



Arduino Powered GPS Motor Vehicle.

A Project Report of Capstone Project – 2

Submitted by

Saurav kumar verma

16SCSE105082

1613105109

In partial fulfillment for the award of the degree

Of

Bachelor of technology

In

Computer science and engineering with specialization of

Cloud computing and virtualization

School of computer science and engineering

Under the Supervision of

Mr C vairavel

April/ May, 2020



**SCHOOL OF COMPUTING AND SCIENCE AND
ENGINEERING**

BONAFIDE CERTIFICATE

Certified that this project report “**Aurdino powered GPS motor vehicle**” is the bonafide work of “**Saurav kumar verma (1613105109)**” who carried out the project work under my supervision.

Signature of the head of department

Signature of the supervisor

Professor & Dean,

Professor

School of Computing Science & Engineering

School of Computing Science & Engineering

Table of Contents

| | |
|--|----|
| 1. Abstract..... | 4 |
| 2. Executive Summary..... | 4 |
| 3. Introduction..... | 5 |
| 4. Integrated System..... | 6 |
| 5. Mobile Platform | 7 |
| 6. Actuation | 8 |
| 7. Sensors..... | 11 |
| 8. Navigation Algorithm..... | 26 |
| 9. MCU {Microcontroller Unit}..... | 29 |
| 10. Behaviors..... | 34 |
| 11. Experimental Layout and Results..... | 36 |
| 12. Conclusion | 37 |
| 13. Appendices..... | 38 |
| 14. Arduino IDE..... | 39 |
| 15. Actual Layout..... | 55 |
| 16. Reference..... | 57 |
| 17. Actual Coding..... | 58 |

1. Abstract

My robot is a mobile platform of any kind of exploration robot so that it can navigate through designated waypoints while trying to avoid obstacles. And finally comes back to the starting point or another place. To minimize the time and money for building a new mobile platform, it is based on commonly used scale RC model car with some modification. We can do potential future enhancements by adding a SD card for logging GPS track. We can also add a camera for taking photos and videos. The arduino board acts as a controller which help us to control the speed and change the speed. It also controls the steering of the car to achieve automatic obstacle avoidance. The vehicle's speed is controlled with the help of pulse wave modulation (PWM) provided to us by the Motor shield. GPS helps us in providing global coordinates of the current location that where the vehicle is present in real time and it also tell us that where that vehicle is heading towards. With the combination of hardware and software we can easily navigate the vehicle and guide it towards right direction.

2. Executive Summary

For many years, navigation of the robot has been the most basic and yet most challenging stuff in developing a mobile robot. It is because moving the robot to the required place is getting more important as they become mobile. For example, if hundreds of researchers developed a bomb deactivation robot with great sensors and arms but it cannot go to the target by it, how can we evaluate the robot as useful? This shows how navigation is important in developing a mobile robot. Especially for outdoor application, utilizing the GPS receiver became a significant breakthrough in developing mobile robot. With an affordable GPS receiver and better accuracy, now everyone can develop mobile robot much easier than before.

However, since the GPS is nothing more than just providing global coordinates of the current location and heading information, it is required to implement a hardware and software system that can navigate the robot. So this project will cover comprehensively from building a hardware system of mobile robot and complete navigation system with obstacle avoidance.

This article will discuss issues in developing autonomous GPS navigation based on RC car. First, I will introduce hardware components including the car, motor driver, GPS system and proximity sensors, and show how they are organized with micro controller. Then, bring some idea about the navigation and problem solving techniques. And finally analyze the result of testing and discussions.

3. Introduction

The concept of my robot starts from very basic requirement about the robot itself. How to make the robot can have mobility? Extensive development in electronics and software enabled robots in nowadays can do almost anything we can imagine and its boundary will be expanded. However most of robots don't have enough mobility thus far and even some of mobile robots have to depend on inefficient mobile platforms. First of all, they are too slow. It is obvious that 4 legs or biped cannot be developed to be more efficient and fast enough then wheeled vehicle. If we can have fully autonomous and independent vehicle then it will change our life dramatically. Primary objective on this project is designing a car based robot that can navigate through GPS coordinates automatically. The robot is based on regular RC car and entire components including microprocessor and sensors are on the same base. More details about the platform and system will be discussed first then go through each components. And will talk about behavior of the robot and experimental result followed by conclusions.

4. Integrated System

As we can imagine a driver in a car use his or her eyes to sense the road ahead and makes decision with the brain then give a command to the hand and legs to control the car, this robot has almost the same components. The robot is composed of four components as microcontroller, sensors, servos and motor.

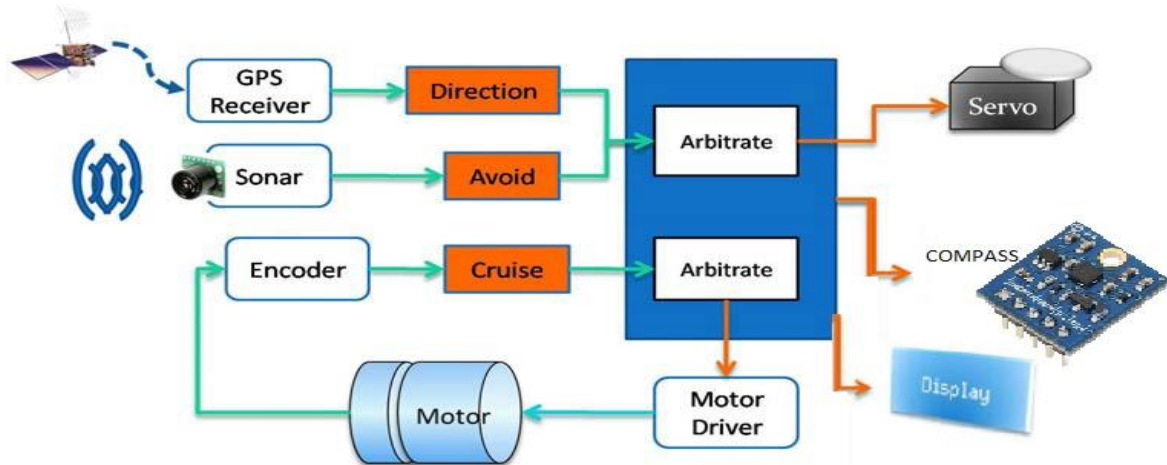


Fig 1 Behavior Control Diagram

Microcontroller, which is almost equivalent to brain of a driver, will process data, make decision and give signal to its actuators. For this project, Arduino Mega 2560 16-bit microprocessor is selected. For the sensor part, ultrasonic sensors will be attached as three in front and one in rear to detect obstacles and measure the distance from them. To make the sensors more functional and flexible, each sensor will be attached on a small stand. And GPS sensor will be placed on the top of the platform in order to prevent interference with other objects. A compass module is also used current heading detection.

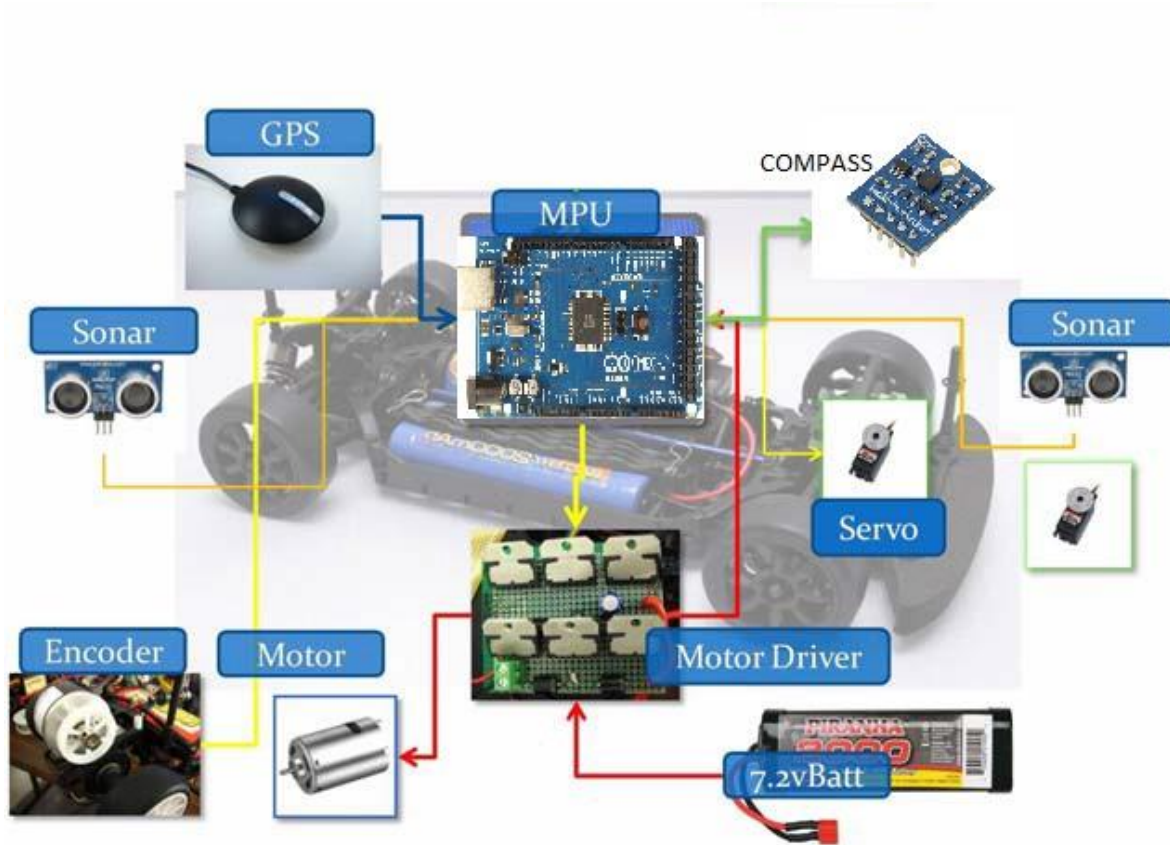


Fig 2 Overall Hardware Integration

5. Mobile Platform

Since this robot is based on a regular RC model car, physical dimension and mechanical specifications are limited to a given RC car.



| Specifications | |
|----------------|----------|
| Length | 378.6mm |
| Width | 188mm |
| Wheel base | 257mm |
| Motor | RS-540SH |
| Drive | Rear 2WD |

6. Actuation

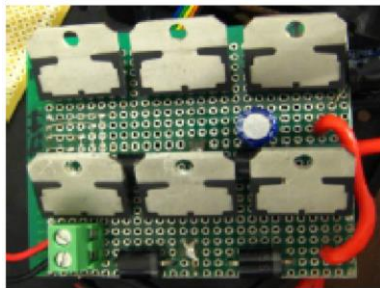
1. Motor



RS-540SH, generic brushed DC motor is attached at the rear section of the platform and produce main power to the car. DC motor especially for this kind of cheap motors are known as very difficult to control because it draws so much electricity from the circuit board that the microcontroller become unstable. To prevent such kind of harmful effect

from running motor, it is highly recommended to separate the source of electric power.

