instance, connect center sweep (middle pin) of a potentiometer to A0, and the outer pins, respectively, to 5v and ground. Next upload the sketch shown below:

File --> Examples --> 1.Basics --> AnalogReadSerial

Click the button to engage the serial monitor which looks like a magnifying glass. You can now see the numbers being read by the analog pin in the serial monitor. When you turn the knob the numbers will increase and decrease. The numbers will be between the range of 0 and 1023. The reason for this is that the analog pin is converting a voltage between 0 and 5V to a discrete number.

The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.

Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.

Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something.

To get the LED tied to digital pin 13 to blink on and off, let's load the blink example. The blink example can be found here:

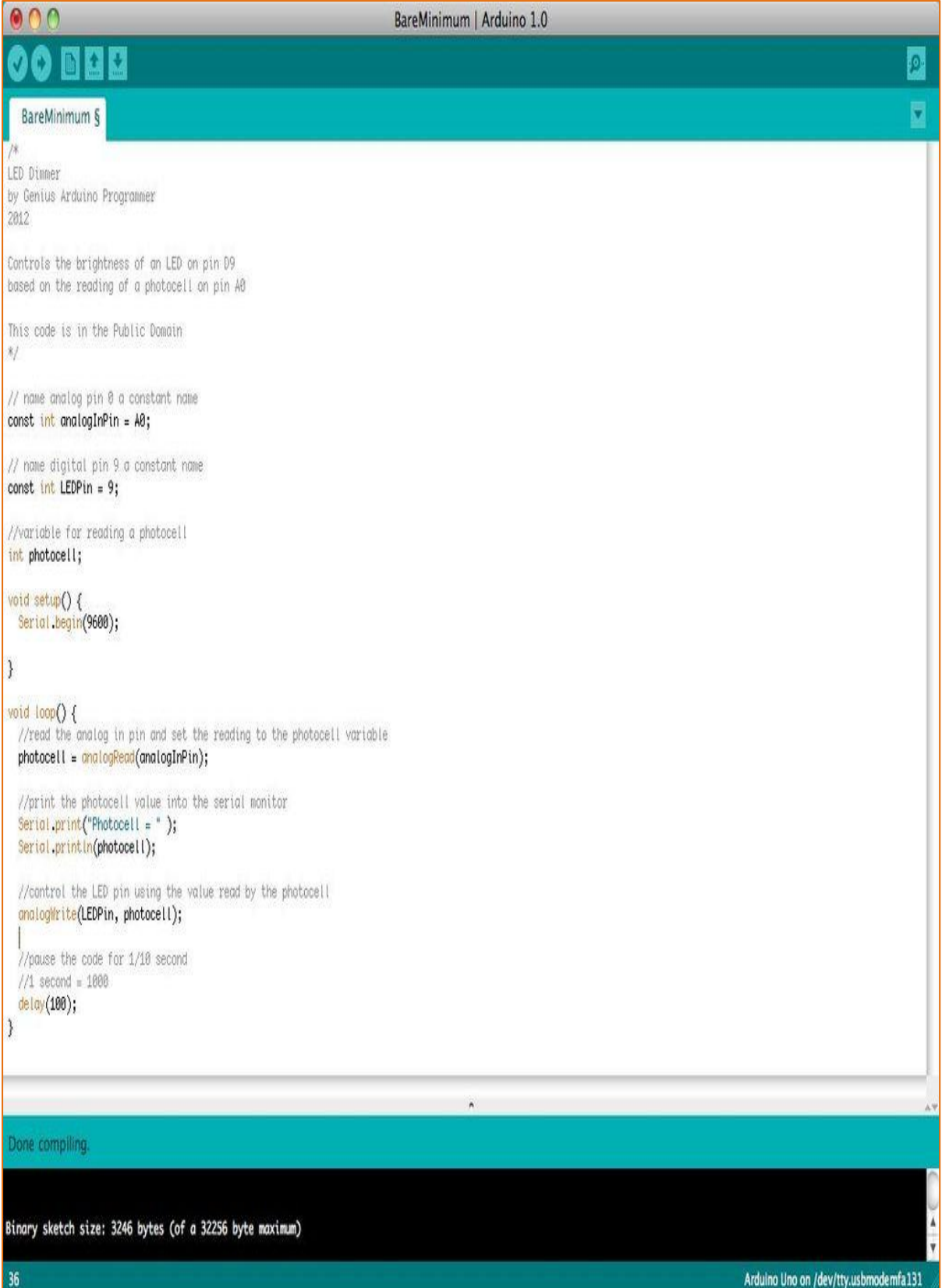
Files --> Examples --> Basics --> Blink

The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.

Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.

Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something.

Step 6: **Write Your Own Code**



```
BareMinimum | Arduino 1.0

BareMinimum 5

/*
LED Dimmer
by Genius Arduino Programmer
2012.

Controls the brightness of an LED on pin D9
based on the reading of a photocell on pin A0

This code is in the Public Domain
*/

// name analog pin 0 a constant name
const int analogInPin = A0;

// name digital pin 9 a constant name
const int LEDPin = 9;

//variable for reading a photocell
int photocell;

void setup() {
  Serial.begin(9600);
}

void loop() {
  //read the analog in pin and set the reading to the photocell variable
  photocell = analogRead(analogInPin);

  //print the photocell value into the serial monitor
  Serial.print("Photocell = ");
  Serial.println(photocell);

  //control the LED pin using the value read by the photocell
  analogWrite(LEDPin, photocell);
  //pause the code for 1/10 second
  //1 second = 1000
  delay(100);
}

Done compiling.

Binary sketch size: 3246 bytes (of a 32256 byte maximum)

36 Arduino Uno on /dev/tty.usbmodemfa131
```

Fig 5.5.6 Writing Code

To write your own code, you will need to learn some basic programming language syntax. In other words, you have to learn how to properly form the code for the programmer to understand it. You can think of this kind of like understanding grammar and punctuation. You can write an entire book without proper grammar and punctuation, but no one will be able to understand it, even if it is in English.

Some important things to keep in mind when writing your own code:

- An Arduino program is called a sketch.
- All code in an Arduino sketch is processed from top to bottom.
- Arduino sketches are typically broken into five parts.
 1. The sketch usually starts with a header that explains what the sketch is doing, and who wrote it.
 2. Next, it usually defines global variables. Often, this is where constant names are given to the different Arduino pins.
 3. After the initial variables are set, the Arduino begins the setup routine. In the setup function, we set initial conditions of variables when necessary, and run any preliminary code that we only want to run once. This is where serial communication is initiated, which is required for running the serial monitor.
 4. From the setup function, we go to the loop routine. This is the main routine of the sketch. This is not only where your main code goes, but it will be executed over and over, so long as the sketch continues to run.
 5. Below the loop routine, there is often other functions listed. These functions are user-defined and only activated when called in the setup and loop routine. When these functions are called, the Arduino processes all of the code in the function from top to bottom and then goes back to the next line in the sketch where it left off when the function was called. Functions are good because they allow you to run standard routines - over and over - without having to write the same lines of code over and over. You can simply call upon a function multiple times, and this will free up memory on the chip because the function routine is only written once. It also makes code easier to read. To learn how to form your own functions, check out [this page](#).

- All of that said, the only two parts of the sketch which are mandatory are the Setup and Loop routines.
- Code must be written in the Arduino Language, which is roughly based on C.
- Almost all statements written in the Arduino language must end with a ;
- Conditionals (such as if statements and for loops) do not need a ;
- Conditionals have their own rules and can be found under "Control Structures" on the Arduino Language page
- Variables are storage compartments for numbers. You can pass values into and out of variables. Variables must be defined (stated in the code) before they can be used and need to have a data type associated with it. To learn some of the basic data types, review the Language Page.

Okay! So let us say we want to write code that reads a photocell connected to pin A0, and use the reading we get from the photocell to control the brightness of an LED connected to pin D9.

First, we want to open the BareMinimum sketch, which can be found at:

File --> Examples --> 1.Basic --> BareMinimum

The BareMinimum Sketch should look like this:

```
<pre>void setup()

{

// put your setup code here, to run once:

}

void loop()

{

// put your main code here, to run repeatedly:
```

```
}
```

Next, lets put a header on the code, so other people know about what we are making, why, and under what terms

```
<pre>/*  
  
LED Dimmer  
  
by Genius Arduino Programmer  
  
Controls the brightness of an LED on pin D9  
  
based on the reading of a photocell on pin A0  
  
This code is in the Public Domain  
  
*/  
  
void setup()  
{  
  // put your setup code here, to run once:  
}  
  
void loop()  
{  
  // put your main code here, to run repeatedly:  
}
```

