

# **AUTONOMOUS ROBOTIC SYSTEM WITH GPS GUIDED INTERVENTIONS**

**A Project Report Submitted in partial fulfilment of the requirements for the award of**

**Bachelor of Technology in**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**Submitted By**

<b>BADAM A L MANI HARIKA</b>	<b>17011M2009</b>
<b>MEKAM ABHINAV SIDDHARTH</b>	<b>17011M2010</b>
<b>A ROHAN</b>	<b>17011M2007</b>
<b>JABBU SHIVA KUMAR YADAV</b>	<b>17011M2002</b>

**Under the Guidance of**

**Dr. P. Chandrasekhar Reddy**  
**Professor (ECE DEPT)**



Jawaharlal Nehru Technological University Hyderabad  
JNTUH College of Engineering, Hyderabad Department of  
Electronics and Communication Engineering, Kukatpally,  
Hyderabad – 500085

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**  
**HYDERABAD**

JNTUH College of Engineering, Hyderabad (Autonomous)  
(Kukatpally, Hyderabad - 500085)  
Department of Electronics and Communication Engineering



*DECLARATION BY THE SUPERVISOR*

This is to certify that the project entitled, “**AUTONOMOUS ROBOTIC SYSTEM WITH GPS GUIDED INTERVENTIONS**” submitted by

**BADAM A L MANI HARIKA**

**17011M2009**

**MEKAM ABHINAV SIDDHARTH**

**17011M2010**

**A ROHAN**

**17011M2007**

**JABBU SHIVA KUMAR YADAV**

**17011M2002**

In partial fulfilment of the requirements for the award of major project in Electronics and Communication Engineering at Jawaharlal Nehru Technological University Hyderabad during the academic year 2020 -2021, is an authentic work carried out by them under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree or Diploma.

**PROJECT SUPERVISOR**  
**Dr. P. Chandrasekhar Reddy**  
**Professor (ECE DEPT)**

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**  
**HYDERABAD**

**JNTUH College of Engineering, Hyderabad (Autonomous)**

**(Kukatpally, Hyderabad - 500085)**

**Department of Electronics and Communication Engineering**



*CERTIFICATE BY THE HEAD OF THE DEPARTMENT*

This is to certify that the project entitled, “ **AUTONOMOUS ROBOTIC SYSTEM WITH GPS GUIDED INTERVENTIONS** ” submitted by

**BADAM A L MANI HARIKA**

**17011M2009**

**MEKAM ABHINAV SIDDHARTH**

**17011M2010**

**A ROHAN**

**17011M2007**

**JABBU SHIVA KUMAR YADAV**

**17011M2002**

In partial fulfilment of the requirements for the award of major project in Electronics and Communication Engineering at Jawaharlal Nehru Technological University Hyderabad during the academic year 2020 -2021, is an authentic work carried out by them under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree or Diploma.

**HEAD OF DEPARTMENT**

**DR. K. ANITHA SHEELA**

PROFESSOR AND HEAD,

Department of ECE, JNTUHCEH

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY**  
**HYDERABAD**

**JNTUH College of Engineering, Hyderabad (Autonomous)**

**(Kukatpally, Hyderabad - 500085)**

**Department of Electronics and Communication Engineering**



***DECLARATION BY THE CANDIDATES***

We hereby declare that the major project entitled “ **AUTONOMOUS ROBOTIC SYSTEM WITH GPS GUIDED INTERVENTIONS** ” is a bona fide record work done and submitted under the esteemed guidance of **Dr. P CHANDRASEKHAR REDDY**, Professor, Department of ECE, JNTU, Hyderabad, in partial fulfilment of the requirements for the major project in Electronics and Communication Engineering at the Jawaharlal Nehru Technological University Hyderabad during the academic year 2020 -2021. This record is a bonafide work carried out by us and the results kept in the major project have not been reproduced or copied.

The results in the major project have not been submitted in any other university or institution for the award of degree or diploma.

**BADAM A L MANI HARIKA**

**17011M2009**

**MEKAM ABHINAV SIDDHARTH**

**17011M2010**

**A ROHAN**

**17011M2007**

**JABBU SHIVA KUMAR YADAV**

**17011M2002**

## **ACKNOWLEDGEMENTS**

We would like to express our sincere deep appreciation and indebtedness particularly to our guide **Dr. P CHANDRASEKHAR REDDY** , Professor & **DR.K. ANITHA SHEELA** Professor and Head, Electronics & Communication Engineering, for their endless support, kind and understanding spirit.

The completion of this undertaking could not have been possible without the participation and assistance of so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. I also thank my relatives, friends and others who shared their support, morally, financially, or physically. Above all, I thank the Great Almighty, the author of knowledge and wisdom, for his countless love.

**BADAM A L MANI HARIKA**

**17011M2009**

**MEKAM ABHINAV SIDDHARTH**

**17011M2010**

**A ROHAN**

**17011M2007**

**JABBU SHIVA KUMAR YADAV**

**17011M2002**



# **GPS GUIDED ROBOTIC SYSTEM**

## **1.1 Introduction:**

In this project, the design methodology and implementation of a low cost autonomous robot prototype that can reach prescribed destination using Global Positioning System (GPS). By calculating the heading angle from the current GPS co-ordinate, the robot can adjust its direction using a digital compass reading. Our designed autonomous robot can carry and transport small objects perfectly to the target destination and after fulfilling its task the robot can return independently to its starting location. The functionality and accuracy of this GPS guided transporter robot are tested in different locations and accuracy rate is measured in terms of heading angle and average distance deviation from the target. Our robotic module can be very effective as a surveillance robot in the management of a disaster such as earthquake. Apart from its scope of use in industrial automation, our designed object transporter robot can be a helping hand for the disabled people, therefore offering them flexibility and comfort in day-to-day life.

## **1.2 Objective:**

The main objective of this project is to build a modular mobile robot that can navigate around its way using GPS. The primary component of this robot is its navigation system, with an augmented GPS system integrated to provide the local coordinates and help in positioning of the robot. It can be programmed to carryout multiple functionalities.

## **1.3 Future scope:**

This proposed project has the potential to automate federally provided robotic control navigation systems for the future and will be augmented and improved to satisfy future military and civil requirements for accuracy, coverage, availability, continuity, and integrity in terms of automation and management systems.



## **1.4 Technical Approach:**

### **1.4.1 Hardware description**

- ROBOTIC CHASSIS
- ARDUINO MEGA
- L293D ADAFRUIT OR COMPATIBLE MOTOR CONTROLLER
- CELLPHONE OR TABLET
- HC-06 BLUETOOTH MODULE
- HMC5883L MAGNETOMETER/ COMPASS
- UBLOX NEO 6M
- JUMPER WIRES

### **1.4.2 SOFTWARE REQUIREMENT:**

- UNO IDE for Programming.

## **1.5 OUT LINE OF THESIS**

In this, Description of hardware components and its functionalities are discussed in MODULE-2. Implementation of project is discussed in MODULE-3. MODULE-3 consists of Block diagram, system hardware, principle of operation, Results and conclusions are discussed in the MODULE- 4

**MODULE- 2**

**HARDWARE**

**COMPONENT**

**DESCRIPTION**

## 2.1 ARDUINO MEGA

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), 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 Mega is compatible with most shields designed for the Arduino

The Arduino MEGA is frequently used microcontroller board in the family of an Arduino. This is the latest third version of an Arduino board and released in the year 2011. The main advantage of this board is if we make a mistake, we can change the microcontroller on the board. The main features of this board mainly include, it is available in DIP (dual-inline-package), detachable and ATmega328 microcontroller. The programming of this board can easily be loaded by using an Arduino computer program. This board has huge support from the Arduino community, which will make a very simple way to start working in embedded electronics, and many more applications. Please refer the link to know about Arduino – Basics, and Design.

The Mega 2560 is an update to the Arduino Mega, which it replaces.



**Figure: ARDUINO MEGA FRONT END**

## **Specifications:**

Microcontroller      ATmega2560

Operating Voltage      5V

Input Voltage (recommended) 7-12V

Input Voltage (limits) 6-20V

Digital I/O Pins      54 (of which 14 provide PWM output)

Analog Input Pins      16

DC Current per I/O Pin      40 mA

DC Current for 3.3V Pin      50 mA

Flash Memory      256 KB of which 8 KB used by bootloader

SRAM      8 KB

EEPROM      4 KB

Clock Speed      16 MHz

## **Power**

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically. 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 center-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 Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the ATmega16U2 (ATmega8U2 in the revision 1 and revision 2 boards) programmed as a USB-to-serial converter.

Revision 2 of the Mega2560 board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode. Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

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 50 mA.

GND. Ground pins.

## **Memory**

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

## Input and Output

Each of the 54 digital pins on the Mega 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 40 mA and has an internal pull-up resistor (disconnected by default) of 2050 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.

- External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt

2). 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: 0 to 13. Provide 8-bit PWM output with the `analogWrite()` function.

- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

- TWI: 20 (SDA) and 21 (SCL). Support TWI communication using the Wire library. Note that these pins are not in the same location as the TWI pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and `analogReference()` function. 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 Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The

ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega16U2 (ATmega 8U2 on the revision 1 and revision 2 boards) on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2/ATmega16U2 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 Mega2560's digital pins.

The ATmega2560 also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library.

## **Programming**

The Arduino Mega can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The ATmega2560 on the Arduino Mega 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 ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available in the Arduino repository. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode. 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 Mega2560 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 ATmega2560 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 Mega2560 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 Mega2560. 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 Mega2560 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 reenable 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 Mega2560 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 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

## **Physical Characteristics and Shield Compatibility**

The maximum length and width of the Mega2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three 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.

The Mega2560 is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the

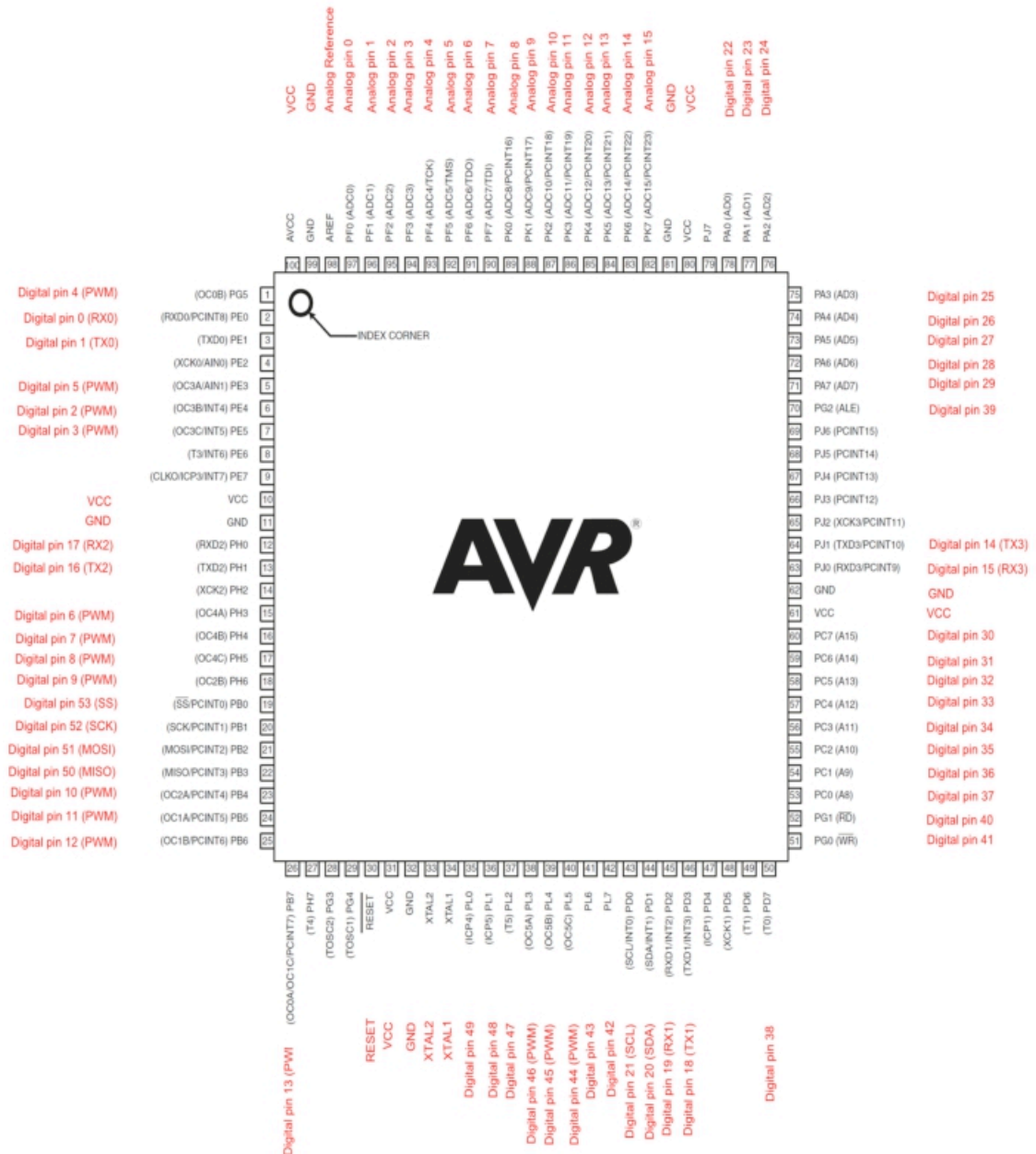
ICSP header on both the Mega2560 and Duemilanove / Diecimila. Please note that I2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).



## ATmega2560-Arduino Pin Mapping:

Below is the pin mapping for the Atmega2560. The chip used in Arduino 2560.