Motor Controller L293D



An array of six L293D Dual-full-bridge motor drivers is used for the motor driver. According to the datasheet of L293D, each driver can draw 4A, hence by combining them into 6, it can produce up to 24A theoretically.

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoids, DC and stepping motors) and switching power transistors. To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included. This device is suitable for use in switching applications at frequencies up to 5 kHz. The L293D is assembled in a 16 lead plastic package which has 4 center pins connected together and used for heatsinking The L293DD is assembled in a 20 lead surface mount which has 8 center pins connected together and used for heatsinking.

PID control

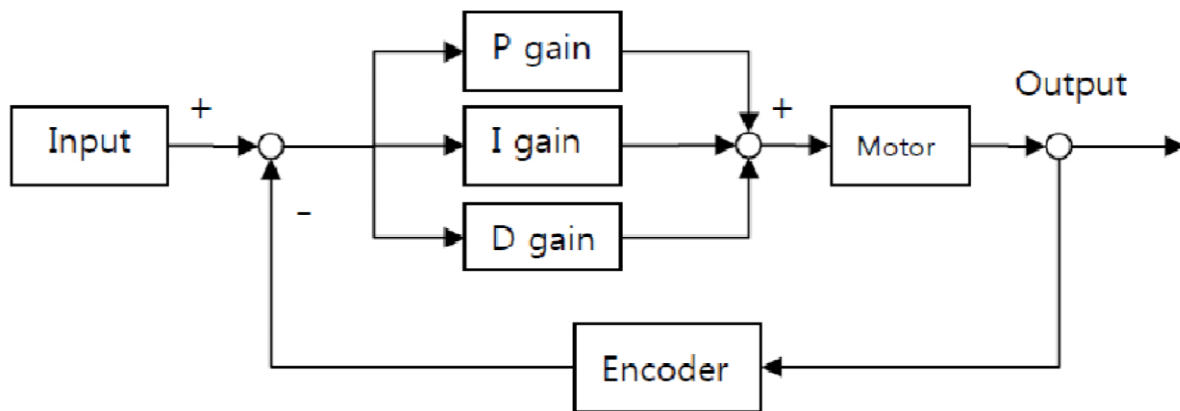


Fig 3 A block diagram of a PID controller

To control the DC motor more precisely and reliably, PID control is required. An incremental encoder with resolution of 1440 ppr(Pulse Per Revolution) is connected to the main spur gear which drive main shaft. This encoder will measure the rotational speed of the shaft by counting the pulse at a specific timing. By comparing the error between the input and output, the controller can determine if it is going to increase or decrease the power of the motor.

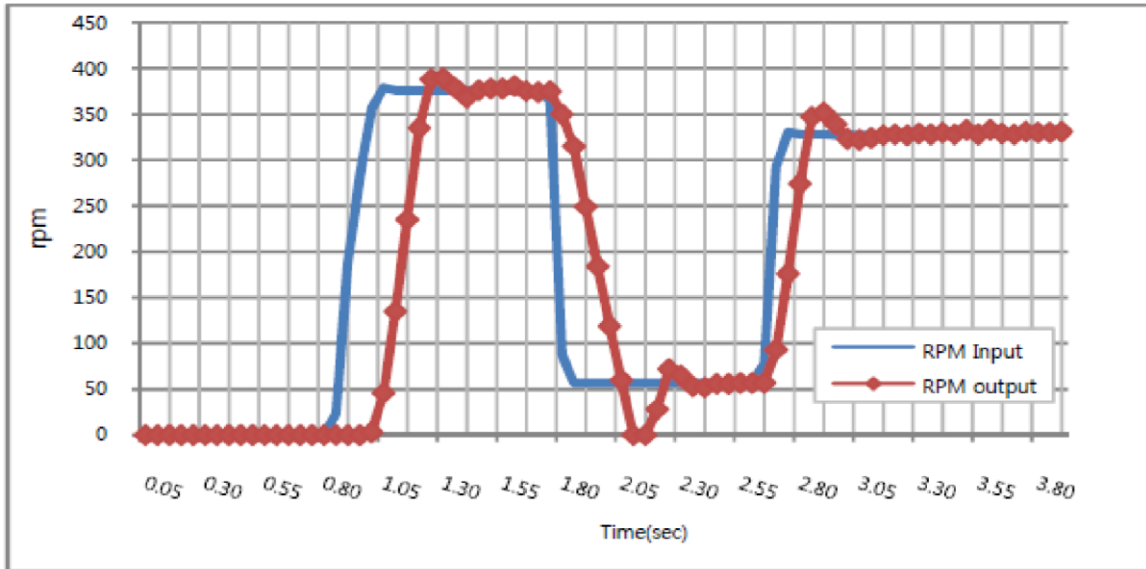


Fig 4 PID control result

2. Servo High speed HS-5925MG digital servo is mounted in front section of the car to change the direction.

7. Sensors

Some proximity sensors and encoder are used for this project. Proximity sensors are essential for obstacle avoidance and encoder for motor control. In order to minimize cost and time, each sensor is evaluated in terms of suitability, affordability and feasibility. The point of choosing sensor is about doing more with less. I tried to minimize requirement of sensors.

1. Obstacle Avoidance

Obstacle avoidance requires a robot can detect or measure the distance from the object in front or in the course. A robot or micro controller can perform maneuvers to avoid obstacles according to the measured signal from those sensors. Therefore it is easy to say that better sensor gives better outputs. This table shows some common sensors that can be used for the robot.

| Type | Distance | Accuracy | Signal | Price(EA) | Out door |
|---------------|-----------|--------------------|-------------------------------------|-------------|----------|
| IR sensor | 0.1-0.3m | | Analogue | Rs.90-200 | No |
| SONAR | 0-6.4m | $\pm 10\%$ | Analogue RS-232 PWM Timing | Rs. 100-500 | Yes |
| Bump switch | 1.0-5.0cm | | On/Off | | Yes |
| Vision Sensor | 0-10m+ | | RS-232 Digital | Rs. 1000-? | Yes |
| Laser | 0-1,000m | $\pm 1.5\text{mm}$ | Digital | Rs. 2500-? | Yes |

Some sensors like IR sensors are very common in obstacle avoidance and/or proximity measurements; however it works poorly in outdoor application. So I excluded this type of sensor.

| | IR | Sonar | Bump | Vision | Laser |
|---------------|-----------|-----------|-----------|-----------|----------|
| Suitability | 1 | 5 | 2 | 5 | 4 |
| Affordability | 5 | 4 | 5 | 3 | 1 |
| Feasibility | 5 | 5 | 5 | 2 | 2 |
| Total | 11 | 14 | 12 | 10 | 7 |

1:Very poor, 2:Poor, 3:Middle, 4:Good 5:Very good


Table 1 My sensor evaluation

- i. Ultrasonic sensor (SONAR) SONAR or Ultrasonic sensor is very common type of sensor detecting object and measuring distance. Three ultrasonic sensors are mounted in front of the car and one in rear. Front sensors are mainly used for detecting obstacles and measuring the distance between the car and the objects. To help microcontroller finding the way more easily, these sensors can be tilted around the front area.

HC-SR04 Sensor

I choose HC-SR04 for the project because this is the cheapest of all

Kind and it also provides various types of output signal including PWM, RS-232 And analogue voltage. One of the good things about using analogue output is that the output is proportional to every inch. So the user does not have to worry about handling timer interrupt and calculating distance with the flight time of ping.

|  | Specification | |
|---|----------------------|--|
| | Vcc | 5v |
| | Range | 0-6.45m |
| | Output | RS232 Serial Analogue – 10mV/in PWM – 147uS/in |
| | Price | Rs. 135 x 4 |

1. 1 Ultrasonic Definition

The human ear can hear sound frequency around 20HZ ~ 20KHZ, and ultrasonic is the sound wave beyond the human ability of 20KHZ .

1.2 Ultrasonic distance measurement principle

Ultrasonic transmitter emitted an ultrasonic wave in one direction, and started timing when it launched. Ultrasonic spread in the air, and would return immediately when it encountered obstacles on the way. At last, the ultrasonic receiver would stop timing when it received the reflected wave. As Ultrasonic spread velocity is 340m / s in the air, based on the timer record t , we can calculate the distance (s) between the obstacle and transmitter, namely: $s = 340t / 2$, which is so-called time difference distance measurement principle. The principle of ultrasonic distance measurement used the already-known air spreading velocity, measuring the time from launch to reflection when it encountered obstacle, and then calculate the distance between the transmitter and the obstacle according to the time and the velocity. Thus, the principle of ultrasonic distance measurement is the same with radar.

Distance Measurement formula is expressed as: $L = C \times T$

In the formula, L is the measured distance, and C is the ultrasonic spreading velocity in air, also, T represents time (T is half the time value from transmitting to receiving).

1.3 Ultrasonic Application

Ultrasonic Application Technology is the thing which developed in recent decades. With the ultrasonic advance, and the electronic technology development, especially as high-power semiconductor device technology matures, the application of ultrasonic has become increasingly widespread: □ Ultrasonic measurement of distance, depth and thickness;

- Ultrasonic testing;
- Ultrasound imaging;
- Ultrasonic machining, such as polishing, drilling;
- Ultrasonic cleaning;
- Ultrasonic welding;

1.3 Product Features

- Stable performance

- Accurate distance measurement
- High-density
- Small blind

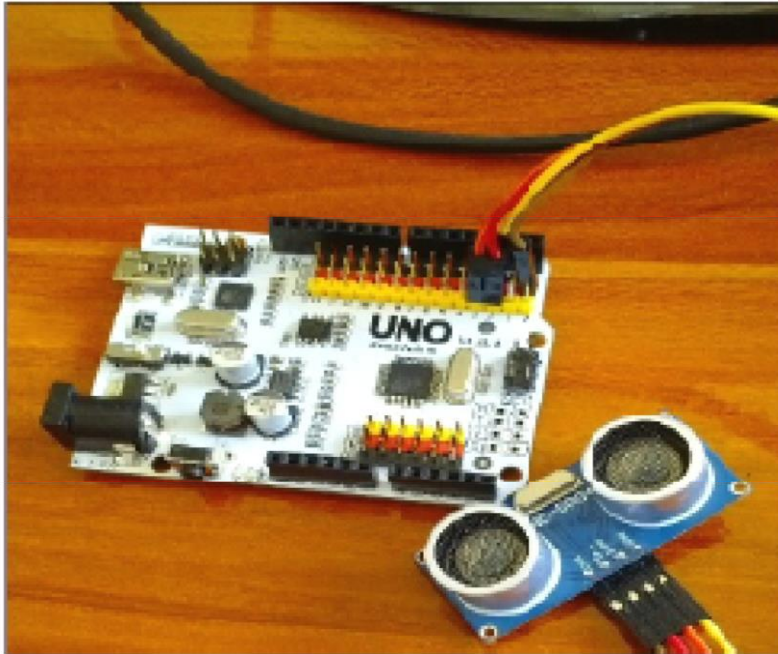
Application Areas:

- Robotics barrier
- Object distance measurement
- Level detection
- Public security
- Parking detection

1.4 Module operating Principle

Set low the Trig and Echo port when the module initializes , firstly, transmit at least 10us high level pulse to the Trig pin (module automatically sends eight 40K square wave), and then wait to capture the rising edge output by echo port, at the same time, open the timer to start timing. Next, once again capture the falling edge output by echo port, at the same time, read the time of the counter, which is the ultrasonic running time in the air. According to the formular: test distance = (high level time * ultrasonic spreading velocity in air) / 2, you can calculate the distance to the obstacle.