Once that is all squared away, let us define the pin names, and establish variables:

```
<pre>/*  
  
LED Dimmer  
  
by Genius Arduino Programmer
```


2012

Controls the brightness of an LED on pin D9

based on the reading of a photocell on pin A0

This code is in the Public Domain

```
*/  
  
// name analog pin 0 a constant name  
const int analogInPin = A0;  
  
// name digital pin 9 a constant name  
const int LEDPin = 9;  
  
//variable for reading a photocell  
int photocell;  
  
void setup()  
{  
  // put your setup code here, to run once:  
}  
  
void loop()  
{  
  // put your main code here, to run repeatedly:  
}
```

Now that variables and pin names are set, let us write the actual code:

```
<pre>/*
```

LED Dimmer

by Genius Arduino Programmer

2012

Controls the brightness of an LED on pin D9

based on the reading of a photocell on pin A0

This code is in the Public Domain

```
*/  
  
// name analog pin 0 a constant name  
  
const int analogInPin = A0;  
  
// name digital pin 9 a constant name  
  
const int LEDPin = 9;  
  
//variable for reading a photocell  
  
int photocell;  
  
void setup()  
  
{  
  
//nothing here right now  
  
}  
  
Void loop()  
  
{  
  
//read the analog in pin and set the reading to the photocell variable  
  
photocell = analogRead(analogInPin);  
  
//control the LED pin using the value read by the photocell  
  
AnalogWrite (LEDPin, photocell);  
  
//pause the code for 1/10 second  
  
//1 second = 1000  
  
delay (100);
```

```
}
```

If we want to see what numbers the analog pin is actually reading from the photocell, we will need to use the serial monitor. Let's activate the serial port and output those numbers:

```
<pre>/*
```

LED Dimmer

by Genius Arduino Programmer

2012

Controls the brightness of an LED on pin D9

based on the reading of a photocell on pin A0

This code is in the Public Domain

```
*/
```

```
// name analog pin 0 a constant name
```

```
const int analogInPin = A0;
```

```
// name digital pin 9 a constant name
```

```
const int LEDPin = 9;
```

```
//variable for reading a photocell
```

```
int photocell;
```

```
void setup()
```

```
{
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
  //read the analog in pin and set the reading to the photocell variable
```

```
  photocell = analogRead(analogInPin);
```