Arduino Mega 2560 PIN diagram



Arduino Mega 2560 PIN mapping table

Pin Number	Pin Name	Mapped Pin Name
1	PG5 ( OC0B )	Digital pin 4 (PWM)
2	PE0 ( RXD0/PCINT8 )	Digital pin 0 (RX0)

3	PE1 ( TXD0 )	Digital pin 1 (TX0)
4	PE2 ( XCK0/AIN0 )	
5	PE3 ( OC3A/AIN1 )	Digital pin 5 (PWM)
6	PE4 ( OC3B/INT4 )	Digital pin 2 (PWM)
7	PE5 ( OC3C/INT5 )	Digital pin 3 (PWM)
8	PE6 ( T3/INT6 )	
9	PE7 ( CLK0/ICP3/INT7 )	
10	VCC	VCC
11	GND	GND
12	PH0 ( RXD2 )	Digital pin 17 (RX2)
13	PH1 ( TXD2 )	Digital pin 16 (TX2)
14	PH2 ( XCK2 )	
15	PH3 ( OC4A )	Digital pin 6 (PWM)
16	PH4 ( OC4B )	Digital pin 7 (PWM)
17	PH5 ( OC4C )	Digital pin 8 (PWM)
18	PH6 ( OC2B )	Digital pin 9 (PWM)
19	PB0 ( SS/PCINT0 )	Digital pin 53 (SS)
20	PB1 ( SCK/PCINT1 )	Digital pin 52 (SCK)
21	PB2 ( MOSI/PCINT2 )	Digital pin 51 (MOSI)
22	PB3 ( MISO/PCINT3 )	Digital pin 50 (MISO)
23	PB4 ( OC2A/PCINT4 )	Digital pin 10 (PWM)
24	PB5 ( OC1A/PCINT5 )	Digital pin 11 (PWM)
25	PB6 ( OC1B/PCINT6 )	Digital pin 12 (PWM)
26	PB7 ( OC0A/OC1C/PCINT7 )	Digital pin 13 (PWM)
27	PH7 ( T4 )	
28	PG3 ( TOSC2 )	
29	PG4 ( TOSC1 )	
30	RESET	RESET
31	VCC	VCC
32	GND	GND
33	XTAL2	XTAL2
34	XTAL1	XTAL1
35	PL0 ( ICP4 )	Digital pin 49
36	PL1 ( ICP5 )	Digital pin 48
37	PL2 ( T5 )	Digital pin 47
38	PL3 ( OC5A )	Digital pin 46 (PWM)
39	PL4 ( OC5B )	Digital pin 45 (PWM)
40	PL5 ( OC5C )	Digital pin 44 (PWM)
41	PL6	Digital pin 43

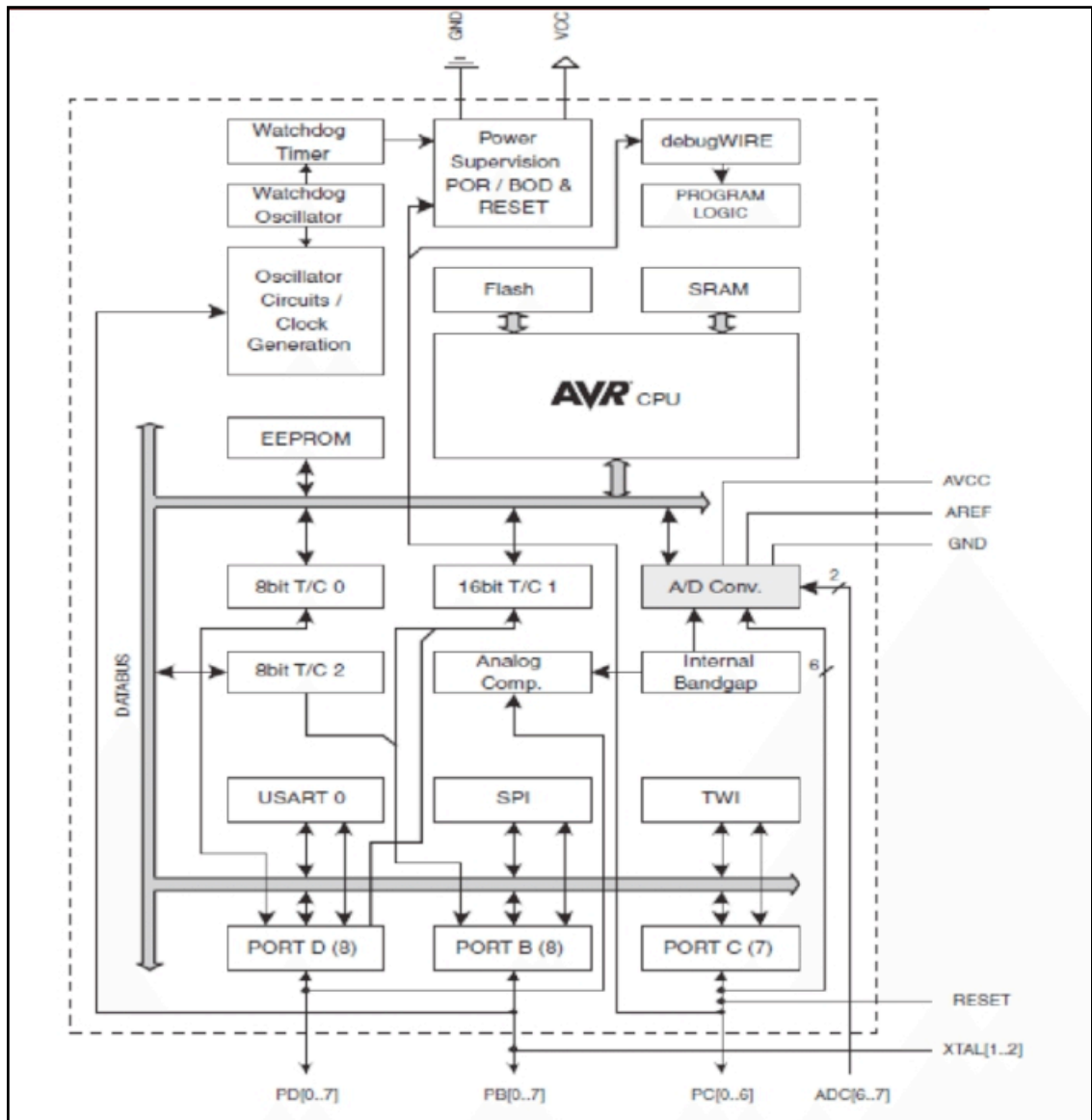
42	PL7	Digital pin 42
43	PD0 ( SCL/INT0 )	Digital pin 21 (SCL)
44	PD1 ( SDA/INT1 )	Digital pin 20 (SDA)
45	PD2 ( RXDI/INT2 )	Digital pin 19 (RX1)
46	PD3 ( TXD1/INT3 )	Digital pin 18 (TX1)
47	PD4 ( ICP1 )	
48	PD5 ( XCK1 )	
49	PD6 ( T1 )	
50	PD7 ( T0 )	Digital pin 38
51	PG0 ( WR )	Digital pin 41
52	PG1 ( RD )	Digital pin 40
53	PC0 ( A8 )	Digital pin 37
54	PC1 ( A9 )	Digital pin 36
55	PC2 ( A10 )	Digital pin 35
56	PC3 ( A11 )	Digital pin 34
57	PC4 ( A12 )	Digital pin 33
58	PC5 ( A13 )	Digital pin 32
59	PC6 ( A14 )	Digital pin 31
60	PC7 ( A15 )	Digital pin 30
61	VCC	VCC
62	GND	GND
63	PJ0 ( RXD3/PCINT9 )	Digital pin 15 (RX3)
64	PJ1 ( TXD3/PCINT10 )	Digital pin 14 (TX3)
65	PJ2 ( XCK3/PCINT11 )	
66	PJ3 ( PCINT12 )	
67	PJ4 ( PCINT13 )	
68	PJ5 ( PCINT14 )	
69	PJ6 ( PCINT 15 )	
70	PG2 ( ALE )	Digital pin 39
71	PA7 ( AD7 )	Digital pin 29
72	PA6 ( AD6 )	Digital pin 28
73	PA5 ( AD5 )	Digital pin 27
74	PA4 ( AD4 )	Digital pin 26
75	PA3 ( AD3 )	Digital pin 25
76	PA2 ( AD2 )	Digital pin 24
77	PA1 ( AD1 )	Digital pin 23
78	PA0 ( AD0 )	Digital pin 22
79	PJ7	
80	VCC	VCC

81	GND	GND
82	PK7 ( ADC15/PCINT23 )	Analog pin 15
83	PK6 ( ADC14/PCINT22 )	Analog pin 14
84	PK5 ( ADC13/PCINT21 )	Analog pin 13
85	PK4 ( ADC12/PCINT20 )	Analog pin 12
86	PK3 ( ADC11/PCINT19 )	Analog pin 11
87	PK2 ( ADC10/PCINT18 )	Analog pin 10
88	PK1 ( ADC9/PCINT17 )	Analog pin 9
89	PK0 ( ADC8/PCINT16 )	Analog pin 8
90	PF7 ( ADC7/TDI )	Analog pin 7
91	PF6 ( ADC6/TDO )	Analog pin 6
92	PF5 ( ADC5/TMS )	Analog pin 5
93	PF4 ( ADC4/TCK )	Analog pin 4
94	PF3 ( ADC3 )	Analog pin 3
95	PF2 ( ADC2 )	Analog pin 2
96	PF1 ( ADC1 )	Analog pin 1
97	PF0 ( ADC0 )	Analog pin 0
98	AREF	Analog Reference
99	GND	GND
100	AVCC	VCC

## 2.2 Block Diagram:

The ARDUINO MEGA 2560 uses a 32bit processor with 16 bit instructions. It is Harvard architecture which mostly means that instruction memory and data memory are completely separate.

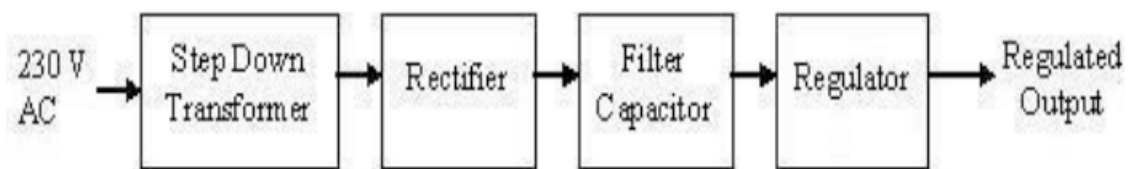
The ARDUINO MEGA 2560 has on die program Read-Only Memory (ROM) which includes some library code and a first stage boot loader. All the rest of the code must be stored in external Serial flash memory (provides only serial access to the data - rather than addressing individual bytes, the user reads or writes large contiguous groups of bytes in the address space serially). Depending on your ARDUINO MEGA 2560, the amount of available flash memory can vary.



**Figure2.2 : Block diagram of MICROCONTROLLER ARDUINO MEGA 2560**

### 2.3 POWERSUPPLY:

All digital circuits require regulated DC voltage to operate. Regulated Power supply is a device, which converts, regulates, and transmits the required power to the circuit to be operated in given figure process.



**Figure2.4: Basic block diagram of a fixed regulated power supply**

## Elements of power supply

### 2.3.1 Transformer

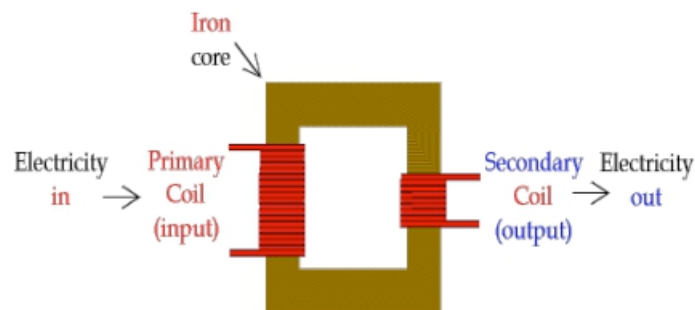
### 2.3.2 Rectifier

### 2.3.3 Filter

### 2.3.4 Regulator

#### 2.3.1 Transformer:

The AC line voltage available for commercial purpose is not suitable for electronic circuits. Most of the electronic circuits require a considerably lower voltage. The transformer is a device used to convert the ac line voltage to a voltage level more appropriate to the needs of the circuit to be operated. At the same time, the transformer provides electrical isolation between the ac line and the circuit to be operated. This is an important safety consideration.



**Figure 2.3.1: Transformer**

A transformer consists of two coils also called as “WINDINGS” namely PRIMARY & SECONDARY. They are linked together through inductively coupled electrical conductors also called as CORE. If load is applied to the secondary then an alternating current will flow through the load.

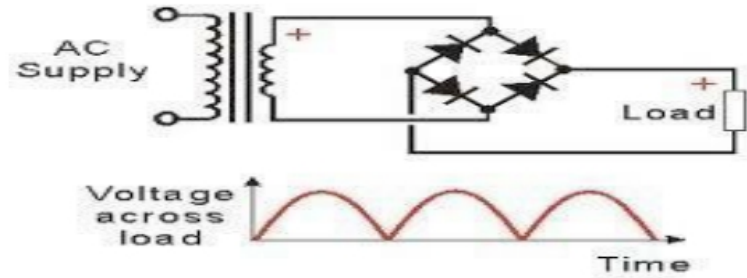
#### 2.3.2 Rectifier:

A rectifier is a device that converts an AC signal into DC signal. For rectification purpose we use a diode, a diode is a device that allows current to pass only in one direction i.e. When the anode of the diode is positive with respect to the cathode also

called as forward biased condition & blocks current in the reversed biased condition. Rectifier can be classified as

1. Half wave Rectifier
2. Full wave Rectifier

## 2.4 Bridge Rectifier



**Figure 2.4: Bridge Rectifier and its Output**

Full Bridge Wave Rectifier consists of four diodes namely D1, D2, D3 and D4. During the positive half cycle diodes D1 & D4 conduct whereas in the negative

half cycle diodes D2 & D3 conduct thus the diodes keep switching the transformer connections so we get positive half cycles in the output. If we use a center tapped transformer for a bridge rectifier we can get both positive & negative half cycles which can thus be used for generating fixed positive & fixed negative voltages.

**Table 2.4: Rectifiers Performance Parameters**

Parameter	Half wave rectifier	Full wave rectifier
Ripple factor	1.211	0.482
Efficiency	40.6%	81.2
TUF	0.21	0.693

### 2.4.1 FilterCapacitor

Even though half wave & full wave rectifier give DC output, none of them provides a constant output voltage. For this we require to smoothen the waveform received from the rectifier. This can be done by using a capacitor at the output of

The rectifier this capacitor is also called as “FILTER CAPACITOR” or “RESERVOIR CAPACITOR”.

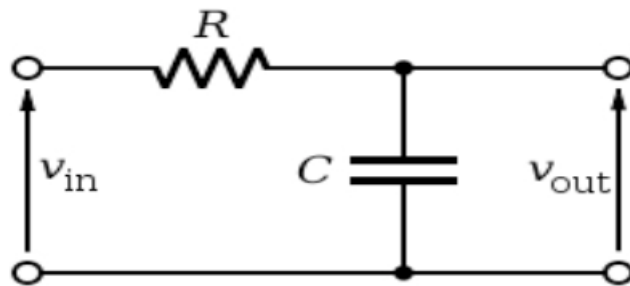


Figure 2.5.1: Filter Circuit

We place the Filter Capacitor at the output of the rectifier the capacitor will charge to the peak voltage during each half cycle then will discharge its stored energy slowly through the load while the rectified voltage drops to zero, thus trying to keep the voltage as constant as possible.