1.6 HC – SR04 Connection to Microcontroller



Connection Description: D2<----->Trig D3<----->Echo (The users can define the connection pin by themselves)

2. GPS Navigation

Currently there are hundreds of different kinds of GPS receivers in the market.

2.1 Overview

The NEO-6 module series is a family of stand-alone GPS receivers featuring the high performance u-blox 6 positioning engine. These flexible and cost effective receivers offer numerous connectivity options in a miniature 16 x 12.2 x 2.4 mm package. Their compact architecture and power and memory options make NEO-6 modules ideal for battery operated mobile devices with very strict cost and space constraints. The 50-channel u-blox 6 positioning engine boasts a Time-To-First-Fix (TTFF) of under 1 second. The dedicated acquisition engine, with 2 million correlators, is capable of massive parallel time/frequency space searches, enabling it to find satellites instantly. Innovative design and technology suppresses jamming sources and mitigates multipath effects, giving NEO-6 GPS receivers excellent navigation performance even in the most challenging environments.

2.2 Raw data

Raw data output is supported at an update rate of 5 Hz on the NEO-6T and NEO6P. The UBX-RXM-RAW message includes carrier phase with half-cycle ambiguity resolved, code phase and Doppler measurements, which can be used in external applications that offer precision positioning, real-time kinematics (RTK) and attitude sensing.

2.3 Automotive Dead Reckoning

Automotive Dead Reckoning (ADR) is u-blox' industry proven off-the-shelf Dead Reckoning solution for tier-one automotive customers. u-blox' ADR solution combines GPS and sensor digital data using a tightly coupled Kalman filter. This improves position accuracy during periods of no or degraded GPS signal. The NEO-6V provides ADR functionality over its software sensor interface. A variety of sensors (such as wheel ticks and gyroscope) are supported, with the sensor data received via UBX messages from the application processor. This allows for easy integration and a simple hardware interface, lowering costs. By using digital sensor data available on the vehicle bus, hardware costs are minimized since no extra sensors are required for Dead Reckoning functionality. ADR is designed for simple integration and easy configuration of different sensor options (e.g. with or without gyroscope) and vehicle variants, and is completely self-calibrating.

2.4 Precise Point Positioning u-blox' industry proven PPP algorithm provides extremely high levels of position accuracy in static and slow moving applications, and makes the NEO-6P an ideal solution for a variety of high precision applications such as surveying, mapping, marine, agriculture or leisure activities.

Ionospheric corrections such as those received from local SBAS12 geostationary satellites (WAAS, EGNOS, MSAS) or from GPS enable the highest positioning accuracy with the PPP algorithm. The maximum improvement of positioning accuracy is reached with PPP+SBAS and can only be expected in an environment with unobstructed sky view during a period in the order of minutes.

2.5 Oscillators

NEO-6 GPS modules are available in Crystal and TCXO versions. The TCXO allows accelerated weak signal acquisition, enabling faster start and reacquisition times.

2.6 Protocols and Interfaces

Protocol Type

NMEA Input/output, ASCII,
0183, 2.3 (compatible to
3.0)

UBX Input/output, binary, u-

blox proprietary

RTCM Input, 2.3

2.7 Display Data Channel (DDC)

The I2C compatible DDC interface can be used either to access external devices with a serial interface EEPROM or to interface with a host CPU. It is capable of master and slave operation. The DDC interface is I2C Standard Mode compliant. For timing parameters consult the I2C standard.

The DDC Interface supports serial communication with u-blox wireless modules.

See the specification of the applicable wireless module to confirm compatibility. The maximum bandwidth is 100kbit/s.

2.7.1 External serial EEPROM

NEO-6 modules allow an optional external serial EEPROM to be connected to the DDC interface. This can be used to store Configurations permanently.

For more information see the LEA-6/NEO-6/MAX-6 Hardware Integration Manual [1].

Use caution when implementing since forward compatibility is not guaranteed.

2.8 Power Management u-blox receivers support different power modes. These modes represent strategies of how to control the acquisition and tracking engines in order to achieve either the best possible performance or good performance with reduced power consumption.

For more information about power management strategies, see the u-blox 6 Receiver Description including Protocol Specification [2].

2.8.1 Maximum Performance Mode

During a Cold start, a receiver in Maximum Performance Mode continuously deploys the acquisition engine to search for all satellites. Once the receiver has a position fix

(or if pre-positioning information is available), the acquisition engine continues to be used to search for all visible satellites that are not being tracked.

2.8.2 Eco Mode

During a Cold start, a receiver in Eco Mode works exactly as in Maximum Performance Mode. Once a position can be calculated and a sufficient number of satellites are being tracked, the acquisition engine is powered off resulting in significant power savings. The tracking engine continuously tracks acquired satellites and acquires other available or emerging satellites.

Note that even if the acquisition engine is powered off, satellites continue to be acquired.

2.8.3 Power Save Mode

Power Save Mode (PSM) allows a reduction in system power consumption by selectively switching parts of the receiver on and off.

Power Save mode is not available with NEO-6P, NEO-6T and NEO-6V.

2.9 Configuration

2.9.1 Boot-time configuration

NEO-6 modules provide configuration pins for boot-time configuration. These become effective immediately after start-up. Once the module has started, the configuration settings can be modified with UBX configuration messages. The modified settings remain effective until power-down or reset. If these settings have been stored in battery-backup RAM, then the modified configuration will be retained, as long as the backup battery supply is not interrupted.

GPS Receiver



NEO-6

u-blox 6 GPS Modules

| | |
|--------------------------|---|
| Chipset | NEO-6M-0-001 50 CHANNELS, WAAS,EGNOS,MSAS |
| Accuracy Position | 10 meters, 2D RMS 5 meters, 2D RMS, WAAS enabled |
| Reacquisition | 0.1 sec., average |
| Power consumption | 80mA |
| Baud rate | 4,800 bps |
| Main power input | 4.5V ~ 6.5V DC input |
| Output message | NMEA 0183 GGA, GSA, GSV, RMC, VTG, GLL |
| Physical Characteristics | Dimension: . 16.0 x 12.2 x 2.4 mm |

GPS data analysis

PARSE GPRMC DATA

```
$GPRMC,194205.000,A,2938.5724,N,08220.8584,W,0.37,78.34,010308,.,*21
      (1)   (2)   (3)   (4)   (5)   (6) (7) (8)   (9)
```

(1) Current Time: hhmmss.ss

(2) Validity: A- AOK,

(3) Latitude: DDMM.SS.ss

(4) North South

(5) Longitude: DDMM.SS.ss

(6) East West

(7) Speed : Knots

(8) Heading: 0, 360: north (9) Date today

Reading and Parsing the Data

Processing the GPS data is composed of two parts as receiver and parser. Receiver is simply an interrupt service routine that stores data into certain variable. Since

the interrupt is set very quickly, the code should be as small as possible to avoid delayed interrupt problem.

Every time the controller receives a character through a SCI channel, an interrupt is set and store it to a global variable. If it finds a character '\$' which means the starting of GPS code, then the controller compare next 5 characters with 'GPRMC' which I'm looking for. If the data contains GPRMC code, a flag is set and the main program calls the Parsing function.

```
////////////////////////////////////GPS MODULE////////////////////////////////////
//Function for gps parsing
void processGPS(void)
{
  // in case you are not using the interrupt above, you'll
  // need to 'hand query' the GPS, not suggested :(

  if (! usingInterrupt) {
    // read data from the GPS in the 'main loop'

    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
    if (GPSECHO)
      if (c) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {

    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA()); // this also sets the newNMEAreceived() flag to false

    if (!GPS.parse(GPS.lastNMEA())) // this also sets the newNMEAreceived() flag to false
      return; // we can fail to parse a sentence in which case we should just wait for another
  }

  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis()) timer = millis();

  // approximately every 2 seconds or so, print out the current stats
  if (millis() - timer > 2000) {
    timer = millis(); // reset the timer

    Serial.print("Fix: "); Serial.print((int)GPS.fix);
    Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
  }
}
```

```

Serial.print("Fix: "); Serial.print((int)GPS.fix);
Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
if (GPS.fix) {
  Serial.print("Location (in degrees, works with Google Maps): ");
  Serial.print(GPS.latitudeDegrees, 4);
  Serial.print(", ");
  Serial.println(GPS.longitudeDegrees, 4);
  Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
  currentLat = GPS.latitudeDegrees;
  currentLong = GPS.longitudeDegrees;

  if (GPS.lat == 'S')          // make them signed
    currentLat = -currentLat;
  if (GPS.lon == 'W')
    currentLong = -currentLong;

// update the course and distance to waypoint based on our new position
distanceToWaypoint();
courseToWaypoint();

Serial.print(" \n Current Latitude   ");
Serial.print(currentLat );
Serial.print(", "); Serial.print(" \n Current Longitude   ");
Serial.print(currentLong );
Serial.print(", \n");

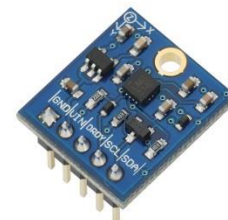
```

3. HMC5883 Magnetic Compass Module

HMC5883L

SPECIFICATIONS (* Tested at 25°C except stated otherwise.)

| Characteristics | Conditions* | Min | Typ | Max | Units |
|----------------------|--|------|-----|---------|-----------|
| Power Supply | | | | | |
| Supply Voltage | VDD Referenced to AGND | 2.16 | 2.5 | 3.6 | Volts |
| | VDDIO Referenced to DGND | 1.71 | 1.8 | VDD+0.1 | Volts |
| Average Current Draw | Idle Mode | - | 2 | - | µA |
| | Measurement Mode (7.5 Hz ODR; No measurement average, MA1:MA0 = 00) | - | 100 | - | µA |
| | VDD = 2.5V, VDDIO = 1.8V (Dual Supply) | | | | |
| | VDD = VDDIO = 2.5V (Single Supply) | | | | |
| Performance | | | | | |
| Field Range | Full scale (FS) | -8 | | +8 | µgauss |
| Mag Dynamic Range | 3-bit gain control | ±1 | | ±8 | µgauss |
| Sensitivity (Gain) | VDD=3.0V, GN=0 to 7, 12-bit ADC | 230 | | 1370 | LSB/gauss |