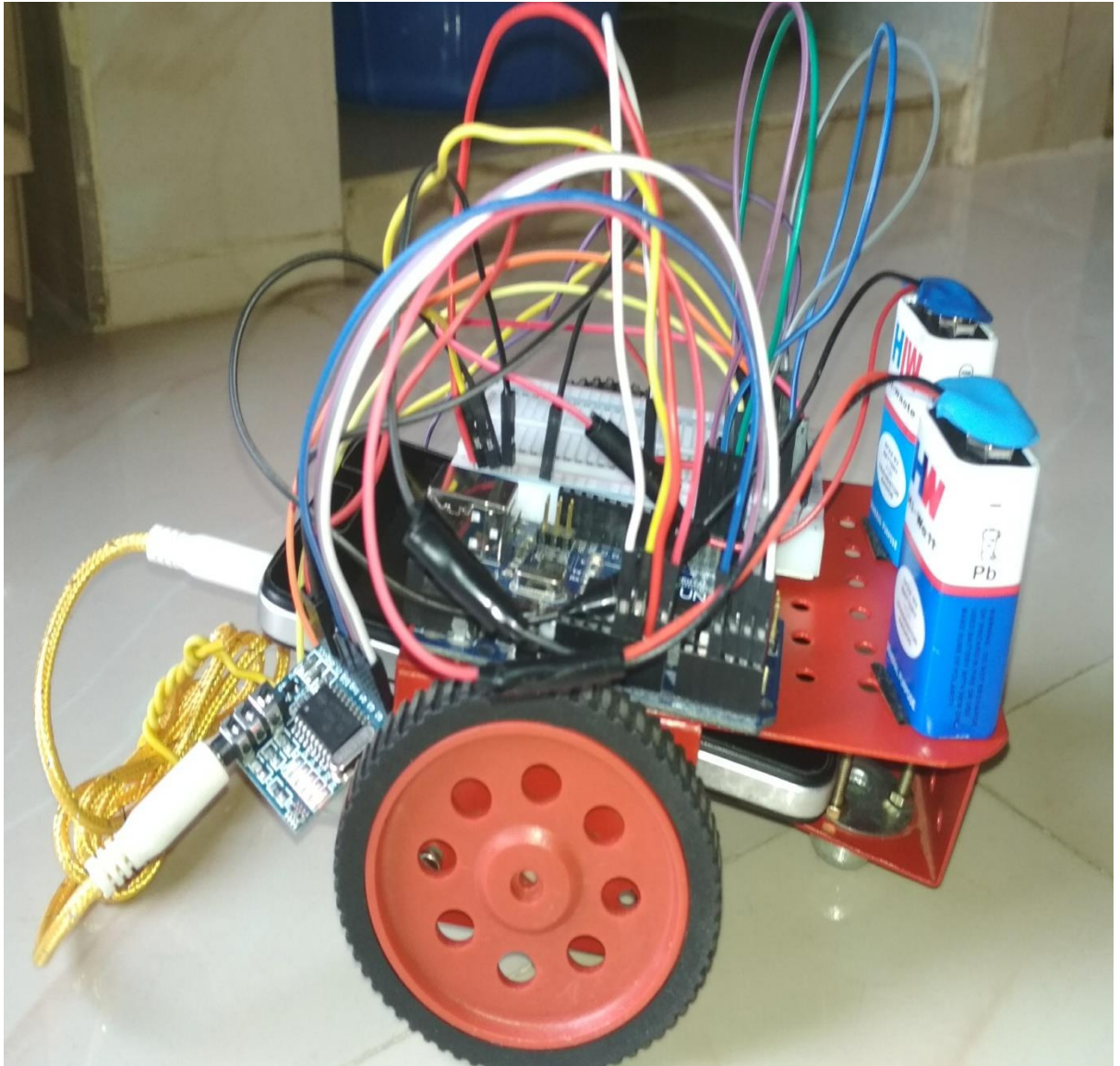
```
  //print the photocell value into the serial monitor
```

```
Serial.print ("Photocell = " );  
  
Serial.println (photocell);  
  
//control the LED pin using the value read by the photocell  
  
analogWrite(LEDPin, photocell);  
  
//pause the code for 1/10 second  
  
//1 second = 1000  
  
delay(100);
```

CHAPTER 6

RESULT

6.1 OUTPUT:



CHAPTER 7

ADVANTAGES AND APPLICATIONS

7.1 ADVANTAGES:

- This project uses mobile technology, due to which it has high range.
- Faster operating speed.
- No interference with other controllers.
- Low power requirement.
- Can be moved from one location to another location easily.
- It reduces man power.

7.2 APPLICATIONS:

- This robot can be used as rescue robot.
- This robot is small in size and can be used for spying.
- It can be used for surveillance.
- Can be used in military applications as bomb detectors.
- Can be used in search and rescue operations.

CHAPTER 8

CONCLUSION

8.1 CONCLUSION:

One mobile in your hand (cell 1) and other on the robot (cell 2). Cell 2 should be set to auto answer mode so that cell 2 automatically picks up the call. The earphone pin plug, MT8870 decoder is connected to cell 2. You make a call from cell 1 to cell 2 being in auto-answer mode picks up the call. Now keypad of cell 1 acts as remote. As any key is pressed on the cell 1, the corresponding 4-bit output appears on output pin of MT8870 which generates binary output and fed to arduino microcontroller which is programmed to accept the input and process it and send the output to the L283D motor driver, which gives input to the DC motors of the robot to move in particular direction as per the input of the user.

CHAPTER 9

FUTURE SCOPE

9.1 FUTURE SCOPE:

- Remote control vehicles like unmanned aerial vehicles in space exploration and military.
- IR sensors can be added to detect obstacles.
- Camera can be used to monitor surroundings.
- Metal detector and robotic arm can be connected to pick up and place objects.

REFERENCE

1. Raj Kamal – Microcontrollers Architecture, Programming, Interfacing and System Design.
2. C in depth – S.K. Srivastava.
3. Programming Arduino – Simon Monk
4. Arduino Robotics – josh Adams
5. Arduino uno Tutorials.
6. www.circuitdiggest.com
7. www.dtmfrobot/embeddedsource.com