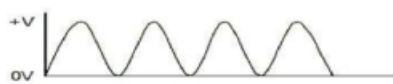


Figure 2.4.1: (i) Input waveform



(ii) Filter output

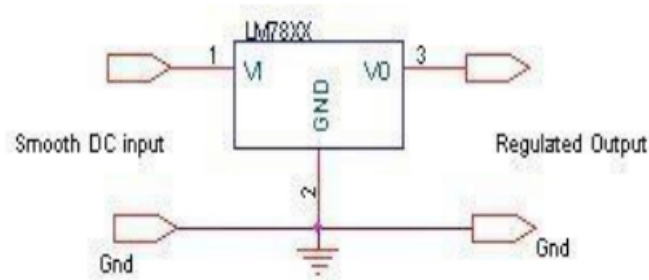
### 2.5 Voltage Regulator:

A Voltage regulator is a device which converts varying input voltage into a constant regulated output voltage. Voltage regulator can be of two types

**1.Linear Voltage Regulator** also called as Resistive Voltage regulator because they dissipate the excessive voltage resistively as heat.



**2.Switching Regulators** they regulate the output voltage by switching the Current ON/OFF very rapidly. Since their output is either ON or OFF it dissipates very low power thus achieving higher efficiency as compared to linear voltage regulators.



**Figure 2.5: Circuit Diagram IC 7805**

IC 7805 is an integrated three-terminal positive fixed linear voltage regulator. It supports an input voltage of 10 volts to 35 volts and output voltage of 5 volts. The last two digits represent the voltage. The 78xx series of regulators is designed to work in complement with the 79xx series of negative voltage regulators in systems that provide both positive and negative regulated voltages, since the 78xx series can't regulate negative voltages in such a system.

**Table 2.6: 7805 specifications**

SPECIFICATIONS	IC 7805
$V_{out}$	5V
$V_{in} - V_{out}$ Difference	5V – 20V
Operation Ambient Temp	0 – 125°C
Output $I_{max}$	1A

## 2.6 REGULATOR:

### 2.6.1 7805:

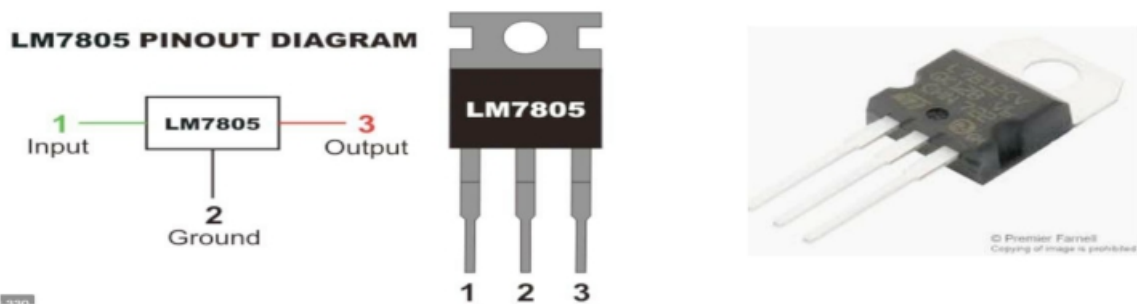
7805 is a three terminal linear voltage regulator IC with a fixed output voltage of 5V which is useful in a wide range of applications. Currently, the 7805 Voltage Regulator IC is manufactured by Texas Instruments, ON Semiconductor, STMicroelectronics, Diodes incorporated, Infineon Technologies, etc. They are available in several IC Packages like TO-220, SOT-223, TO-263.

### Features:

- It can deliver up to 1.5 A of current (with heat sink).
- Has both internal current limiting and thermal shutdown features.
- 3-Terminal Regulators.

### Applications:

- Fixed-Output Regulator
- Positive Regulator in Negative Configuration
- Adjustable Output Regulator
- Current Regulator
- Adjustable DC Voltage Regulator
- Regulated Dual-Supply



**Figure 2.7.1(i): Circuit Diagram IC 7805****Figure 2.7.1(ii): Circuit Diagram IC 7812**

### 2.6.2 7812:

7812 is a 12V Voltage Regulator that restricts the voltage output to 12V and draws 12V regulated power supply. The 7812 is the most common, as its regulated 12-volt supply provides a convenient power source for most TTL components.

### Features:

- Fixed 12V voltage regulators
- Thermal overload protection
- Short circuit protection
- Output transition SOA protection
- Heatsink is required

## **Specifications:**

- Output Type: Fixed
- Output Voltage: +12V DC
- Current Output: up to 1.5A
- Input Voltage: 14.6 - 35VDC
- Quiescent (standby) current: 8mA
- Dropout Voltage (Max): 2V @ 1A
- Polarity: Positive
- Operating Temperature: 0 to +125°C

## **2.7: L293D MOTOR DRIVER**

### **What is a Motor Driver Module?**

#### Motor Drivers

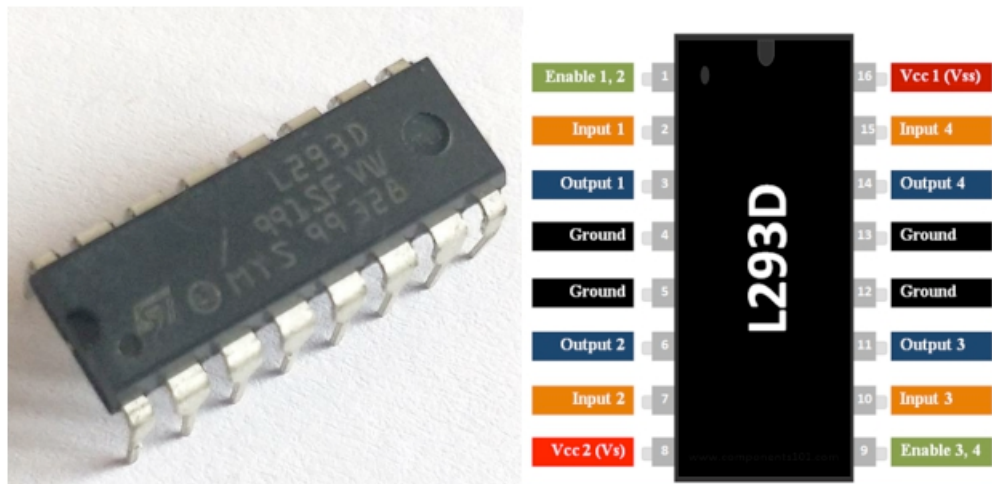
A motor driver module is a simple circuit used for controlling a DC motor. It is commonly used in autonomous robots and RC cars (L2938N and L293D are the most regularly utilized motor driver chips). A motor driver module takes the low voltage input from a controller like Arduino. This input logic controls the direction of DC motors connected to the driver. To put it in simple words, you can control the direction of DC motors by giving appropriate logic to the motor driver module.

The motor driver module consists of a motor driver IC, which is the heart of the module. The IC alone can control the DC motor but using the module makes the interfacing with Arduino easy.

### **Why do we need a motor driver module?**

All microcontrollers operate on low-level voltage/current signals, unlike motors. For instance, the Arduino or PIC microcontroller can output a maximum voltage of 5V or 3.3V. But a decent DC motor needs at least 5V or 12V. Also, the output current limit of Arduino is relatively very low.

Hence the output of Arduino is not enough to power up the motors. To solve this problem the use of a motor driver is essential. We bridge the gap between the Arduino and motor by introducing a motor driver between them. And to supply the voltage/current required to operate the motor, an external supply is connected to the motor driver module.



L293D Motor Driver IC  
Motor Driver IC L293D Pinout

## L293D Pin Configuration

Pin Number	Pin Name	Description
1	Enable 1,2	This pin enables the input pin Input 1(2) and Input 2(7)
2	Input 1	Directly controls the Output 1 pin. Controlled by digital circuits
3	Output 1	Connected to one end of Motor 1
4	Ground	Ground pins are connected to ground of circuit (0V)
5	Ground	Ground pins are connected to ground of circuit (0V)
6	Output 2	Connected to another end of Motor 1
7	Input 2	Directly controls the Output 2 pin. Controlled by digital circuits
8	Vcc2 (Vs)	Connected to Voltage pin for running motors (4.5V to 36V)
9	Enable 3,4	This pin enables the input pin Input 3(10) and Input 4(15)
10	Input 3	Directly controls the Output 3 pin. Controlled by digital circuits
11	Output 3	Connected to one end of Motor 2
12	Ground	Ground pins are connected to ground of circuit (0V)
13	Ground	Ground pins are connected to ground of circuit (0V)
14	Output 4	Connected to another end of Motor 2
15	Input 4	Directly controls the Output 4 pin. Controlled by digital circuits

16	VCC2 (VSS)	Connected to +5V to enable IC function.
----	------------	---

## Features

- Can be used to run Two DC motors with the same IC.
- Speed and Direction control is possible
- Motor voltage Vcc2 (Vs): 4.5V to 36V
- Maximum Peak motor current: 1.2A
- Maximum Continuous Motor Current: 600mA
- Supply Voltage to Vcc1(vss): 4.5V to 7V
- Transition time: 300ns (at 5V and 24V)
- Automatic Thermal shutdown is available
- Available in 16-pin DIP, TSSOP, SOIC packages

## L293D Equivalent Dual Timer IC

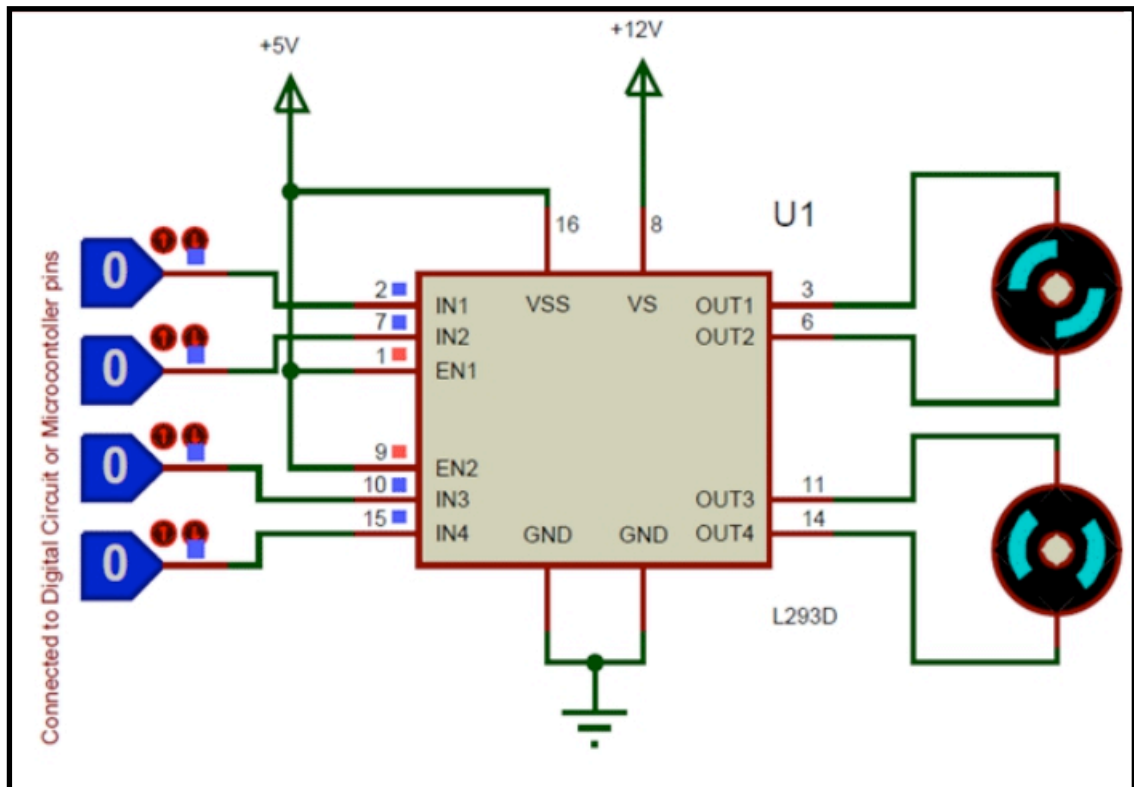
LB1909MC, SN754410, ULN2003

## Where to use L293D IC

The L293D is a popular 16-Pin **Motor Driver IC**. As the name suggests it is mainly used to drive motors. A single **L293D IC** is capable of running two DC motors at the same time; also the direction of these two motors can be controlled independently. So if you have motors which has operating voltage less than 36V and operating current less than 600mA, which are to be controlled by digital circuits like Op-Amp, 555 timers, digital gates or even Micro controllers like Arduino, PIC, ARM etc.. this IC will be the right choice for you.

## How to use a L293D Motor Driver IC

**Using this L293D motor driver IC** is very simple. The IC works on the principle of **Half H-Bridge**, let us not go too deep into what H-Bridge means, but for now just know that H bridge is a set up which is used to run motors both in clock wise and anti clockwise direction. As said earlier this IC is capable of running two motors at the any direction at the same time, the circuit to achieve the same is shown below.



All the Ground pins should be grounded. There are two power pins for this IC, one is the Vss(Vcc1) which provides the voltage for the IC to work, this must be connected to +5V. The other is Vs(Vcc2) which provides voltage for the motors to run, based on the specification of your motor you can connect this pin to anywhere between 4.5V to 36V, here I have connected to +12V.

The Enable pins (Enable 1,2 and Enable 3,4) are used to Enable Input pins for Motor 1 and Motor 2 respectively. Since in most cases we will be using both the motors both the pins are held high by default by connecting to +5V supply. The input pins Input 1,2 are used to control the motor 1 and Input pins 3,4 are used to control the Motor 2. The input pins are connected to the any Digital circuit or microcontroller to control the speed and direction of the motor. You can toggle the input pins based on the following table to control your motor.

Input 1 = HIGH(5v)	Output 1 = HIGH	Motor 1 rotates in Clock wise Direction
Input 2 = LOW(0v)	Output 2 = LOW	
Input 3 = HIGH(5v)	Output 1 = HIGH	Motor 2 rotates in Clock wise Direction
Input 4 = LOW(0v)	Output 2 = LOW	

Input 1 = LOW(0v)	Output 1 = LOW	Motor 1 rotates in Anti-Clock wise Direction
Input 2 = HIGH(5v)	Output 2 = HIGH	



# **MODULE-3**

## **IMPLEMENTATION**



### **3.1 Introduction:**

This project have been designed and implemented an autonomous object transporter robot which can operate independently using GPS data and carry any object with it. Our designed prototype can deliver small objects successfully and automatically return to its starting point. GPS receiver module is used to capture the GPS signal by locking with satellites. However, limitations in satellite signal reception limit the use of GPS systems for navigation . These limitations can be overcome by utilizing differential GPS but at an additional expense . A digital compass continuously reads current heading angle of robot using magnetic earth pole. Our robot can set its moving direction using compass by taking feedback from GPS receiver. This robot is controlled by an Arduino Mega 2560 micro-controller which provides PWM to DC motors. After moving each step, the controller checks the error again between the actual GPS angle and heading angle. Depending on the error value, controller generates PWM signal. Thus it repeats the process again and again until it arrives the destination. Using sonar sensor, it can detect an obstacle coming in front of it. By sensing obstacle, our robot can change its direction to avoid it. Additional features include live video feedback with a camera mounted on the robotic structure and auto-saving of the captured data or images via an attached memory card. The functionality and accuracy of this GPS guided robot have been tested in different locations. Our robotic system shows reasonably good accuracy in terms of heading angle and a small average deviation from the target co-ordinates. Our designed autonomous path-traversing robot with its multidimensional features and application prospects can ensure safety, flexibility and accuracy in performing sensitive and risky tasks in multiple aspects of day-to-day life.