The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometry. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit

ADC that enables 1° to 2° compass heading accuracy. The I2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices. The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

```
//////////////////////////////////////setup for compass//////////////////////////////////////
// Initialize the serial port.

Serial.println("Starting the I2C interface.");
Wire.begin(); // Start the I2C interface.

compass = HMC5883L(); // Construct a new HMC5883 compass.

error = compass.SetScale(1.3); // Set the scale of the compass.
if(error != 0) // If there is an error, print it out.
  Serial.println(compass.GetErrorText(error));

Serial.println("Setting measurement mode to continuous.");
error = compass.SetMeasurementMode(Measurement_Continuous); // Set the measurement mode to Continuous
if(error != 0) // If there is an error, print it out.
  Serial.println(compass.GetErrorText(error));
//////////////////////////////////////
```

FEATURES

- ▶ 3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package
- ▶ 12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 mill-gauss Field Resolution in ± 8 Gauss Fields
- ▶ Built-In Self Test
- ▶ Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 μ A)
- ▶ Built-In Strap Drive Circuits
- ▶ I²C Digital Interface
- ▶ Lead Free Package Construction
- ▶ Wide Magnetic Field Range (± 8 Oe)
- ▶ Software and Algorithm Support Available
- ▶ Fast 160 Hz Maximum Output Rate

BENEFITS

- ▶ Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly
- ▶ Enables 1° to 2° Degree Compass Heading Accuracy
- ▶ Enables Low-Cost Functionality Test after Assembly in Production
- ▶ Compatible for Battery Powered Applications
- ▶ Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation
- ▶ Popular Two-Wire Serial Data Interface for Consumer Electronics
- ▶ RoHS Compliance
- ▶ Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy
- ▶ Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available
- ▶ Enables Pedestrian Navigation and LBS Applications

Fig 6.Code Snippet of Compass Setup

4. 16 X 2 LCD Screen Module

4.1. Overview

LinkSprite's LCD Shield provides a handy 16-character by 2-line display with controllable backlight. It can be plugged to an Arduino board or other Arduino shield boards. The LCD display is on the back of the shield. The board includes are panel mounting screw holes. This shield makes it easy to build a stand-alone project with its own user interface. With this shield, a computer is not required to send commands to your Arduino. The 4 direction buttons ("left", "right", "up", "down", and "select") plus the selection button allow basic user control and input. This shield

works perfectly in 4-bit mode. With the "Liquid Crystal" library included in the Arduino IDE, the shield allows control of the LCD through only six digital I/O pins. Pins D4-D9 are deliberately selected to avoid possible interference with the pins of other popular Arduino products, e.g., the Ethernet shield and the EtherTen. Therefore, this shield can be stacked on top of other shields to provide a local display.

4.2. Hardware Specification

4.2.1 Highlights of Hardware Features

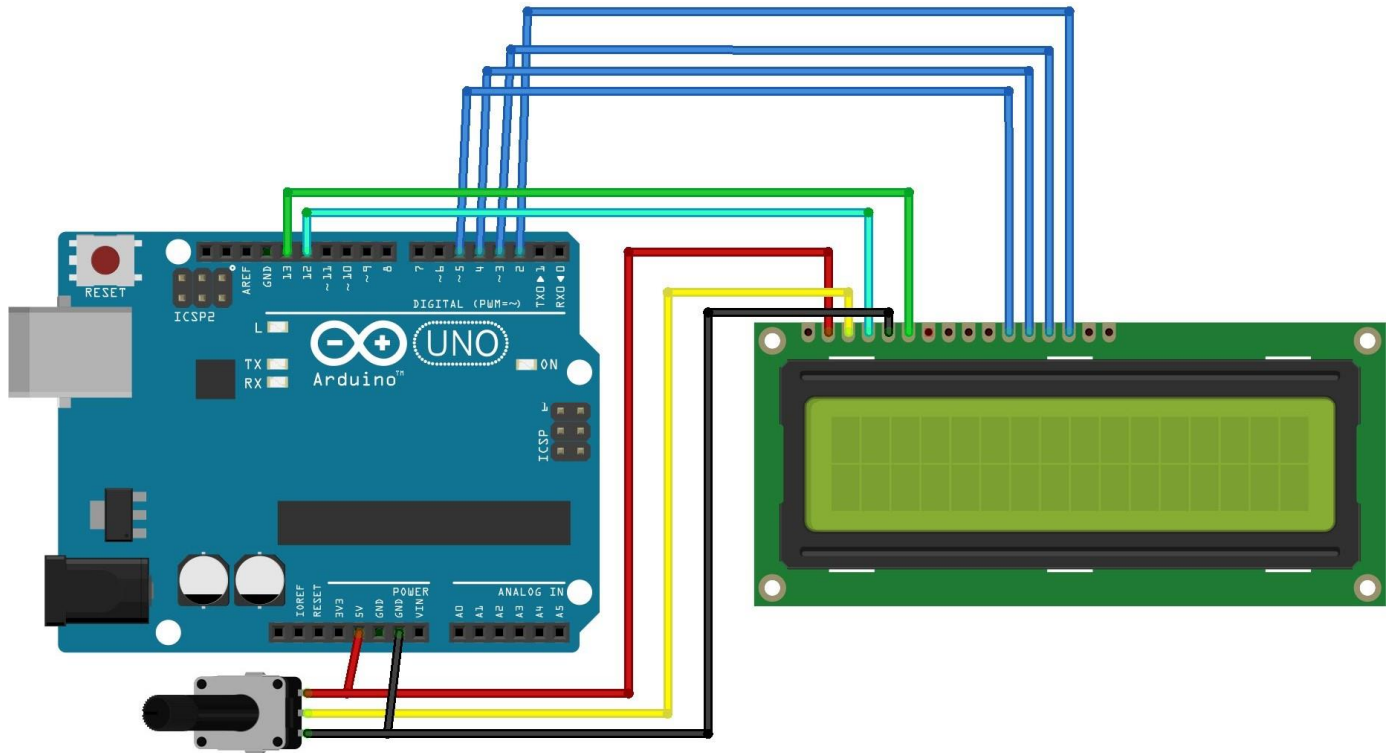
- 16x2 LCD using HD44780-compatible display module (black characters on green background).
- 5 buttons on one analog input (A0).
- LCD with current limiting, adjustable brightness, and on/off control by software through a PWM pin.
- LCD contrast adjustable with a potentiometer.
- Recessed LCD, panel mount screw holes and button layout suitable for panel or cabinet mounting.
- Reset button.
- Power supply smoothing capacitor.

4.2.2 Pin Definition

- A0: Buttons
- D4: LCD bit 4
- D5: LCD bit 5
- D6: LCD bit 6
- D7: LCD bit 7
- D8: LCD RS
- D9: LCD Enable

· D10:LCD backlight brightness adjustment

4.3 Connection to Arduino



fritzing

8. NavigationAlgorithm

1. Navigation

- i. How the navigation system works?

The idea of finding designated waypoint and following the heading was based on very simple geometry. If I know where I am and where my heading is then I can determine whether I should turn to right or left. In this project, the robot can figure out its location in the global coordinate system from GPS signal. However the GPS does not provide anything about the directional information.

This kind of navigation system is similar to how a pilot maneuvers the airplane towards the waypoint.

```
void calcDesiredTurn(void)
{
    // calculate where we need to turn to head to destination
    headingError = targetHeading - currentHeading;

    // adjust for compass wrap
    if (headingError < -180)
        headingError += 360;
    if (headingError > 180)
        headingError -= 360;

    // calculate which way to turn to intercept the targetHeading
    if (abs(headingError) <= HEADING_TOLERANCE){ // if within tolerance,
don't turn
        turnDirection = straight;
        Serial.print(" Turning Straight "); }
    else if (headingError < 0){
        Serial.print(" Turning Left ");
        turnDirection = left;}
    else if (headingError > 0){
        Serial.print(" Turning Right ");
        turnDirection = right;}
    else{
        Serial.print(" Going Straight ");
        turnDirection = straight;
    }
} // calcDesiredTurn()
```

Above code snippet shows that the compass module is used to determine the direction & calculate the heading of the vehicle on giving GPS coordinates by using Great Circle Formula and determine the shortest route to the destination co-ordinate.

| Variable Name | Description | In Code Variable |
|---------------|-----------------------------------|--------------------|
| cHDG | Current Vehicle Heading | Current Heading |
| gHDG | Goal Heading (atan2(dLat, dLong)) | Target Heading |
| dHDG | Heading Differential | Heading Error |
| LOC | Current Location | Current Lat & Long |
| gPoint | Goal Point | Target Lat & Long |

Goal Heading represents the angle between current location and next way point measured from the north pole. This can be obtained from arctangent of difference of latitude and longitude between the two.

$$gHDG = \tan^{-1} \left(\frac{dLat}{dLon} \right)$$

$$dAngle = gHDG - cHDG$$

The dAngle which is directly related to control the servo to align the robot can be calculated simply by subtracting current heading from goal heading. At first, this simple algorithm worked fine with some waypoints, however it turns out that only this process could be confusing when the target is in the north and the robot is heading toward the north. This is due to the specialty of the North pole. Simply because that the north pole can be represent either way as 0 degree or 360 degree and this makes the robot confused. These problematic cases are discussed below.

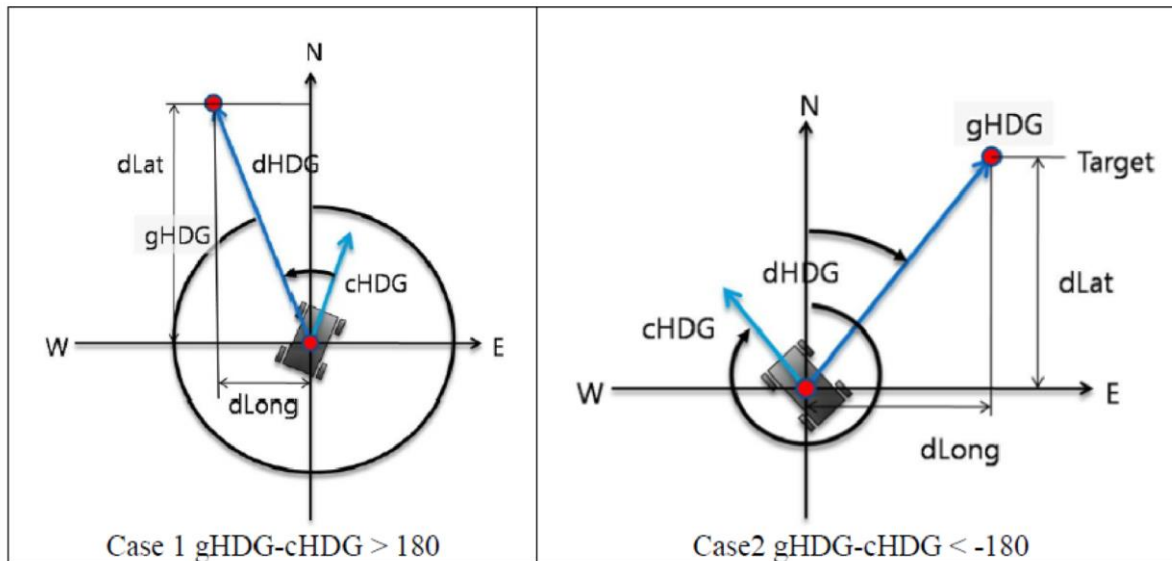


Fig 7 Exceptional Cases in GPS navigation

The first case shows when the robot is almost aligned to the target but the current angle is way smaller than the goal heading. At this moment, the robot will turn opposite direction and keep confused and never get to the target. The second case is an opposite case of the first one. In either case, the robot becomes disoriented and would never get to the target in the experiment. I solved this problem by adding some code here.

If case 1: $dHDG = gHDG - 360 - cHDG$

If case 2: $dHDG = 360 - cHDG + gHDG$

This works great with navigation and the robot wouldn't be confused again.

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

9. Microcontroller Unit

Arduino Mega 2560-R3



| | |
|-----------------------------|---|
| 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 |

started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

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 Atmega8U2 programmed as a USB-to-serial converter.

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 20-50 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 ATmega8U2 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.

I2C: 20 (SDA) and 21 (SCL). Support I2C (TWI) communication using the **Wire library** (documentation on the Wiring website). Note that these pins are not in the same location as the I2C 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 ATmega8U2 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 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 I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website 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 ATmega8U2 firmware source code is available in the Arduino repository. The ATmega8U2 is loaded with a DFU bootloader, which can be activated by connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this usercontributed 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 re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino 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).*

10. Behaviors

1. Navigation Mode

i. Initialization

Boot up the GPS module and find the current coordinate.

ii. Set the course

Load designated waypoints into the list. Select the first waypoint and turn to the direction.

iii. Go to the waypoint

Start traveling to the next waypoint. Once it gets to the waypoint, set the waypoint and the next waypoint as its new starting point and end point respectively.

Repeat until it reaches the final destination.

2. Obstacle Avoidance

i. Obstacle detection

ii. Decide whether the object is in left or right.

iii. Correct actuator signal.

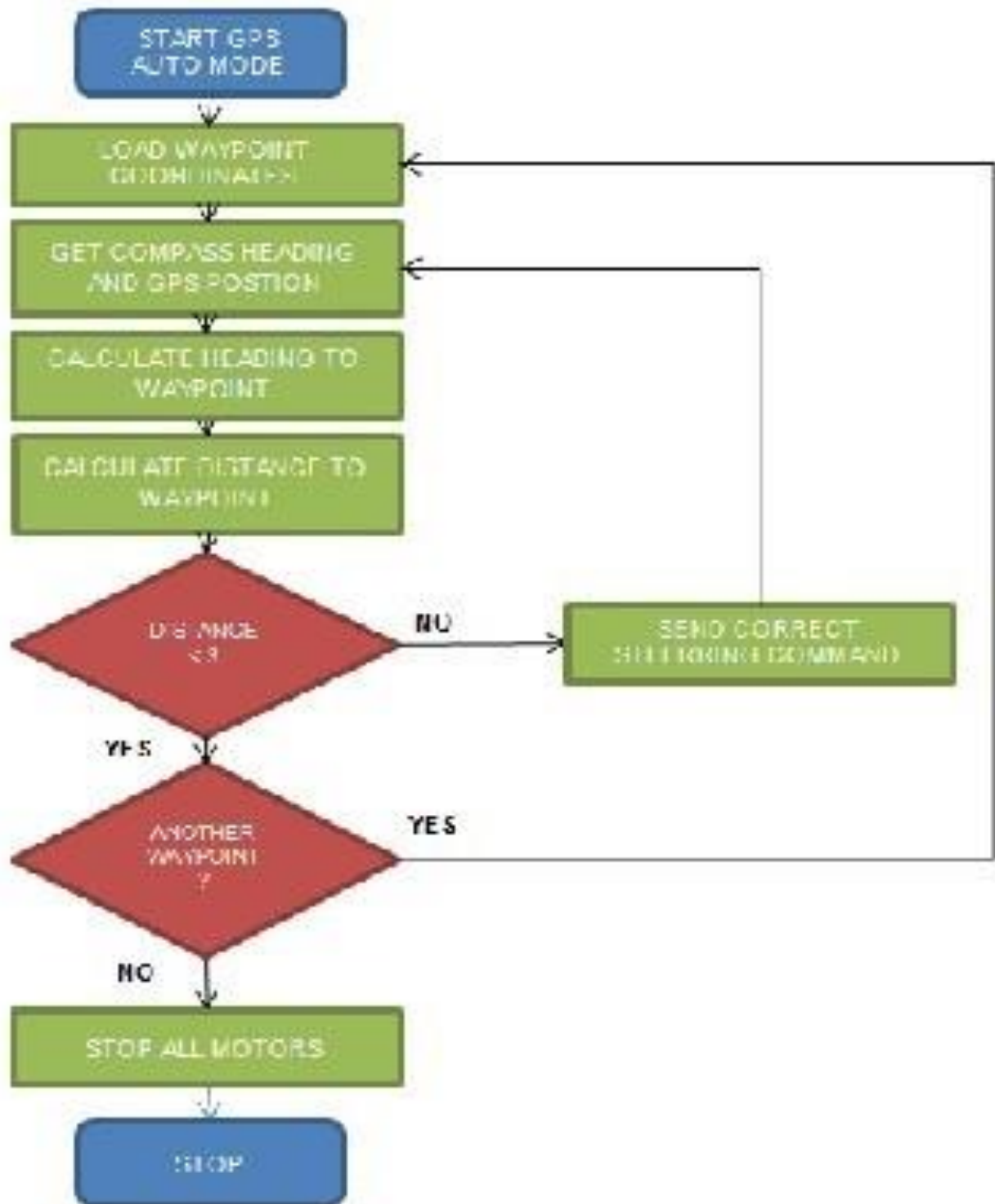


Fig 5. A block diagram of behaviors of vehicle.

11. Experimental Layouts & Results

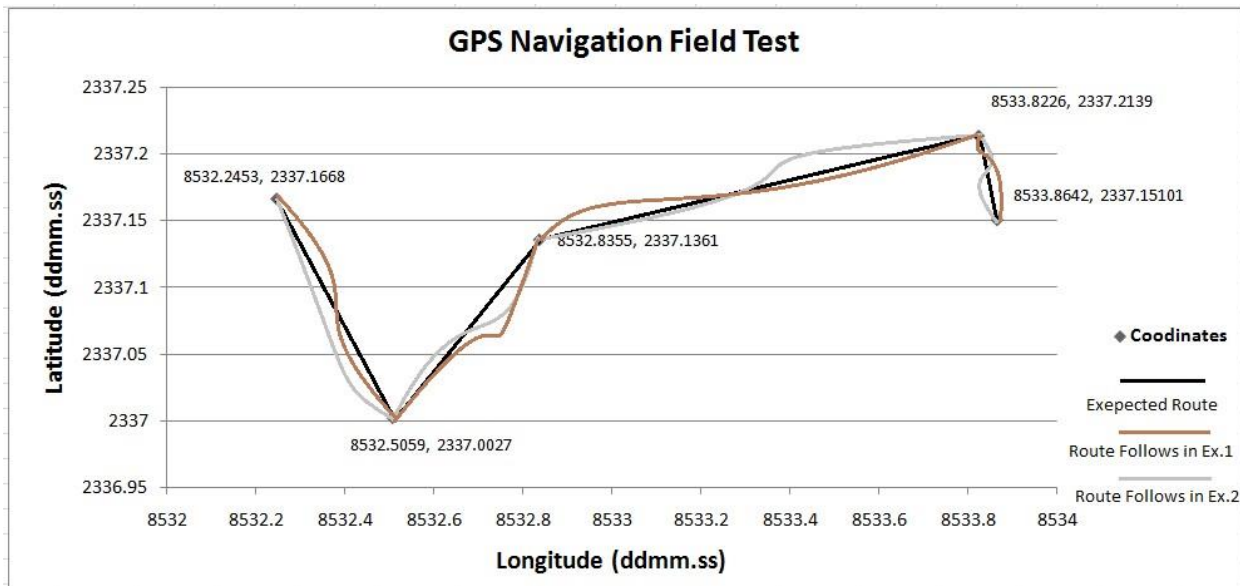


Fig 9. Experimental Data on GPS Navigation

Actual GPS navigation test conducted in one of the empty road in city of Ranchi. We conducted two experiments with same starting point and with some obstacle in the road. Each point is recorded on the externally attached logger module whenever a GPS signal received by MCU. Data is saved as a text file and processed using spreadsheet package. No filtering and smoothing has been made during data processing. Notice that a distance from the starting point to the first waypoint is approximately 75 meters.

This experiment shows how the robot accurately can go through the waypoints. Initial instability of tracking can be observed and this is caused by disorientation at the beginning part of navigation. This problem happens when the robot is stopped for a while and the GPS could not calculate orientation of the robot. I have attached a compass module to the robot helps a lot to determine orientation of the robot.

12. Conclusion

Before choosing this project, I have researched what students who took this class had previously created in the internet and it inspired me to go with GPS navigation. This decision was made simply because that nobody actually made successful robot with this theme. Another reason is that I was interested in autonomous GPS navigation. So, receiving the data would be the highest goal in my project

Processing the GPS signal and controlling the robot is much difficult than I initially anticipated. It took me about one full month just to get the reliable GPS data from the receiver. I started writing a data processing code right after receiving the GPS receiver and I thought to be almost successful but it gets the data on and off. I was almost desperate when time passed more than two weeks and almost giving up. One significant breakthrough was made when I got some suggestions and source code from a researcher in Korea who I added as a reference. Thanks to his work, I could change my crappy GPS receiving source code and finally can grab the data I needed. But this was just a beginning of the whole project. Implementing the navigation system was a big challenge and demanding work since it requires lots of actual tests. Moving the robot according to a navigation algorithm is not just simple as making software. I was literally exhausted after countless testing of the robot in the field. I had to perform the test in the campus parking lot during late night when people drive their car out of the place to avoid accident.