### 3.3 Block Diagram:

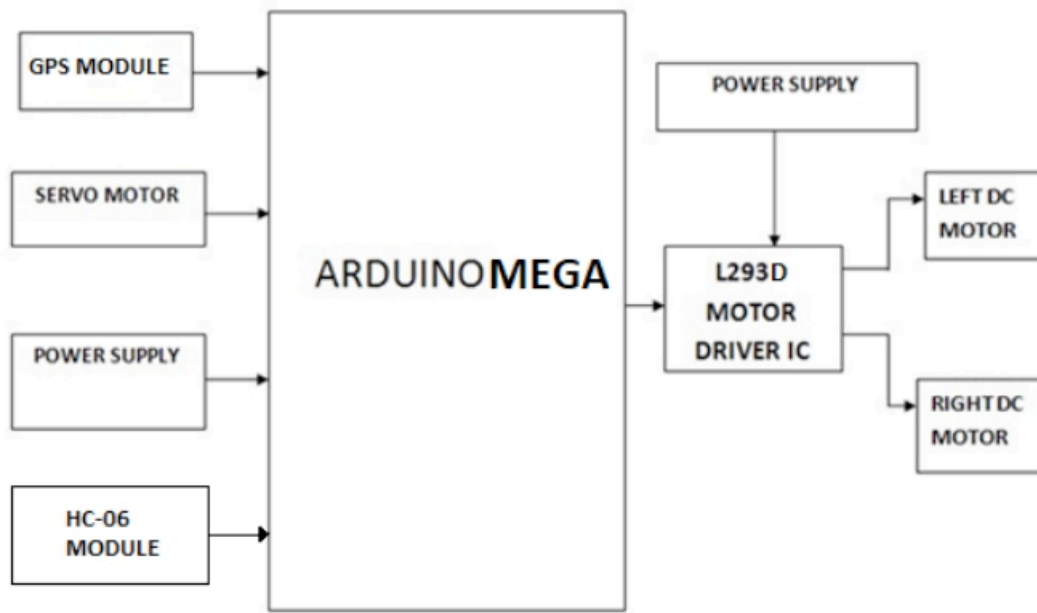


Figure 3.2: Block diagram of Automated Robotic system

### 3.3 Working:

The algorithm begins with finding the direction required for the rover to move. This can be done by continuously calculating current angle of the rover from digital compass reading and target angle from the two sets (current and destination location) of GPS coordinate. Since GPS data needs some time to become stable, the rover must wait some time initially to retrieve current GPS coordinate accurately. The destination coordinate is given initially. The coordinates are then passed to a function that calculates the target angular position and the rover is rotated accordingly. The rover must continuously loop through this set of instructions until the desired point is reached. After every 3 seconds, the rover checks for the next GPS coordinates and reevaluates its target angular position. The working principle is described below–

- At first GPS module reads the current location. Then we calculate the target angle according to the target location by this algorithm–  
$$dlon = \text{radians}(\text{target longitude} - \text{current longitude}); \text{lat1} = \text{radians}(\text{current latitude}); \text{lat2} = \text{radians}(\text{target latitude}); x = \sin(dlon) \times \cos(\text{lat2});$$
$$y = \cos(\text{lat1}) \times \sin(\text{lat2}) - \sin(\text{lat1}) \times \cos(\text{lat2}) \times \cos(dlon); \text{target angle} = \text{atan2}(x, y) \times (180/\pi); \text{if } (\text{target\_angle} < 0) \text{ target angle} += 360^\circ$$
- The compass also reads the current heading angle of the rover. Then it calculates the error angle between target and current heading angle.
- From P-controller, two motors will get pulse according to the error and will try to align to the target angular position. Thus it will minimize the error.
- After 3 seconds, we again calculate the current GPS location and measure the target angle if there is any disturbance occurred during the path.
- Then again by reading the compass angle, the rover will align to the target direction.

Sonar sensor emits burst signal after 2 seconds interval to find the existence of any obstacle.

Each time GPS module reads the current location and calculates the difference with the target location. If the difference is below a threshold value, we stop the motor pulse as the rover has reached its destination approximately.

### 3.6 Flow Chart:

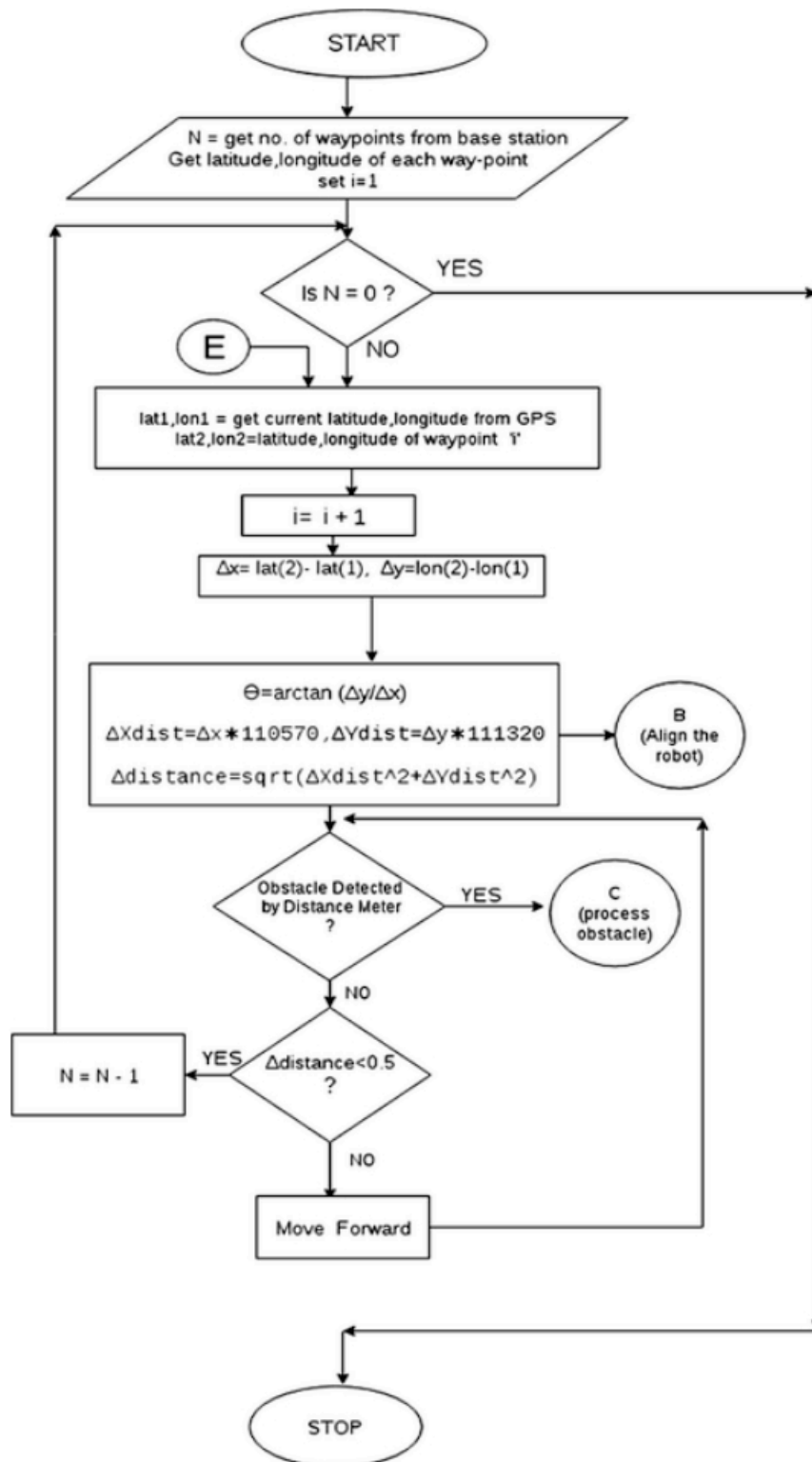


Figure:3.6:Flowchart

**MODULE-4**  
**RESULTS & CONCLUSION**

## 1. Results:

The assumptions of the ideal testing condition for the robot is that the track will be smooth and the GPS module will constantly provide accurate information. But during testing, these factors however came into play and caused some minor variations in the results. Some actual heading angles and the heading angles after motor drive are given in Table II. The TABLE II: Actual and current heading angle errors between these angles were higher if there was no Pcontroller. By implementing P-controller, PWM motor drive gives more accurate heading angle. We measured eight random samples and found the accuracy within ~95% to ~98.5%.

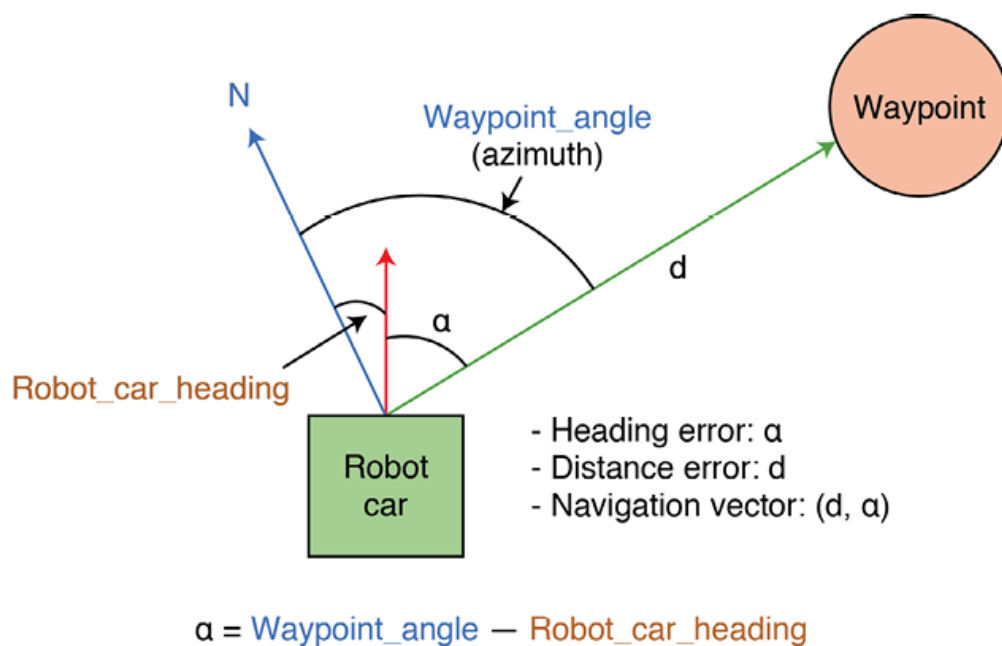


Table II

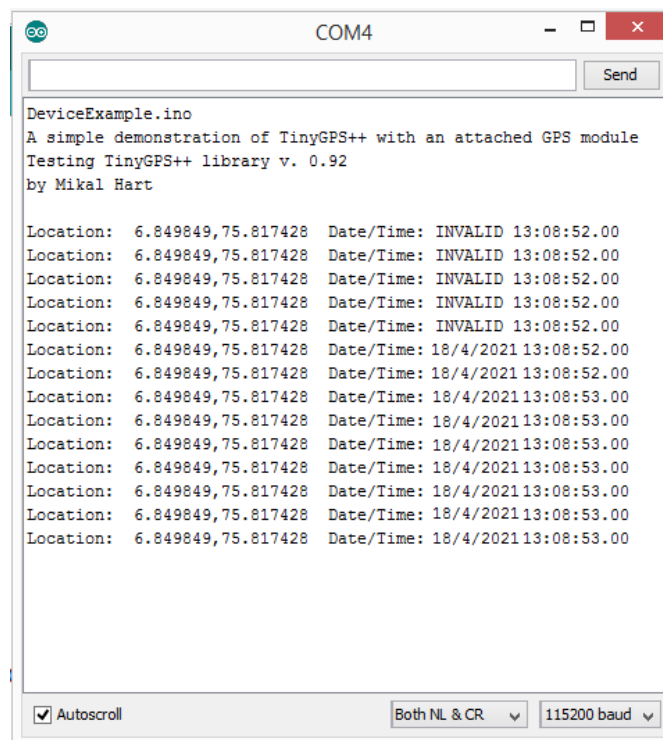
Sometimes the small caster ball in the back can get lodged in a crack in the surface of the track, thus altering the direction of movement. But our controller detects the heading angle after 1 second interval. So, if the robot moves in different direction from its actual path, the algorithm of P-controller takes the robot in the right course smoothly. Moreover, the robot can change its course by itself by sensing obstacles. After passing static obstacle, it then goes independently in the right track. But for moving obstacle, it stops for a while to pass the obstacle and then starts toward its right course. Using sensitive digital compass, our robot deviates very little from its path due to mechanical and surface constraints. The deviation from heading angle and accuracy rate for some random tests is illustrated .

GPS data sometimes fluctuates randomly within a fixed location. As our robot takes decision based on the data from GPS module, so it goes to wrong direction sometimes. But since it takes data after fixed interval, it can go back to its

destination by balancing data with digital compass

## 2. Conclusion:

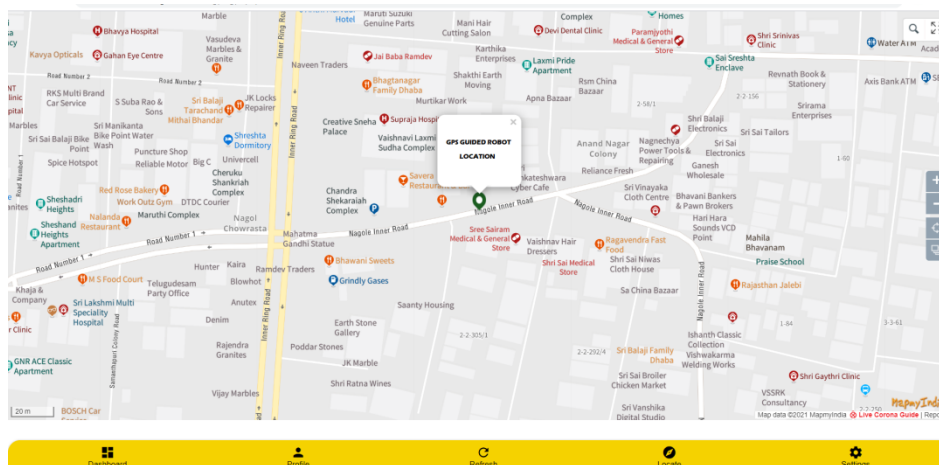
In summary, a low cost GPS guided autonomous transporter robot has been designed, constructed and tested. The employed system autonomously navigates its way through a path, set out in GPS coordinates. By avoiding obstacle, it can successfully deliver object to the prescribed location with an accuracy more than 95%. This land vehicle can be a potential candidate for transporting object independently instead of human thereby ensuring industrial automation as well as exact controlling of operations in sensitive research and health hazardous areas. At the same time, this can be of great use as a surveillance robot during emergency such as in the event of a disaster. Thus, the versatility of our designed autonomous object transporter robot coupled with its high degree of accuracy and precision can guarantee safety and flexibility in performing sensitive tasks in multiple aspects of day-to-day life.