In sum, I was satisfied with the result that my robot performed with limited number of components. Actually this project was done with just a RC car, controller and a GPS module. This robot doesn't even have any accelerometers or gyros which is very common in mobile robot! Although I still think I should have applied digital compass I think keeping the hardware as simple as possible enabled me to accomplish this project.

13. Appendices

1. Budget

| Item | Desec. | Price | Qty | Sub Total | Remark |
|---------------------|-------------------------------------|-------|-----|-------------|--------------|
| MCU | Arduino Mega 2560R3 | 725 | 1 | 725 | Amazon.in |
| Motor Driver | Adafruit Motor 220 Controller L293D | | 1 | 220 | Amazon.in |
| GPS | u-Blox Neo-6M-0-001 | 1100 | 1 | 1100 | Snapdeal.com |
| Compass | GY-271 HMC5883L | 328 | 1 | 328 | Amazon.in |
| Ping Sensor | Adafruit HC-SR04 | 164 | 4 | 656 | Amazon.in |
| RC-Car | Cross Country* | | 1 | | Pre owned |
| Battery | Li-Po 1800mAh | 500 | 3 | 1500 | Ebay.in |
| Servo | Tower-Pro SG-90 | 200 | 1 | 200 | Amazon.in |
| Motor | Brushed RS-540SH | 175 | 1 | 175 | Amazon.in |
| Misc. Stuff | Wire, Jack, etc.* | | | | Pre owned |
| Total : | | | | 4904 | |

*Pre owned items are excluded from the budget

2. Project Schedule

Following Table shows development schedule of the project. I divided the project as Hardware part and software part. Each arrow represents the duration of each task.

| Month | | Feb | | | | Mar | | | | Apr | |
|----------|-----------|--------------------|------|------------|------------|--------------|-----------------|---------------|------|--------|---|
| week | | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 |
| Event | | Interface feedback | | | Functional | Spring Break | S Sensor Report | S Sensor Demo | Demo | | |
| Hardware | Motor | | → | | | | | | | | |
| | Sensor | Compass | → | | | Sonar | → | | | | |
| | Servo | Steer | → | | | Sonar | → | | | | |
| | GPS | | Mod* | → | | | | | | | |
| | Body | | | Design | Assembly | | | | | Tuning | |
| Software | Driver | | PID | GPS | SONAR | | | | | | |
| | Algorithm | | | Navigation | | | | O.A* | | | |
| Etc | | | | | | | | | | | |

Remarks: *Mod: Modification of the GPS module
O.A: Obstacle Avoidance

Table 3 Project schedule

14. Arduino IDE

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

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.

- Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the File | Sketchbookmenu instead.
- *Save*
Saves your sketch.
- *Serial Monitor*
Opens the [serial monitor](#).

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.

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

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](#).

To write your own library, see [this tutorial](#).

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.

For details on creating packages for third-party hardware, see the [Arduino IDE 1.5 3rd party Hardware specification](#).

Serial Monitor

Displays serial data being sent from the Arduino or Genuino board (USB or serial board). 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 that matches the rate passed to `Serial.begin` in your sketch. Note that on Windows, Mac or Linux, the Arduino or Genuino board will reset (rerun your sketch execution to the beginning) when you connect with the serial monitor.

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the [interfacing page](#) for details).

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.

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 Arduino Due, Arduino Zero, Edison, Galileo and so on.

- *Arduino Yùn*

An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.

- *Arduino/Genuino Uno*

An ATmega328 running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino Diecimila or Duemilanove w/ ATmega168* An ATmega168 running at 16 MHz with auto-reset.

- *Arduino Nano w/ ATmega328*

An ATmega328 running at 16 MHz with auto-reset. Has eight analog inputs.

- *Arduino/Genuino Mega 2560*

An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.

- *Arduino Mega*

An ATmega1280 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.

- *Arduino Mega ADK*

An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.

- *Arduino Leonardo*

An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.

- *Arduino/Genuino Micro*

An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.

- *Arduino Esplora*

An ATmega32u4 running at 16 MHz with auto-reset.

- *Arduino Mini w/ ATmega328*

An ATmega328 running at 16 MHz with auto-reset, 8 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino Ethernet*

Equivalent to Arduino UNO with an Ethernet shield: An ATmega328 running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino Fio*

An ATmega328 running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328, 6 Analog In, 14 Digital I/O and 6 PWM. □

Arduino BT w/ ATmega328

ATmega328 running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board bluetooth module, 6 Analog In, 14 Digital I/O and 6 PWM..

- *LilyPad Arduino USB*

An ATmega32u4 running at 8 MHz with auto-reset, 4 Analog In, 9 Digital I/O and 4 PWM.

- *LilyPad Arduino*

An ATmega168 or ATmega132 running at 8 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328*

An ATmega328 running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328; 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino NG or older w/ ATmega168*

An ATmega168 running at 16 MHz *without* auto-reset. Compilation and upload is equivalent to Arduino Diecimila or Duemilanove w/ ATmega168, but the bootloader burned has a slower timeout (and blinks the pin 13 LED three times on reset); 6 Analog In, 14 Digital I/O and 6 PWM.

- *Arduino Robot Control*

An ATmega328 running at 16 MHz with auto-reset.

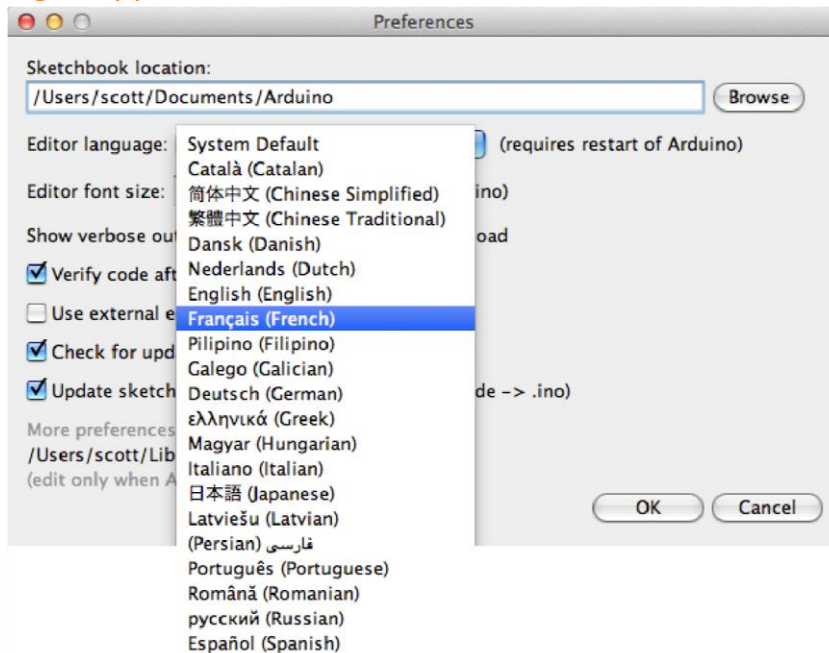
- *Arduino Robot Motor*

An ATmega328 running at 16 MHz with auto-reset.

- *Arduino Gemma*

An ATtiny85 running at 8 MHz with auto-reset, 1 Analog In, 3 Digital I/O and 2 PWM.

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.



ARDUINO 1.8.2 2017.03.22

[ide]

* Fix command line: works again with relative paths (regression)

* Fix command line: "--save-prefs" works again (regression)

* AVR toolchain has been updated with a tentative fix for the ld-returned-5-exit-

status bug

* Update arduino-builder to 1.3.25

- avoid name clashing for libraries

- cache core archives to speedup compilation consistently

* Allow BoardManager to fetch FreeBSD tools (thanks @kevans91)

* Serial monitor: the input string box is automatically focused when window is selected

* Serial monitor: now can not be opened during upload

* Serial monitor: now properly decodes UTF8 characters (thanks

@aknrdureegaesr)

* Serial monitor: added 500k, 1M and 2M baudrates (thanks @dsstutts)

* Updated RSyntaxTextArea to 2.6.1 (textarea component)

* Updated jmdsn (mDNS discovery)

* Allow plugins to attach a listener to compile progress (thanks @tomneutens) [core]

* Add Atmel-ICE and JTAGICE3 programmers for AVR chips (thanks

@matthijskooijman)

* AVR: Set unused bits of extended fuse to 1, should remove some avrdude

warning during burn bootloader (thanks @descamps)

* AVR: USB: send ZLP when needed (allows full 64 bytes packets)

* AVR: USB: use IAD descriptors instead than Generic (thanks @cuitoldfish)

[other]

* SAM platform source code has been moved to its own repository

(<https://github.com/arduino/ArduinoCore-sam>) - all PRs and issues have been

moved as well

* Update Wifi101 Firmware Updater plugin

ARDUINO 1.8.1 - 2017.01.09

[ide]

* Fixed font rendering not anti-aliased on Windows (regression)

* Increased number of colors on serial plotter to 8, thanks

@cousteaulecommandant

[libraries]

* Fixed regression in SD library. Thanks @greiman

ARDUINO 1.8.0 - 2016.12.20

[ide]

* Linux: running in command line mode doesn't require an X11 display anymore

* "Save as" now clears the "modified" status

- * builder: Paths with strange UTF8 chars are now correctly handled
- * builder: .hpp and .hh file extensions are now considered valid sketch extension
- * builder: core.a is not rebuild if not needed (improve build time in particular for big projects)
- * Fixed swapped actions "Copy for Forum" and "Copy as HTML"
- * Linux/osx: If an editor tab is a symbolic link it is no more replaced with a real file when saving (see #5478)
- * Increased the upload timeout to 5 minutes (it was 2 min, but it may be not sufficient when uploading via UART a big sketch)

[core]

* Added Arduino.org boards

* Added Adafruit Circuit Playground board

* Added "-g" option to linker to keep debug information in the .elf file (see #5539)

* avrdude: Added fake configuration for EFUSE on atmega8 part. This solves a long standing issue with "Burn bootloader".

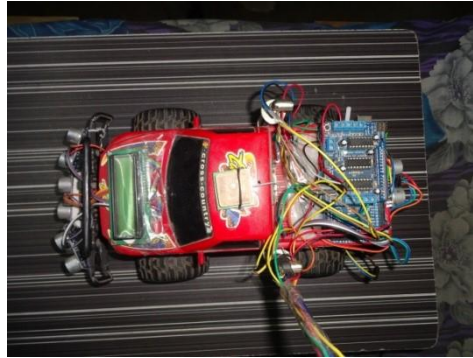
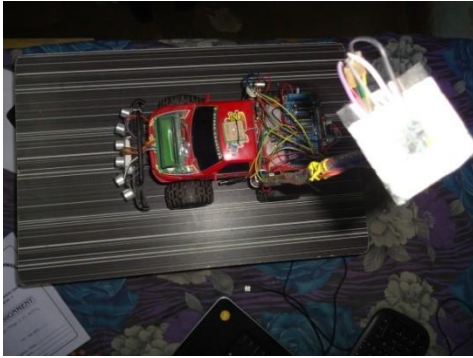
Thanks @rigelinorion, @awatterott

15. Actual Layout

Front View



Top View



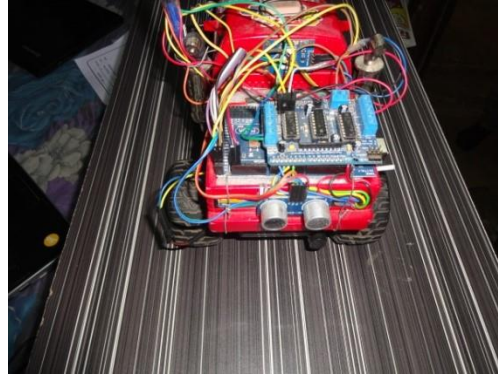
Side {Right} View



Side {Left} View



Back View



16. References

- [1]. Compass & Base Design & Calculation , By CPARKTK – Instructables ,
“<http://www.instructables.com/id/ArduinoPowered/>”
- [2]. Basic Arduino Concept , Arduino Forum ,
“<http://forum.arduino.cc/index.php?topic=165987.0/>”
- [3]. Jason , GPS Navigation,
“<http://www.ke4nyv.com/navigation.htm>”
- [4]. SlashDevin , Neo GPS Library , GitHub ,
“<http://www.github.com/slashdevin/NeoGps/>”
- [5]. Jarzelski , HMC5883L – Compass Module Library ,
“<http://www.github.com/jarzelski/arduino-HMC5883L/>”
- [6]. Great Circle Formula , Wikipedia ,
“http://en.wikipedia.org/wiki/Great_Circle_Formula”
- [7]. GPS NMEA PASER , Wikipedia ,
http://en.wikipedia.org/wiki/NMEA_0183
- [8]. Arduino Mega-2560 R3 Specs , Arduino.cc ,
<http://arduino.cc/en/main/arduinoBoardMega2560>

- [9]. u-blox GPS 6M Module Specs , u-blox.com ,
[“http://www.u-blox.com/en/product/neo-6-series/”](http://www.u-blox.com/en/product/neo-6-series/)
- [10]. L293D Motor Controller Board , Adafruit.com,
[“http://www.adafruit.com/product/807/”](http://www.adafruit.com/product/807/)
- [11]. Arduino IDE , Arduino.cc ,
[“http://www.arduino.cc/en/main/software”](http://www.arduino.cc/en/main/software)

17. Actual Coding

```
// -----
// Final Sketch for Automous Object Aviodance Gps car With Ping & Compass
// Sensor
// -----

/*
-----Pin Configuration For modules
          Pins                POWERS
GPS-                3.3V
Compass- SDA- & SCL- {ANALOG PINS}      5.0V
Ping - TRIGER- & ECHO -      5.0V
Drive MOrtor- Motor No. 2 {Motor Shield}    EXTERNAL POWER
REQURIED FOR BETTER WORKING FOR BOTH MOTOR
Turn Motor- Motor No. 4 {Motor Shield}
*/

//////////////////////////////////////Include Files//////////////////////////////////////
#include "IRremote.h"
#include <NewPing.h>      ///For ping sensor
#include <AFMotor.h>     ///For motor Sheild
// Reference the I2C Library
#include <Wire.h>
```

```

// Reference the HMC5883L Compass Library
#include <HMC5883L.h>
//LIBRARY FOR GPS
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include <math.h> // used by: GPS
#include <waypointClass.h>
#include <LiquidCrystal.h>

/////////////////////////////////////////////////////////////////
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
// Signal Pin of IR receiver to Arduino Digital Pin 33 int
receiver = 33;

/*-----( Declare objects )-----*/
IRrecv irrecv(receiver); // create instance of 'irrecv' decode_results
results; // create instance of 'decode_results'

/////////////////////////////////////////////////////////////////
//definition of motor/////////////////////////////////////////////////////////////////
AF_DCMotor motor(2);
AF_DCMotor motork(4);
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////Definition of ping sensor/////////////////////////////////////////////////////////////////
#define TRIGGER_PIN 22 // Arduino pin tied to trigger pin on the ultrasonic
sensor.
#define ECHO_PIN 23// Arduino pin tied to echo pin on the ultrasonic sensor.
#define TigR 24
#define EchR 25
#define TigL 26
#define EchL 27
#define TigB 28

```

```

#define EchB 29
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in
centimeters). Maximum sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing
setup of pins and maximum distance.
NewPing sonar_RT(TigR      ,      EchR , MAX_DISTANCE);
NewPing sonar_LT(TigL ,      EchL , MAX_DISTANCE);
NewPing sonar_BK(TigB      ,      EchB , MAX_DISTANCE);

/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////Extra Configuration/////////////////////////////////////////////////////////////////
// Object avoidance distances (in inches)
#define SAFE_DISTANCE 70
#define TURN_DISTANCE 21
#define STOP_DISTANCE 12
#define SSD 70
#define TURN_LEFT BACKWARD
#define TURN_RIGHT FORWARD
#define TURN_STRAIGHT RELEASE

// Speeds (range: 0 - 255)
#define FAST_SPEED 150
#define NORMAL_SPEED 125
#define TURN_SPEED 100 #define
SLOW_SPEED 75
int speed = NORMAL_SPEED;
int sonarDistance, SR, SL, BckDist = 0;

// Steering/turning
enum directions {left = TURN_LEFT, right = TURN_RIGHT, straight =
TURN_STRAIGHT} ;
directions turnDirection = straight;

```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////Configr. For GPS  
System////////////////////////////////////////////////////////////////
```

```
// If you're using the Adafruit GPS shield, change  
// SoftwareSerial mySerial(30, 31); -> SoftwareSerial mySerial(8, 7);  
// and make sure the switch is set to SoftSerial  
// If using software serial, keep this line enabled  
// (you can change the pin numbers to match your wiring):
```

```
SoftwareSerial mySerial(31, 30);  
Adafruit_GPS GPS(&mySerial);  
float currentLat,    currentLong,  
targetLat,    targetLong;  
int distanceToTarget,    // current distance to target (current waypoint)  
originalDistanceToTarget; // distance to original waypoing when we started  
navigating to it
```

```
// Waypoints  
#define WAYPOINT_DIST_TOLERANE 5 // tolerance in meters to waypoint;  
once within this tolerance, will advance to the next waypoint  
#define NUMBER_WAYPOINTS 5 // enter the numebr of way points here  
(will run from 0 to (n-1)) int waypointNumber = -1; // current waypoint  
number; will run from 0 to (NUMBER_WAYPOINTS -1); start at -1 and gets  
initialized during setup() waypointClass  
waypointList[NUMBER_WAYPOINTS] = {  
waypointClass(23.371809,85.323378), waypointClass(24.037528,84.058476 ),  
waypointClass(24.037387,84.058550), waypointClass(24.037092,84.058844),  
waypointClass(24.038089, 84.059084)};
```

```
// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console //  
Set to 'true' if you want to debug and listen to the raw GPS sentences.
```

```
#define GPSECHO true
```

```

// this keeps track of whether we're using the interrupt //
off by default!
boolean usingInterrupt = false; void useInterrupt(boolean); // Func
prototype keeps Arduino 0023 happy
/////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////confi for compass/////////////////////////////////////////////////////////////////

// Store our compass as a variable.
HMC5883L compass;
// Record any errors that may occur in the compass.
int error = 0;

// Compass navigation
int targetHeading = 0;      // where we want to go to reach current waypoint
int currentHeading;        // where we are actually facing now int
headingError;              // signed (+/-) difference between targetHeading and
currentHeading
#define HEADING_TOLERANCE 5 // tolerance +/- (in degrees) within which
we don't attempt to turn to intercept targetHeading

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////Setup
programs/////////////////////////////////////////////////////////////////

void setup() {

    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
    lcd.print("Welcome!");
    //Setup IR Remote
    Serial.println("IR Receiver Button Decode");
    irrecv.enableIRIn(); // Start the receiver

```

```

////////////////////////////////////// // Setup for
motor//////////////////////////////////////
  Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.
  // turn on motor
  motor.setSpeed(200);
  motork.setSpeed(255);
  Serial.println("Motor test!");
  motor.run(RELEASE);
  motork.run(RELEASE);
  ////////////////////////////////////////End-of motor//////////////////////////////////////
  ////////////////////////////////////////Steup. For GPS
  System//////////////////////////////////////
  Serial.println("Adafruit GPS library basic test!");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix
  data) including altitude

  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);

  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or
  RMC+GGA since
  // the parser doesn't care about other sentences at this time

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // 1 Hz update rate

  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

```

```

// Request updates on antenna status, comment out to keep quiet
GPS.sendCommand(PGCMD_ANTENNA);

// the nice thing about this code is you can have a timer0 interrupt go off
// every 1 millisecond, and read data from the GPS for you. that makes the
// loop code a heck of a lot easier!
useInterrupt(true);

delay(1000);

// Ask for firmware version
mySerial.println(PMTK_Q_RELEASE);

//
// get initial waypoint; also sets the distanceToTarget and courseToTarget
variables
// nextWaypoint();
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////setup for
compass////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// // Initialize
the serial port.