The screenshot shows a serial monitor window titled 'COM4'. It displays a series of log messages from a program named 'DeviceExample.ino'. The messages show the robot's location (6.849849, 75.817428) and the date/time (18/4/2021 13:08:52.00 to 13:08:53.00). The location remains constant while the time progresses. At the bottom, there are controls for 'Autoscroll' (checked), a line ending dropdown set to 'Both NL & CR', and a baud rate dropdown set to '115200 baud'.

```
DeviceExample.ino
A simple demonstration of TinyGPS++ with an attached GPS module
Testing TinyGPS++ library v. 0.92
by Mikal Hart

Location: 6.849849,75.817428 Date/Time: INVALID 13:08:52.00
Location: 6.849849,75.817428 Date/Time: INVALID 13:08:52.00
Location: 6.849849,75.817428 Date/Time: INVALID 13:08:52.00
Location: 6.849849,75.817428 Date/Time: INVALID 13:08:52.00
Location: 6.849849,75.817428 Date/Time: INVALID 13:08:52.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:52.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:52.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
Location: 6.849849,75.817428 Date/Time: 18/4/2021 13:08:53.00
```



## **4.1 Results:**

The assumptions of the ideal testing condition for the robot is that the track will be smooth and the GPS module will constantly provide accurate information. But during testing, these factors however came into play and caused some minor variations in the results. Some actual heading angles and the heading angles after motor drive are given in Table II. The TABLE II: Actual and current heading angle errors between these angles were higher if there was no Pcontroller. By implementing P-controller, PWM motor drive gives more accurate heading angle. We measured eight random samples and found the accuracy within ~95% to ~98.5%.

Sometimes the small caster ball in the back can get lodged in a crack in the surface of the track, thus altering the direction of movement. But our controller detects the heading angle after 1 second interval. So, if the robot moves in different direction from its actual path, the algorithm of P-controller takes the robot in the right course smoothly. Moreover, the robot can change its course by itself by sensing obstacles. After passing static obstacle, it then goes independently in the right track. But for moving obstacle, it stops for a while to pass the obstacle and then starts toward its right course. Using sensitive digital compass, our robot deviates very little from its path due to mechanical and surface constraints. The deviation from heading angle and accuracy rate for some random tests is illustrated .

GPS data sometimes fluctuates randomly within a fixed location. As our robot takes decision based on the data from GPS module, so it goes to wrong direction sometimes. But since it takes data after fixed interval, it can go back to its destination by balancing data with digital compass

## **4.2 Conclusion:**

In summary, a low cost GPS guided autonomous transporter robot has been designed, constructed and tested. The employed system autonomously navigates its way through a path, set out in GPS coordinates. By avoiding obstacle, it can successfully deliver object to the prescribed location with an accuracy more than 95%. This land vehicle can be a potential candidate for transporting object independently instead of human thereby ensuring industrial automation as well as exact controlling of operations in sensitive research and health hazardous areas. At the same time, this can be of great use as a surveillance robot during emergency such as in the event of a disaster. Thus, the versatility of our designed autonomous object transporter robot coupled with its high degree of accuracy and precision can guarantee safety and flexibility in performing sensitive tasks in multiple aspects of day-to-day life.

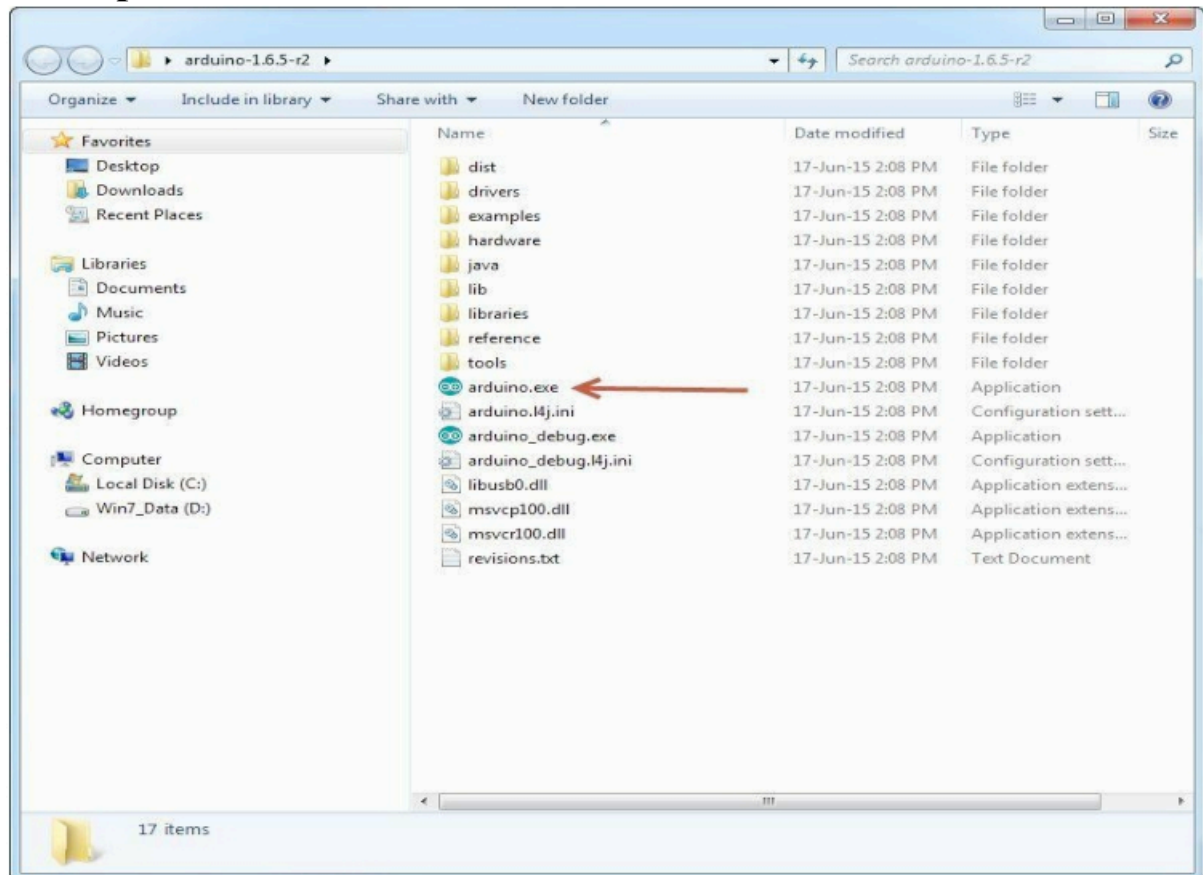
# Appendix



## A.SOFTWARE DESCRIPTION

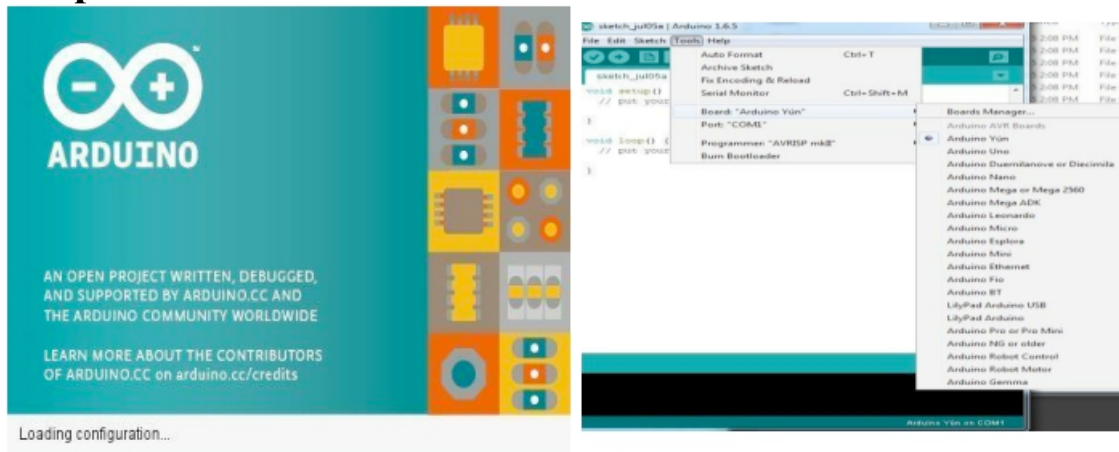
### STEPS FOR WRITING EMBEDDED PROGRAMMING:

#### Step 1: Download Arduino IDE V1.6.5



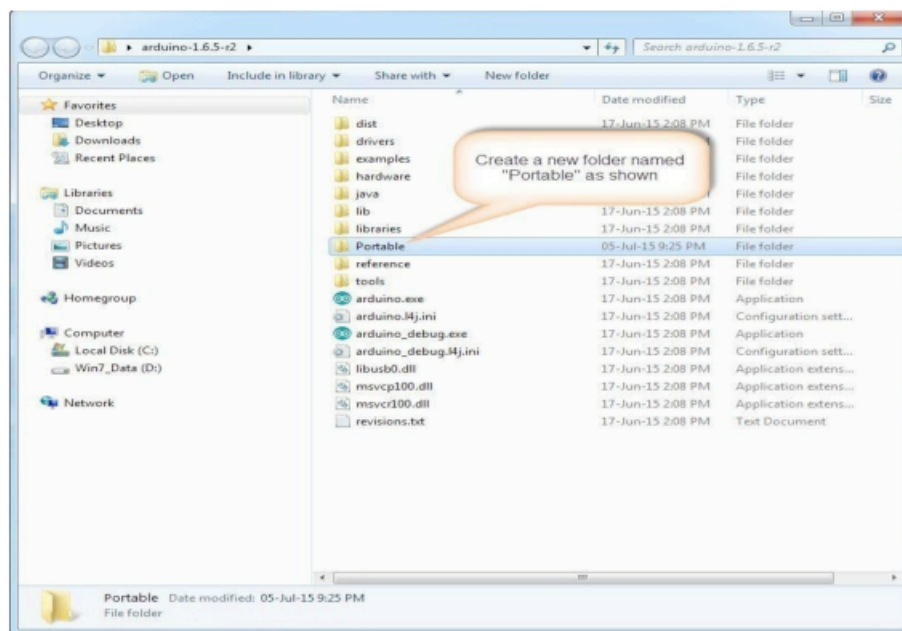
- Download Arduino IDE v1.6.5 from Arduino IDE I prefer to download a zip file (arduino-1.6.5-r2-windows.zip). Then extract the zip file to your preferred location. All folders will appear as the figure.
- Double click "arduino.exe" to run the IDE.

## Step 2: Check the Available Boards of Arduino IDE



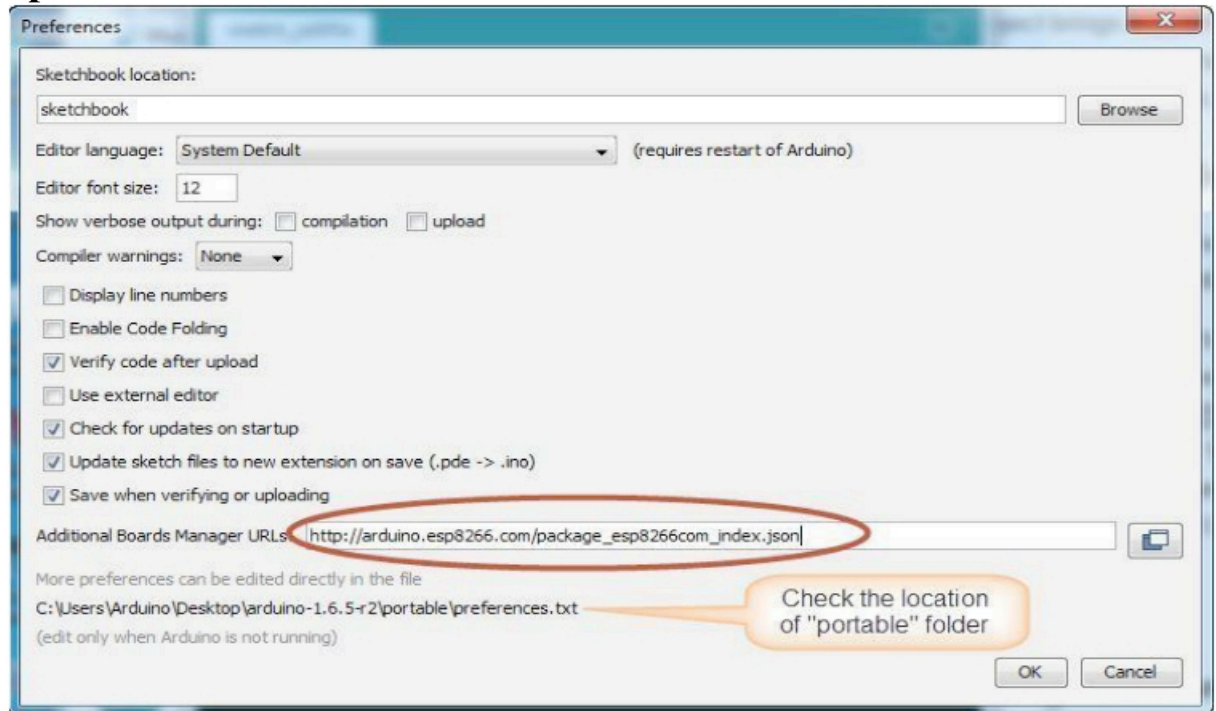
- After running the "arduino.exe", Now we will see the ARDUINO logo, then check the available boards by Tools -> Board as shown in figure. Note that there is no board for ATMEGA 328P.

## Step 3: Add a Folder at Arduino IDE Folder Location



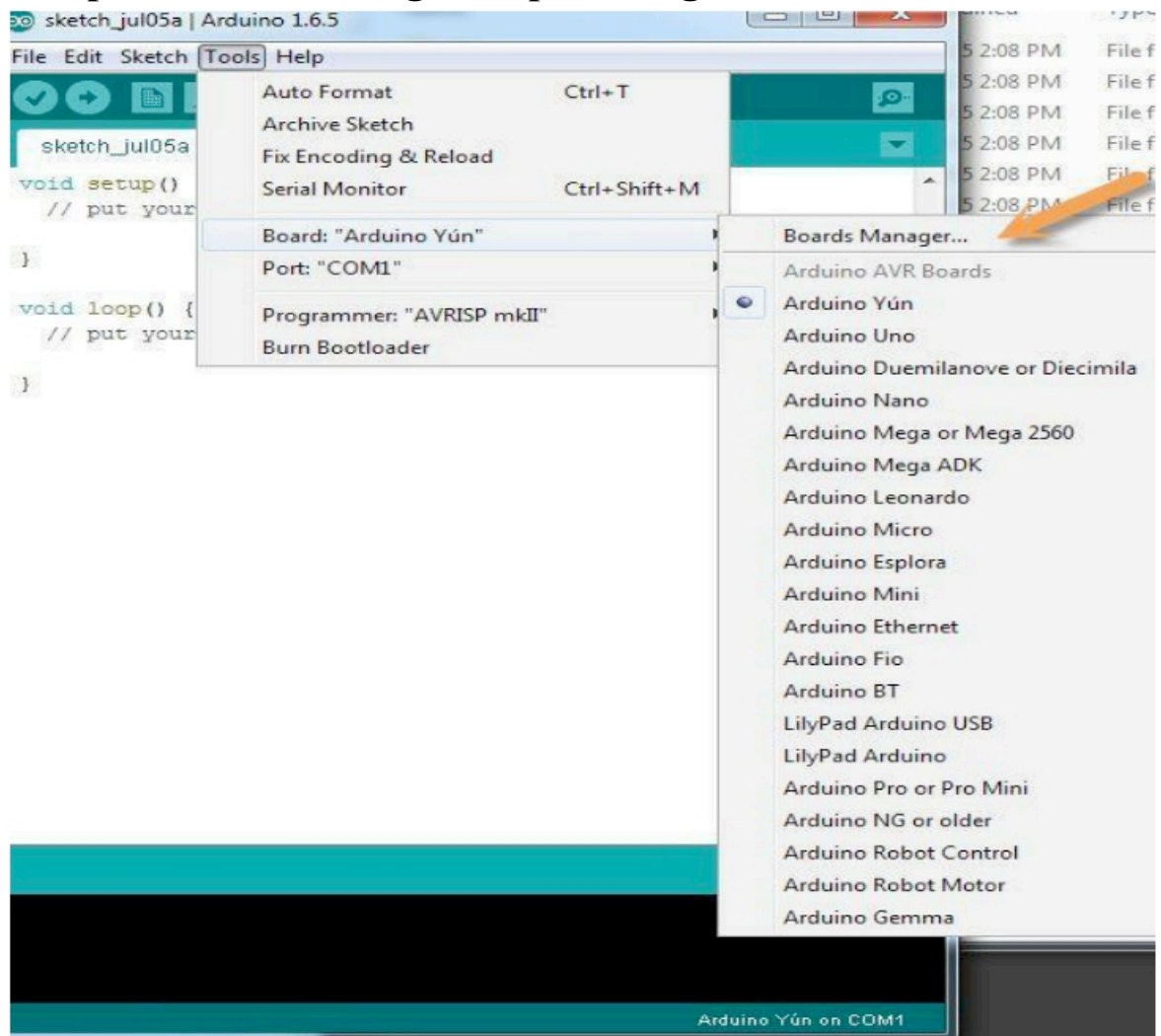
- Now we are going to create a folder named "Portable" at Arduino IDE folder location as shown in figure.:
- Arduino IDE with "Portable Arduino" experimental feature. The feature is disabled by default. To enable the feature, we have to manually create a folder

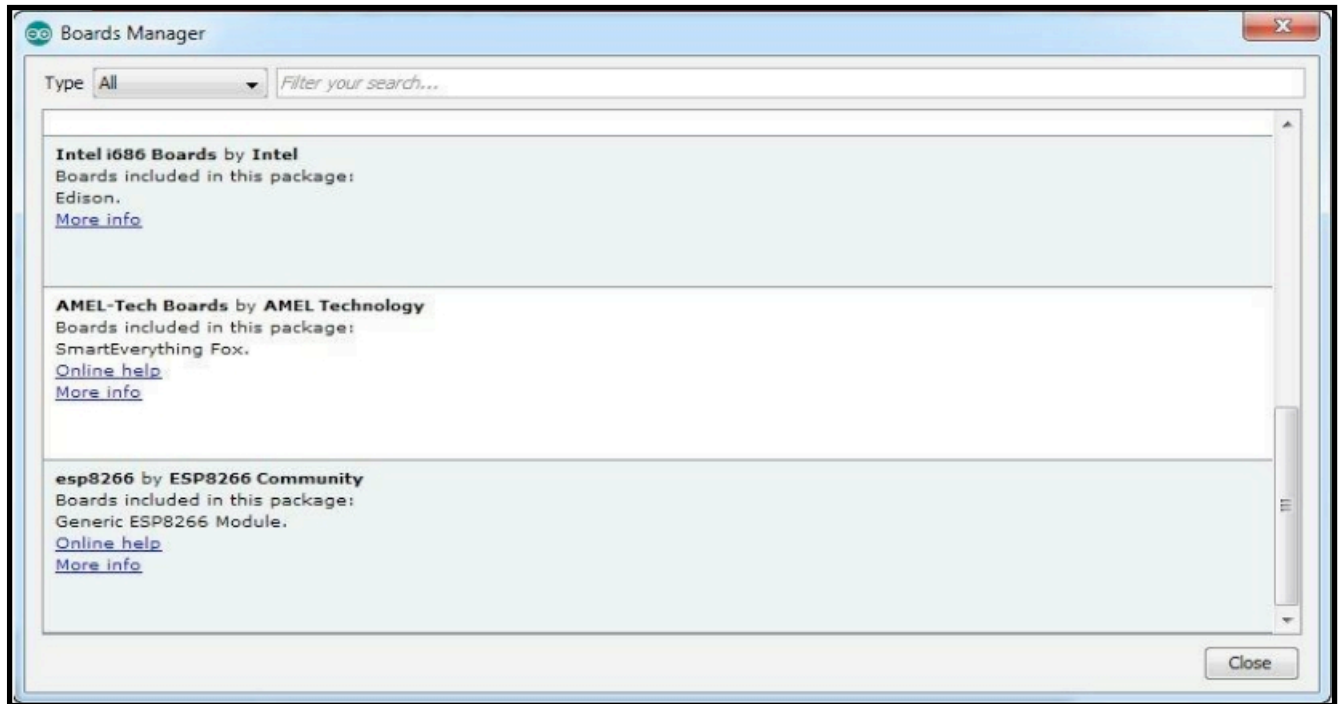
## Step 4: Edit Preferences of Arduino IDE



- Now edit Preferences of the IDE by File -> Preferences and we will see the "Preferences" window.
- From there, search for "Additional Boards Manager URLs": and add the link:
- [http://arduino.Atmega 328p.com/package\\_Atmega 328pcom\\_index.json](http://arduino.Atmega 328p.com/package_Atmega 328pcom_index.json)
- And note that the location of preferences.txt file is now under "portable" folder that we can created as shown in figure.
- Then click OK button.
- In case of "Additional Boards Manager URLs" problem, please refer to the web site Arduino-compatible IDE with ATMEGA 328P for URL.

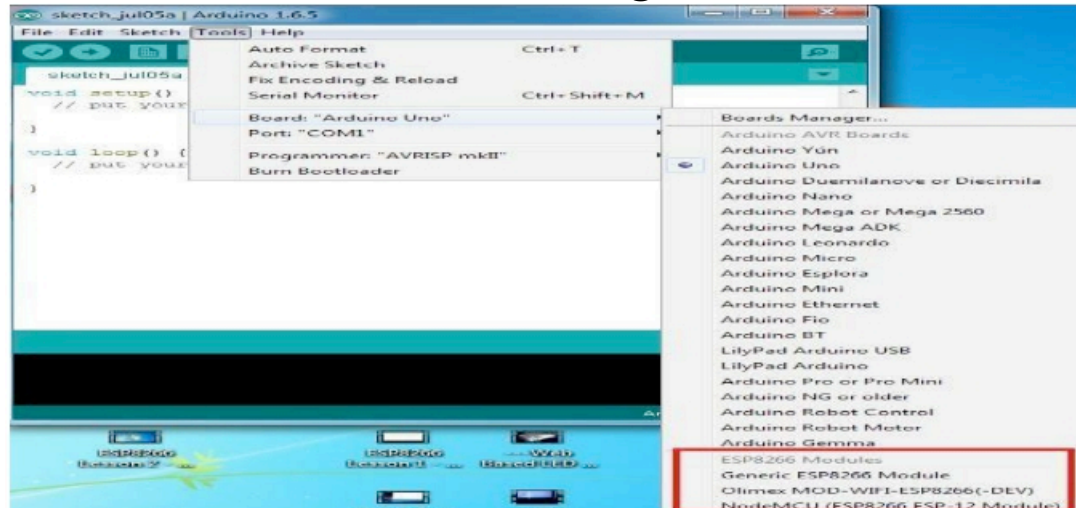
## Step 5: Install Atmega 328p Package





- From Arduino IDE, Tools -> Board -> Boards Manager as shown in figure. Then click Boards Manager. From Boards Manager window, wait for a while for refresh the list and search for "Atmega 328p by ATMEGA 328P Community" and click Install button.
- Installation process can take a while, so please be patient! It's about 153,807 kb. After installation finished, restart the Arduino IDE again to make sure that all the boards loaded.

## Step 6: Check the Available Boards Again



- Now we can check for ESP5266 Module by clicking Tools -> Board -> Generic ATMEGA 328P Module as shown in figure. And we can test Blink program.
- Now the size of Arduino IDE folder becomes about 915 MB. I suggest that we may zip or rar it to reduce the size.
- Now we have a folder which can be portable for Arduino-compatible IDE with ATMEGA 328P support.

### Arduino Software (IDE) used to

- Writing Sketches
  - File
  - Edit
  - Sketch
  - Tools
- Sketchbook
- Uploading
- Libraries
- Serial Monitor
- Preferences
- Language Support
- Boards

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.

## 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.

**NB:** Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.



### Verify

Checks your code for errors compiling it.



### Upload

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

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



### New

Creates a new sketch.



### Open

Presents a menu of all the sketches in your sketchbook.

Clicking one will open it within the current window overwriting its content.



**Save**

Saves your sketch.

**SerialMonitor**

Opens the serial monitor.

Additional commands are found within the five menus: **File**, **Edit**, **Sketch**, **Tools**, **Help**. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

**File****New**

Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.

**Open**

Allows to load a sketch file browsing through the computer drives and folders.

**OpenRecent**

Provides a short list of the most recent sketches, ready to be opened.

**Sketchbook**

Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.

**Examples**

Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.

**Close**

Closes the instance of the Arduino Software from which it is clicked.

**Save**

Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.

**Save as...**

Allows to save the current sketch with a different name.



**Page Setup**

It shows the Page Setup window for printing.

**Print**

Sends the current sketch to the printer according to the settings defined in Page Setup.

**Preferences**

Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface

**Quit**

Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

**Edit****Undo/Redo**

Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.

**Cut**

Removes the selected text from the editor and places it into the clipboard.

**Copy**

Duplicates the selected text in the editor and places it into the clipboard.

**Copy for Forum**

Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.

**Copy as HTML**

Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.

**Paste**

Puts the contents of the clipboard at the cursor position, in the editor.

**Select All**

Selects and highlights the whole content of the editor.

**Comment/Uncomment**

Puts or removes the // comment marker at the beginning of each selected line.

**Increase/Decrease Indent**

Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.

**Find**

Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.

*Find Next* Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.

**Find Previous**

Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

**Sketch****Verify/Compile**

Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.

**Upload**

Compiles and loads the binary file onto the configured board through the configured Port.

**Upload Using Programmer**

This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again.

However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a Tools -> Burn Bootloader command must be executed.

**Export Compiled Binary**

Saves a .hex file that may be kept as archive or sent to the board using other tools.

**Show Sketch Folder**

Opens the current sketch folder.

**Include Library**

Adds a library to your sketch by inserting `#include` statements at the start of your code. For more details, see libraries below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.

**Add File...**

Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right side of the toolbar.

**Tools****Auto Format**

This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.

**Archive Sketch**

Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.

**Fix Encoding & Reload**

Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.

**SerialMonitor**

Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.

**Board**

Select the board that you're using. See below for descriptions of the various boards.

**Port**

This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.

**Programmer**

For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're burning a bootloader to a new microcontroller, you will use this.

## **BurnBootloader**

The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino or Genuino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the Boards menu before burning the bootloader on the target board. This command also set the right fuses.

## **Help**

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website. This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

## **Sketchbook**

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the File > Sketchbook menu or from the Open button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the Preferences dialog.

Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

## **Tabs, Multiple Files, and Compilation**

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h)

## 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. Current Arduino boards will reset automatically and begin the upload. 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 we 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).

## Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the Sketch > Import Library menu. This will insert one or more `#include` statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its `#include` statements from the top of your code.

There is a list of libraries in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these instructions for installing a third-party library.

### **Third-Party Hardware**

Support for third-party hardware can be added to the hardware directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the hardware directory, then unzip the third-party

Platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

### **Serial Monitor**

This displays serial sent from the Arduino or Genuino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to Serial.begin in your sketch. Note that on Windows, Mac or Linux the board will reset (it will rerun your sketch) when you connect with the serial monitor. Please note that the Serial Monitor does not process control characters; if your sketch needs a complete management of the serial communication with control characters, you can use an external terminal program and connect it to the COM port assigned to your Arduino board.

### **Preferences**

Some preferences can be set in the preferences dialog (found under the Arduino menu on the Mac, or File on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

## Language Support



Since version 1.0.1 , the Arduino Software (IDE) has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.

If you would like to change the language manually, start the Arduino Software (IDE) and open the Preferences window. Next to the Editor Language there is a dropdown menu of currently supported languages. Select your preferred language from the menu,

and restart the software to use the selected language. If your operating system language is not supported, the Arduino Software (IDE) will default to English.

You can return the software to its default setting of selecting its language based on your operating system by selecting System Default from the Editor

Language drop-down. This setting will take effect when you restart the Arduino Software (IDE). Similarly, after changing your operating system's settings, you must restart the Arduino Software (IDE) to update it to the new default language.

## **Boards**

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader. You can find a comparison table between the various boards [here](#).

Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The Boards Manager included in the standard installation allows to add support for the growing number of new boards based on different cores like ATMEGA 328P so on.

## **a. Source code (Embedded system)**

### **Sweep**

```
void sweep()                // Can be used to simulate Metal Detecting or to rotate
Ping Sensor
{
  myservo.attach(9);
  StopCar();
```



```

Forward_Meter();
StopCar();

for(pos = 60; pos <= 120; pos += 1) // goes from 0 degrees to 180 degrees
{
    // in steps of 1 degree
    myservo.write(pos);          // tell servo to go to position in variable 'pos'
    delay(15);                   // waits 15ms for the servo to reach the position
}
for(pos = 120; pos >= 60; pos -= 1) // goes from 180 degrees to 0 degrees
{
    myservo.write(pos);          // tell servo to go to position in variable 'pos'
    delay(15);                   // waits 15ms for the servo to reach the position
}

myservo.write(90);               // tell servo to go to position in variable 'pos'
delay(15);                       // waits 15ms for the servo to reach the position

myservo.detach();                // Disengage Servo Motor
}

```

## Startup

```

void Startup()
{
    myservo.detach();
    Serial.println("Pause for Startup... ");

    for (int i=5; i >= 1; i--)          // Count down for X seconds
    {
        Serial1.print("Pause for Startup... ");
        Serial1.print(i);
        delay(1000);                   // Delay for X seconds
    }

    Serial1.println("Searching for Satellites ");
    Serial.println("Searching for Satellites ");

    while (Number_of_SATS <= 4)          // Wait until x number of satellites
    are acquired before starting main loop
    {
        getGPS();                       // Update gps data
        Number_of_SATS = (int)(gps.satellites.value()); // Query Tiny GPS for the
        number of Satellites Acquired
        bluetooth();                     // Check to see if there are any bluetooth
        commands being received
    }
}

```

```

}
setWaypoint();                // set initial waypoint to current location
wpCount = 0;                  // zero waypoint counter
ac = 0;                        // zero array counter

Serial1.print(Number_of_SATS);
Serial1.print(" Satellites Acquired");

}

```

## Collision\_Avoid

```

void Ping()
{
    currentMillis = millis();

    if ((currentMillis - previousMillis >= interval) && (pingOn == true))
    {
        previousMillis = currentMillis;
        digitalWrite(trigPin, LOW);
        delayMicroseconds(5);
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);

        pinMode(echoPin, INPUT);
        duration = pulseIn(echoPin, HIGH);

        int inches = (duration / 2) / 74;                // convert the time into a distance
        Ping_distance == inches;

        if ((inches < 12) && (blueToothVal != 5))
        {
            Serial1.print("Crash!");
            Serial1.println(inches);
            Reverse();                // Quick reverse to Stop quickly
            delay(100);
            StopCar();
            blueToothVal = 5;          // Set bluetooth value to "Stop"
        }

    }    // end if statement

}    // end Ping()

void pingToggle()                // Turns Collision avoidance on/ off

```

```

{
  if (pingOn == true) {
    pingOn = false;
    Serial1.print("Collision Avoidance OFF");
  }
  else if (pingOn == false) {
    pingOn = true;
    Serial1.print("Collision Avoidance ON");
  }
}
}

```

## Bluetooth

```

//*****
//*****
//*****

// This procedure reads the serial port - Serial1 - for bluetooth commands being sent
from the Android device

void bluetooth()
{
  while (Serial1.available()) // Read bluetooth commands over
Serial1 // Warning: If an error with Serial1 occurs, make sure Arduino Mega 2560 is
Selected
  {
    {
      str = Serial1.readStringUntil('\n'); // str is the temporary variable for
storing the last string sent over bluetooth from Android device
      //Serial.print(str); // for testing purposes
    }

    blueToothVal = (str.toInt()); // convert the string 'str' into an
integer and assign it to blueToothVal
    Serial.print("BlueTooth Value ");
    Serial.println(blueToothVal);

//
//*****
//*****
//*****

switch (blueToothVal)
{
  case 1:
    Serial1.println("Forward");
    Forward();

```

```
break;
```

```
case 2:
```

```
Serial1.println("Reverse");
```

```
Reverse();
```

```
break;
```

```
case 3:
```

```
Serial1.println("Left");
```

```
LeftTurn();
```

```
StopCar();
```

```
break;
```

```
case 4:
```

```
Serial1.println("Right");
```

```
RightTurn();
```

```
StopCar();
```

```
break;
```

```
case 5:
```

```
Serial1.println("Stop Car ");
```

```
StopCar();
```

```
break;
```

```
case 6:
```

```
setWaypoint();
```

```
break;
```

```
case 7:
```

```
goWaypoint();
```

```
break;
```

```
case 8:
```

```
Serial1.println("Turn Around");
```

```
turnAround();
```

```
break;
```

```
case 9:
```

```
Serial1.println("Compass Forward");
```

```
setHeading();
```

```
Compass_Forward();
```

```
break;
```

```
case 10:
```

```
setHeading();
```

```
break;
```

```

    case 11:
        gpsInfo();
        break;

    case 12:
        Serial1.println("Compass Turn Right");
        CompassTurnRight();
        break;

    case 13:
        Serial1.println("Compass Turn Left");
        CompassTurnLeft();
        break;

    case 14:
        Serial1.println("Calibrate Compass");
        calibrateCompass();
        break;

    case 15:
        pingToggle();
        break;

    case 16:
        clearWaypoints();
        break;

    case 17:                // finish with waypoints
        ac = 0;
        Serial1.print("Waypoints Complete");
        break;

} // end of switch case

//
*****
*****
*****

// Slider Value for Speed

if (blueToothVal)
{ //
    Serial.println(blueToothVal); if
    (blueToothVal >= 1000)
    {
        Serial1.print("Speed set To: ");

```

```

Serial1.println(blueToothVal - 1000);
turn_Speed = (blueToothVal - 1000);
Serial.println();
Serial.print("Turn Speed ");
Serial.println(turn_Speed);
}

} /*

//
*****
*****
*****
// Detect for Mines - Sweep Not Used

else if (blueToothVal== 15)
{
    Serial1.println("Detecting");
    sweep();
}

//
*****
*****
*****
*/

} // end of while loop Serial1 read

// if no data from Bluetooth
if (Serial1.available() < 0) // if an error occurs, confirm that the
arduino mega board is selected in the Tools Menu
{
    Serial1.println("No Bluetooth Data ");
}

} // end of bluetooth procedure

```

## Go\_Waypoint

```

void goWaypoint()
{
    Serial1.println("Go to Waypoint"); //
    Serial.print("Home_Latarray"); //
    Serial.print(Home_LATarray[ac],6);
}

```

```

//Serial.print(""); //
Serial.println(Home_LONarray[ac],6);

//Serial1.print("Distance to Home");
//Serial1.print(Distance_To_Home);

//Serial1.print("ac=");
//Serial1.print(ac);

while(true)
{
    bluetooth(); // Start of Go_Home procedure
                // Run the Bluetooth procedure to see
    if there is any data being sent via BT
    if (blueToothVal == 5){break;} // If a 'Stop' Bluetooth command
    is received then break from the Loop
    getCompass(); // Update Compass heading
    getGPS(); // Tiny GPS function that retrieves GPS
    data - update GPS location// delay time changed from 100 to 10

    if (millis() > 5000 && gps.charsProcessed() < 10) // If no Data from GPS
    within 5 seconds then send error
        Serial1.println(F("No GPS data: check wiring"));

    Distance_To_Home =
    TinyGPSPlus::distanceBetween(gps.location.lat(),gps.location.lng(),Home_LATarray[ac], Home_LONarray[ac]); //Query Tiny GPS for Distance to Destination
    GPS_Course =
    TinyGPSPlus::courseTo(gps.location.lat(),gps.location.lng(),Home_LATarray[ac],Home_LONarray[ac]); //Query Tiny GPS for Course to Destination

    /*
    if (Home_LATarray[ac] == 0)
    { Serial1.print("End of
    Waypoints"); StopCar();
    break;
    }
    */

    if (Distance_To_Home == 0) // If the Vehicle has reached it's
    Destination, then Stop
    {
        StopCar(); // Stop the robot after each waypoint is
        reached
        Serial1.println("You have arrived!"); // Print to Bluetooth device -
        "You have arrived"
        ac++; // increment counter for next waypoint
        break; // Break from Go_Home procedure and
        send control back to the Void Loop
    }
}

```

```

// go to next waypoint

}

if ( abs(GPS_Course - compass_heading) <= 15)           // If GPS Course and
the Compass Heading are within x degrees of each other then go Forward
// otherwise find the shortest turn radius and
turn left or right
{
    Forward();                                           // Go Forward
} else
{
    int x = (GPS_Course - 360);                         // x = the GPS desired heading -
360
    int y = (compass_heading - (x));                    // y = the Compass heading - x
    int z = (y - 360);                                  // z = y - 360

    if ((z <= 180) && (z >= 0))                          // if z is less than 180 and not a
negative value then turn left otherwise turn right
        { SlowLeftTurn(); }
    else { SlowRightTurn(); }
}

}                                                         // End of While Loop

}                                                         // End of Go_Home procedure

```

## GPS\_Compass

```

void calibrateCompass()                                // Experimental Use Only to
Calibrate Magnetometer/ Compass
{
    int minX = 0;
    int maxX = 0;
    int minY = 0;
    int maxY = 0;
    int offX = 0;
    int offY = 0;

    for (int i=1000; i >= 1; i--)
    {
        Vector mag = compass.readRaw();
    }
}

```



```

// Determine Min / Max values
if (mag.XAxis < minX) minX = mag.XAxis;
if (mag.XAxis > maxX) maxX = mag.XAxis;
if (mag.YAxis < minY) minY = mag.YAxis;
if (mag.YAxis > maxY) maxY = mag.YAxis;

offX = (maxX + minX)/2; // Calculate offsets
offY = (maxY + minY)/2;

delay(10);
//Serial.print("Compass X & Y offset:
"); //Serial.print(offX);
//Serial.print(" "); //
Serial.print(offY); //
Serial.print("\n");

} // end of for loop

StopCar();

Serial1.print("Compass X & Y offset: ");
Serial1.print(offX);
Serial1.print(" ");
Serial1.print(offY);
Serial1.print("\n");
compass.setOffset(offX,offY); // Set calibration offset
}

//
*****
*****
*****

void getGPS() // Get Latest GPS coordinates
{
    while (Serial2.available() > 0)
        gps.encode(Serial2.read());
}

//
*****
*****
*****

void setWaypoint() // Set up to 5 GPS waypoints
{

```

```

//if ((wpCount >= 0) && (wpCount < 50))
if (wpCount >= 0)
{
    Serial1.print("GPS Waypoint ");
    Serial1.print(wpCount + 1);
    Serial1.print(" Set ");
    getGPS();                // get the latest GPS coordinates
    getCompass();            // update latest compass heading

    Home_LATarray[ac] = gps.location.lat();    // store waypoint in an array
    Home_LONarray[ac] = gps.location.lng();    // store waypoint in an array

    Serial.print("Waypoint #1: ");
    Serial.print(Home_LATarray[0],6);
    Serial.print(" ");
    Serial.println(Home_LONarray[0],6);
    Serial.print("Waypoint #2: ");
    Serial.print(Home_LATarray[1],6);
    Serial.print(" ");
    Serial.println(Home_LONarray[1],6);
    Serial.print("Waypoint #3: ");
    Serial.print(Home_LATarray[2],6);
    Serial.print(" ");
    Serial.println(Home_LONarray[2],6);
    Serial.print("Waypoint #4: ");
    Serial.print(Home_LATarray[3],6);
    Serial.print(" ");
    Serial.println(Home_LONarray[3],6);
    Serial.print("Waypoint #5: ");
    Serial.print(Home_LATarray[4],6);
    Serial.print(" ");
    Serial.println(Home_LONarray[4],6);

    wpCount++;                // increment waypoint counter
    ac++;                    // increment array counter

}
else {Serial1.print("Waypoints Full");}
}

//
*****
*****
*****

void clearWaypoints()
{

```

```

memset(Home_LATarray, 0, sizeof(Home_LATarray));          // clear the array
memset(Home_LONarray, 0, sizeof(Home_LONarray));          // clear the array
wpCount = 0;                                              // reset increment counter to 0
ac = 0;

Serial1.print("GPS Waypoints Cleared");                  // display waypoints cleared

}

//
*****
*****
*****

void getCompass()                                         // get latest compass value
{

    Vector norm = compass.readNormalize();

    // Calculate heading
    float heading = atan2(norm.YAxis, norm.XAxis);

    if(heading < 0)
        heading += 2 * M_PI;
    compass_heading = (int)(heading * 180/M_PI);
    calculation to variable (compass_heading) and convert to integer to remove decimal
    places

}

//
*****
*****
*****

void setHeading()
// This procedure will set the current heading
and the Heading(s) of the robot going away and returning using opposing degrees
from 0 to 360;
// for instance, if the car is leaving on a 0
degree path (North), it will return on a 180 degree path (South)
{
    for (int i=0; i <= 5; i++)                          // Take several readings from the
    compass to insure accuracy
    {
        getCompass();                                    // get the current compass heading
    }
}

```

```

    desired_heading = compass_heading;           // set the desired heading to
equal the current compass heading
    Heading_A = compass_heading;                 // Set Heading A to current
compass
    Heading_B = compass_heading + 180;           // Set Heading B to current
compass heading + 180

    if (Heading_B >= 360)                         // if the heading is greater than 360
then subtract 360 from it, heading must be between 0 and 360
    {
        Heading_B = Heading_B -
        360; }

    Serial1.print("Compass Heading Set: ");
    Serial1.print(compass_heading);
    Serial1.print(" Degrees");

    Serial.print("desired heading");
    Serial.println(desired_heading);
    Serial.print("compass heading");
    Serial.println(compass_heading);

}

//
*****
*****
*****

void gpsInfo()                                  // displays Satellite data to user
{
    Number_of_SATS = (int)(gps.satellites.value()); //Query Tiny GPS for the
number of Satellites Acquired
    Distance_To_Home =
TinyGPSPlus::distanceBetween(gps.location.lat(),gps.location.lng(),Home_LATarra
y[ac], Home_LONarray[ac]); //Query Tiny GPS for Distance to Destination
    Serial1.print("Lat:");
    Serial1.print(gps.location.lat(),6);
    Serial1.print(" Lon:");
    Serial1.print(gps.location.lng(),6);
    Serial1.print(" ");
    Serial1.print(Number_of_SATS);
    Serial1.print(" SATs ");
    Serial1.print(Distance_To_Home);
    Serial1.print("m");
    Serial.print("Distance to Home ");

```

```
Serial.println(Distance_To_Home);

}
```

## Steering

```
void Forward()
{
    Ping();
    Serial.println("Forward");
    motor1.setSpeed(mtr_Spd);
    motor2.setSpeed(mtr_Spd);
    motor3.setSpeed(mtr_Spd);
    motor4.setSpeed(mtr_Spd);

    motor1.run(FORWARD);           // go forward all wheels
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(FORWARD);
}

//
*****
*****
*****

void Forward_Meter()
{
    motor1.setSpeed(mtr_Spd);
    motor2.setSpeed(mtr_Spd);
    motor3.setSpeed(mtr_Spd);
    motor4.setSpeed(mtr_Spd);

    motor1.run(FORWARD);           // go forward all wheels
    for specified time
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(FORWARD);
    delay(500);
}

//
*****
*****
*****
```

```

void Reverse()
{
    motor1.setSpeed(mtr_Spd);
    motor2.setSpeed(mtr_Spd);
    motor3.setSpeed(mtr_Spd);
    motor4.setSpeed(mtr_Spd);

    motor1.run(BACKWARD);           // Reverse all wheels
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
    motor4.run(BACKWARD);
}

```

```

//
*****
*****
*****

```

```

void LeftTurn()
{
    motor1.setSpeed(mtr_Spd);
    motor2.setSpeed(mtr_Spd);
    motor3.setSpeed(mtr_Spd);
    motor4.setSpeed(mtr_Spd);

    motor1.run(BACKWARD);           // Turn left
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(BACKWARD);
    delay(100);
}

```

```

//
*****
*****
*****

```

```

void RightTurn()
{
    motor1.setSpeed(mtr_Spd);
    motor2.setSpeed(mtr_Spd);
    motor3.setSpeed(mtr_Spd);
    motor4.setSpeed(mtr_Spd);

    motor1.run(FORWARD);
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
}

```

```
    motor4.run(FORWARD);
    delay(100); //delay for 100ms more
    responsive turn per push on bluetooth
}
```

```
//
*****
*****
*****
```

```
void SlowLeftTurn()
{

    motor1.setSpeed(turn_Speed);
    motor2.setSpeed(turn_Speed);
    motor3.setSpeed(turn_Speed);
    motor4.setSpeed(turn_Speed);

    motor1.run(BACKWARD);
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(BACKWARD);
}
```

```
//
*****
*****
*****
```

```
// Turning too fast will over-shoot the compass and GPS course
```

```
void SlowRightTurn()
{

    motor1.setSpeed(turn_Speed);
    motor2.setSpeed(turn_Speed);
    motor3.setSpeed(turn_Speed);
    motor4.setSpeed(turn_Speed);

    motor1.run(FORWARD);
    motor2.run(BACKWARD);
    motor3.run(BACKWARD);
    motor4.run(FORWARD);
}
```

```

//
*****
*****
*****

void StopCar()
{

    motor1.run(RELEASE);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor4.run(RELEASE);

}

// Compass Drive Section
// This Portion of code steers the Vehicle based on the compass heading
//
*****
*****
*****

void CompassTurnRight()                                // This Function Turns
the car 90 degrees to the right based on the current desired heading
{
    StopCar();
    getCompass();                                     // get current compass heading

    desired_heading = (desired_heading + 90);          // set desired_heading
to plus 90 degrees
    if (desired_heading >= 360) {desired_heading = (desired_heading - 360);}    // if
the desired heading is greater than 360 then subtract 360 from it

    while ( abs(desired_heading - compass_heading) >= compass_dev)            // If
the desired heading is more than Compass Deviation in degrees from the actual
compass heading then
    {
                                                // correct direction by turning left
or right

        getCompass();                                // Update compass heading
during While Loop
        bluetooth();                                  // if new bluetooth value
received break from loop
        if (blueToothVal == 5){break;}                // If a Stop Bluetooth
command ('5') is received then break from the Loop

```



```
    if (desired_heading >= 360) {desired_heading = (desired_heading - 360);}    // if
the desired heading is greater than 360 then subtract 360 from it
```

```
    int x = (desired_heading - 359);                                // x = the GPS desired
heading - 360
```

```
    int y = (compass_heading - (x));                                // y = the Compass
heading - x
```

```
    int z = (y - 360);                                            // z = y - 360
```

```
    if ((z <= 180) && (z >= 0))                                    // if z is less than 180
and not a negative value then turn left
```

```
    {                                                            // otherwise turn right
        SlowLeftTurn();
```

```
    }
```

```
    else
```

```
    {
```

```
        SlowRightTurn();
```

```
    }
```

```
}
```

```
{
    StopCar();                                                    // Stop the Car when desired
heading and compass heading match
```

```
}
```

```
}
```

```
//
```

```
*****
*****
*****
```

```
void CompassTurnLeft()                                          // This procedure turns
the car 90 degrees to the left based on the current desired heading
```

```
{
```

```
    StopCar();
```

```
    getCompass();                                                // get current compass heading
```

```
    //desired_heading = (compass_heading - 90);                // set
```

```
desired_heading to compass value minus 90 degrees
```

```
    desired_heading = (desired_heading - 90);                    // set desired_heading
to minus 90 degrees
```

```
    if (desired_heading <= 0) {desired_heading = (desired_heading + 360);}    // if
the desired heading is greater than 360 then add 360 to it
```

```
    while ( abs(desired_heading - compass_heading) >= compass_dev)    // If
the desired heading is more than Compass Deviation in degrees from the actual
compass heading then
```

```

    {
or right
        getCompass();
during While Loop
        bluetooth();
received break from loop
        if (blueToothVal == 5){break;}
command is received then break from the Loop

```

// correct direction by turning left

// Get compass heading again

// if new bluetooth value

// If a 'Stop' Bluetooth

```

    if (desired_heading >= 360) {desired_heading = (desired_heading - 360);}
the desired heading is greater than 360 then subtract 360 from it

```

```

    int x = (desired_heading - 359);
- 360

```

// x = the desired heading

```

    int y = (compass_heading - (x));
heading - x

```

// y = the Compass

```

    int z = (y - 360);
    if (z <= 180)

```

// z = y - 360

// if z is less than 180 and not

a negative value then turn left

```

    // if ((z <= 180) && (z >= 0))

```

// otherwise turn right

```

    {
        SlowLeftTurn();
    }

```

```

    else
    {

```

```

        SlowRightTurn();
    }

```

```

    }
}

```

```

    {
        StopCar();
heading and compass heading match
    }
}

```

// Stop the Car when desired

```

//
*****
*****
*****

```

```

void Compass_Forward()
{

```

```

    while (blueToothVal == 9)
'Stop' command is sent

```

// Go forward until Bluetooth

```

//while (true)
{

```

```

    getCompass();

```

// Update Compass Heading

```

    bluetooth();                                // Check to see if any Bluetooth
commands have been sent
    if (blueToothVal == 5) {break;}              // If a Stop Bluetooth command
('5') is received then break from the Loop

    if ( abs(desired_heading - compass_heading) <= compass_dev ) // If the Desired
Heading and the Compass Heading are within the compass deviation, X degrees of
each other then Go Forward

                                                                    // otherwise find the shortest turn radius
and turn left or right
    {
        Forward();
        Ping();
    } else
    {
        int x = (desired_heading - 359);          // x = the GPS desired heading
- 360
        int y = (compass_heading - (x));          // y = the Compass heading -
x
        int z = (y - 360);                        // z = y - 360

        if ((z <= 180) && (z >= 0))                // if z is less than 180 and not
a negative value then turn left
        {
            SlowLeftTurn();                        // otherwise turn right
            Ping();
        }
        else
        {
            SlowRightTurn();
            Ping();
        }
    }
}
                                                                    // End While Loop
                                                                    // End Compass_Forward

//
*****
*****
*****

void turnAround()                                // This procedure turns the Car
around 180 degrees, every time the "Turn Around" button is pressed
{
    // the car alternates whether the next turn
will be to the left or right - this is determined by the 'pass' variable

```

// Imagine you are cutting the grass, when  
you get to the end of the row - the first pass you are turning one way and on the next  
pass you turn the opposite

if (pass == 0) { CompassTurnRight(); } // If this is the first pass  
then turn right

else { CompassTurnLeft(); } // If this is the second pass then  
turn left

//Forward\_Meter(); // Go forward one meter  
(approximately)  
StopCar(); // Stop the car

if (pass == 0) // If this is the first pass then Turn  
Right  
{  
CompassTurnRight(); // Turn right  
pass = 1 ; // Change the pass value to '1' so that  
the next turn will be to the left  
}

else  
{

if (desired\_heading == Heading\_A) // This section of code  
Alternates the desired heading 180 degrees  
{ // for the Compass drive forward  
desired\_heading = Heading\_B;  
}  
else if (desired\_heading == Heading\_B)  
{  
desired\_heading = Heading\_A;  
}

CompassTurnLeft(); // If this is the second pass then  
Turn Left  
pass = 0; // Change the pass value to '0' so that  
the next turn will be to the right

}

Compass\_Forward(); // Maintain the 'desired heading'  
and drive forward  
}

## GPS\_Guided\_Robot

```
#include <AFMotor.h>
#include "Wire.h"
#include "I2Cdev.h"
#include "HMC5883L.h"
#include <Servo.h>
#include <SoftwareSerial.h>
#include <TinyGPS++.h>

//*****
//*****

// GPS Variables & Setup

int GPS_Course; // variable to hold the gps's determined
course to destination
int Number_of_SATS; // variable to hold the number of
satellites acquired
TinyGPSPlus gps; // gps = instance of TinyGPS
// pin 17 (blue) is connected to the TX on
the GPS
// pin 16 (yellow) is connected to the RX on
the GPS

//*****
//*****

// Setup Drive Motors using the Adafruit Motor Controller version 1.0 Library

AF_DCMotor motor1(1, MOTOR12_64KHZ); // create motor #1,
64KHz pwm
AF_DCMotor motor2(2, MOTOR12_64KHZ); // create motor #2,
64KHz pwm
AF_DCMotor motor3(3, MOTOR12_64KHZ); // create motor #3,
64KHz pwm
AF_DCMotor motor4(4, MOTOR12_64KHZ); // create motor #4,
64KHz pwm

int turn_Speed = 175; // motor speed when using the compass
to turn left and right
int mtr_Spd = 250; // motor speed when moving forward
and reverse
```

```

//*****
//*****
// Compass Variables & Setup

HMC5883L compass; // HMC5883L compass(HMC5883L)
int16_t mx, my, mz; // variables to store x,y,z axis from
compass (HMC5883L)
int desired_heading; // initialize variable - stores value for
the new desired heading
int compass_heading; // initialize variable - stores value
calculated from compass readings
int compass_dev = 5; // the amount of deviation that is
allowed in the compass heading - Adjust as Needed
// setting this variable too low will cause the
robot to continuously pivot left and right
// setting this variable too high will cause the
robot to veer off course

int Heading_A; // variable to store compass heading
int Heading_B; // variable to store compass heading in
Opposite direction
int pass = 0; // variable to store which pass the robot
is on

//*****
//*****
// Servo Control

Servo myservo; // create servo object to control a servo
int pos = 0; // variable to store the servo position

//*****
//*****
// Ping Sensor for Collision Avoidance

boolean pingOn = false; // Turn Collision detection On or
Off

int trigPin = 43; // Trig - Orange
int echoPin = 42; // Echo - Yellow
long duration, inches;
int Ping_distance;

unsigned long currentMillis = 0;
unsigned long previousMillis = 0; // Store last time Ping was
updated
const long interval = 200; // Ping the Distance every X
milliseconds

```

```

//*****
//*****

// Bluetooth Variables & Setup

String str;                                // raw string received from android to
arduino
int blueToothVal;                          // stores the last value sent over via
bluetooth
int bt_Pin = 34;                          // Pin 34 of the Aruino Mega used to
test the Bluetooth connection status - Not Used

//*****
//*****

// GPS Locations

unsigned long Distance_To_Home;            // variable for storing the
distance to destination

int ac = 0;                               // GPS array counter
int wpCount = 0;                          // GPS waypoint counter
double Home_LATarray[50];                // variable for storing the
destination Latitude - Only Programmed for 5 waypoint
double Home_LONarray[50];                // variable for storing the
destination Longitude - up to 50 waypoints

int increment = 0;

void setup()
{
  Serial.begin(115200);                   // Serial 0 is for communication with
the computer
  Serial1.begin(9600);                    // Serial 1 is for Bluetooth communication
- DO NOT MODIFY - JY-MCU HC-06 v1.40
  Serial2.begin(9600);                    // Serial 2 is for GPS communication
at 9600 baud - DO NOT MODIFY - Ublox Neo 6m
  myservo.attach(9);                      // attaches the servo to pin 9 (Servo 0
on the Adafruit Motor Control Board)

  pinMode(36, OUTPUT);                    // define pin 36 as an output for
an LED indicator - Not Used
  pinMode(bt_Pin, INPUT);                 // This pin(34) is used to check
the status of the Bluetooth connection - Not Used

  // Ping Sensor
  pinMode(trigPin, OUTPUT);               // Ping Sensor
  pinMode(echoPin, INPUT);                // Ping Sensor

```

```

// Compass
Wire.begin(); // Join I2C bus used for the HMC5883L
compass
compass.begin(); // initialize the compass (HMC5883L)
compass.setRange(HMC5883L_RANGE_1_3GA); // Set measurement
range
compass.setMeasurementMode(HMC5883L_CONTINUOUS); // Set
measurement mode
compass.setDataRate(HMC5883L_DATARATE_30HZ); // Set data rate
compass.setSamples(HMC5883L_SAMPLES_8); // Set number of
samples averaged
compass.setOffset(0,0); // Set calibration offset

Startup(); // Run the Startup procedure on power-up
one time
}

//*****
//*****
// Main Loop

void loop()
{

bluetooth(); // Run the Bluetooth procedure to see if
there is any data being sent via BT
getGPS(); // Update the GPS location
getCompass(); // Update the Compass Heading
Ping(); // Use at your own discretion, this is not
fully tested

}

```



## REFERENCES :

1. Design of a Remote-controlled and GPS-guided Autonomous Robot for Precision Farming  
<https://journals.sagepub.com/doi/pdf/10.5772/62059>
2. An Autonomous Positioning and Navigation System for Spherical Mobile Robot  
<https://core.ac.uk/download/pdf/81194343.pdf>
3. Autonomous Robot Navigation  
in Highly Populated Pedestrian Zones  
<https://europa2.informatik.uni-freiburg.de/files/kuemmerle14jfr.pdf>
4. Autonomous Navigation for Urban Service Mobile Robots  
<http://www.iri.upc.edu/files/scidoc/1182-Autonomous-Navigation-for-Urban-Service-Mobile-Robots.pdf>
5. Research of Autonomous Navigation Strategy for an Outdoor Mobile Robot  
[http://article.nadiapub.com/IJCA/vol7\\_no12/32.pdf](http://article.nadiapub.com/IJCA/vol7_no12/32.pdf)