Serial.println("Starting the I2C interface.");
Wire.begin(); // Start the I2C interface.

compass = HMC5883L(); // Construct a new HMC5883 compass.

error = compass.SetScale(1.3); // Set the scale of the compass.
if(error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));

Serial.println("Setting measurement mode to continuous.");
error = compass.SetMeasurementMode(Measurement_Continuous); // Set the
measurement mode to Continuous if(error != 0) // If there is an error, print it
out.
Serial.println(compass.GetErrorText(error));

```



```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////END OF
SETUP////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////EXTRA GPS CONFIGRATION////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();

  // if you want to debug, this is a good time to do it!
#ifdef UDR0  if
(GPSECHO)
  if (c) UDR0 = c;

  // writing direct to UDR0 is much much faster than Serial.print

  // but only one character can be written at a time.
#endif
}

void useInterrupt(boolean v) {
if (v) {
  // Timer0 is already used for millis() - we'll just interrupt somewhere
  // in the middle and call the "Compare A" function above

  OCR0A = 0xAF;
  TIMSK0 |= _BV(OCIE0A);
usingInterrupt = true;
} else {

  // do not call the interrupt function COMPA anymore
  TIMSK0 &= ~_BV(OCIE0A);
  usingInterrupt = false;
}
}
}

```

```

uint32_t timer = millis();

/////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////// Main Loop of
Sketch/////////////////////////////////////////////////////////////////

void loop() {

  lcd.setCursor(0, 0); lcd.print("Aquiring-GPS...");
  lcd.setCursor(1, 0); lcd.print("Please-Wait....");
  processGPS(); if (GPS.fix) { lcd.setCursor(0, 0);
  lcd.print("GPS-Aquried....."); if (irrecv.decode(&results)) //
  have we received an IR signal?
  {
    lcd.setCursor(0, 0);
    lcd.print("Engaging-IR....");
    lcd.setCursor(1, 0); lcd.print("Please-
    Wait.."); translateIR();
    irrecv.resume(); // receive the next value
  } else{ lcd.setCursor(0,
  0); lcd.print("Enaging-
  AutoCont"); lcd.setCursor(1,
  0); lcd.print("Starting-Navgt..");
  // navigate
  currentHeading = readCompass(); // get our current heading
  calcDesiredTurn(); // calculate how we would optimatally turn, without
  regard to obstacles

  checkSonar();
  moveAndAvoid();
  updateLcd();

  }
}
}
/////////////////////////////////////////////////////////////////

```

```
////////////////////////////////////Update-Lcd////////////////////////////////////
```

```
void updateLcd() {  
    // set the cursor to column 0, line 1  
    // (note: line 1 is the second row, since counting begins with 0):  
    lcd.setCursor(0, 0);  
    lcd.print("TaD:");  
    lcd.setCursor(0, 4);  
    lcd.print(distanceToTarget);  
    lcd.setCursor(0, 7);  
    lcd.print(" ");  
    lcd.setCursor(0, 8);  
    lcd.print("TrH:");  
    lcd.setCursor(0, 12);  
    lcd.print(currentHeading);  
    lcd.setCursor(1, 0);  
    lcd.print("TrD:");  
    lcd.setCursor(1, 4);  
    lcd.print(turnDirection);  
    lcd.setCursor(1, 7);  
    lcd.print(" ");  
    lcd.setCursor(1, 8);  
    lcd.print("ObD:");  
    lcd.setCursor(1, 12);  
    lcd.print(sonarDistance);  
}
```

```
////////////////////////////////////
```

```
////////////////////////////////////Function-For-  
IRRemote////////////////////////////////////
```

```
/*-----( Function )-----*/
```

```
void translateIR() // takes action based on IR code received
```

```
// describing Remote IR codes
```

```
{
```

```
    switch(results.value)
```

```

{

  case 0xFF629D: Serial.println(" FORWARD"); motor.run(FORWARD);
  motork.run( straight); break; case 0xFF22DD: Serial.println(" LEFT");
  motor.run(FORWARD);          motork.run(left);
  break; case 0xFF02FD: Serial.println(" -OK-");   break; case 0xFFC23D:
  Serial.println(" RIGHT"); motor.run(FORWARD);
  motork.run(right); break; case 0xFFA857: Serial.println(" REVERSE");
  motor.run(BACKWARD);          motork.run( straight); break; default:
  Serial.println(" other button ");

  }// End Case
  delay(500); // Do not get immediate repeat
}

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////GPS MODULE////////////////////////////////////////////////////////////////

```

```

//Function for gps parsing void

```

```

processGPS(void)

```

```

{

```

```

  // in case you are not using the interrupt above, you'll

```

```

  // need to 'hand query' the GPS, not suggested :(

```

```

  if (! usingInterrupt) {

```

```

    // read data from the GPS in the 'main loop'

```

```

    char c = GPS.read();

```

```

    // if you want to debug, this is a good time to do it!

```

```

    if (GPSECHO)

```

```

      if (c) Serial.print(c);

```

```

    }

```

```

  // if a sentence is received, we can check the checksum, parse it...

```

```

  if (GPS.newNMEAreceived()) {

```

```

    // a tricky thing here is if we print the NMEA sentence, or data
// we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trying to print out data
//Serial.println(GPS.lastNMEA()); // this also sets the newNMEAreceived()
flag to false

    if (!GPS.parse(GPS.lastNMEA())) // this also sets the newNMEAreceived() flag
to false    return; // we can fail to parse a sentence in which case we should just
wait for another
}

// if millis() or timer wraps around, we'll just reset it
if (timer > millis()) timer = millis();

// approximately every 2 seconds or so, print out the current stats
if (millis() - timer > 2000) {
    timer = millis(); // reset the timer

    Serial.print("Fix: "); Serial.print((int)GPS.fix);
    Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
if (GPS.fix) {
    Serial.print("Location (in degrees, works with Google Maps): ");
    Serial.print(GPS.latitudeDegrees, 4);
    Serial.print(", ");
    Serial.println(GPS.longitudeDegrees, 4);
    Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
    currentLat = GPS.latitudeDegrees;
    currentLong = GPS.longitudeDegrees;

    if (GPS.lat == 'S') // make them signed
currentLat = -currentLat;    if (GPS.lon == 'W')
        currentLong = -currentLong;

    // update the course and distance to waypoint based on our new position
distanceToWaypoint();
    courseToWaypoint();
}
}

```

```

    Serial.print(" \n Current Latitude  ");
    Serial.print(currentLat );
    Serial.print(", "); Serial.print(", \n Current Longitude  ");
    Serial.print(currentLong );
    Serial.print(", \n");
}
}
}

/* converts lat/long from Adafruit degree-minute format to decimal-degrees;
requires <math.h> library double convertDegMinToDecDeg (float degMin)
{ double min =
0.0;
double decDeg = 0.0;

//get the minutes, fmod() requires double
min = fmod((double)degMin, 100.0);

//rebuild coordinates in decimal degrees
degMin = (int) ( degMin / 100 );
decDeg = degMin + ( min / 60 );

return decDeg;
}*/

void nextWaypoint(void)
{
waypointNumber++; targetLat =
waypointList[waypointNumber].getLong(); targetLong
= waypointList[waypointNumber].getLat();

if ((targetLat == 0 && targetLong == 0) || waypointNumber >=
NUMBER_WAYPOINTS) // last waypoint reached?

```

```

    {
        motor.run(RELEASE); // make sure we stop
    motork.run(RELEASE);
        Serial.println("* LAST WAYPOINT *");
        loopForever();
    }

    processGPS(); distanceToTarget = originalDistanceToTarget =
distanceToWaypoint(); courseToWaypoint();

} // nextWaypoint()

// returns distance in meters between two positions, both specified //
// as signed decimal-degrees latitude and longitude. Uses great-circle
// distance computation for hypothetical sphere of radius 6372795 meters.
// Because Earth is no exact sphere, rounding errors may be up to 0.5%.
// copied from TinyGPS library int
distanceToWaypoint()
{

    float delta = radians(currentLong - targetLong);
    float sdlong = sin(delta); float cdlong =
cos(delta); float lat1 = radians(currentLat);
    float lat2 = radians(targetLat); float slat1 =
sin(lat1); float clat1 = cos(lat1);
    float slat2 = sin(lat2); float clat2 = cos(lat2);
    delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
    delta = sq(delta); delta += sq(clat2 * sdlong);

    delta = sqrt(delta); float denom = (slat1 * slat2) + (clat1
* clat2 * cdlong); delta = atan2(delta, denom);
    distanceToTarget = delta * 6372795;

    // check to see if we have reached the current waypoint
    if (distanceToTarget <= WAYPOINT_DIST_TOLERANE)
nextWaypoint();

```

```

Serial.print(", \n Printing From DistanceToWayPt. ");
Serial.print(distanceToTarget);

return distanceToTarget; } //
distanceToWaypoint()

// returns course in degrees (North=0, West=270) from position 1 to position 2, //
both specified as signed decimal-degrees latitude and longitude.
// Because Earth is no exact sphere, calculated course may be off by a tiny fraction.
// copied from TinyGPS library int
courseToWaypoint()
{
float dlon = radians(targetLong-currentLong);
float cLat = radians(currentLat); float tLat =
radians(targetLat); float a1 = sin(dlon) *
cos(tLat); float a2 = sin(cLat) * cos(tLat) *
cos(dlon); a2 = cos(cLat) * sin(tLat) - a2; a2
= atan2(a1, a2); if (a2 < 0.0)
{
a2 += TWO_PI;
}

Serial.print(", \n Printing From CourseToWayPt. ");
Serial.print(degrees(a2));

targetHeading = degrees(a2);
return targetHeading;

} // courseToWaypoint()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////Module For
Compass////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Output the data down the serial port.
void Output(MagnetometerRaw raw, MagnetometerScaled scaled, float heading,
float headingDegrees)

```



```

{
  Serial.print(" \tHeading:\t");
  Serial.print(heading);
  Serial.print(" Radians \t");
  Serial.print(headingDegrees);
  Serial.println(" Degrees \t");
}

int readCompass(void)
{

  // Retrieve the raw values from the compass (not scaled).
  MagnetometerRaw raw = compass.ReadRawAxis();
  // Retrieved the scaled values from the compass (scaled to the configured scale).
  MagnetometerScaled scaled = compass.ReadScaledAxis();

  // Values are accessed like so:
  int MilliGauss_OnThe_XAxis = scaled.XAxis;// (or YAxis, or ZAxis)

  // Calculate heading when the magnetometer is level, then correct for signs of
  axis. float heading = atan2(scaled.YAxis, scaled.XAxis);

  // Once you have your heading, you must then add your 'Declination Angle',
  which is the 'Error' of the magnetic field in your location. // Find yours here:
  http://www.magnetic-declination.com/
  // Mine is: 2;½? 37' W, which is 2.617 Degrees, or (which we need)
  0.0456752665 radians, I will use 0.0457
  // If you cannot find your Declination, comment out these two lines, your
  compass will be slightly off. float declinationAngle = 0.0005585;
  heading += declinationAngle;

  // Correct for when signs are reversed.
  if(heading < 0)
    heading += 2*PI;

  // Check for wrap due to addition of declination.

```

```

if(heading > 2*PI)
    heading -= 2*PI;

// Convert radians to degrees for readability.
float headingDegrees = heading * 180/M_PI;

return ((int)headingDegrees);

// Output the data via the serial port.
Output(raw, scaled, heading, headingDegrees);

// Normally we would delay the application by 66ms to allow the loop
// to run at 15Hz (default bandwidth for the HMC5883L).
// However since we have a long serial out (104ms at 9600) we will let
// it run at its natural speed.
// delay(66);m m
}

void calcDesiredTurn(void)
{
    // calculate where we need to turn to head to destination
    headingError = targetHeading - currentHeading;

    // adjust for compass wrap
    if (headingError < -180)
        headingError += 360;
    if (headingError > 180)
        headingError -= 360;

    // calculate which way to turn to intercept the targetHeading
    if (abs(headingError) <= HEADING_TOLERANCE){ // if within tolerance,
        don't turn
        turnDirection = straight;

```

```

    Serial.print(" Turning Straight "); }
else if (headingError < 0){
Serial.print(" Turning Left ");
turnDirection = left;} else if
(headingError > 0){ Serial.print("
Turning Right "); turnDirection =
right;}
else{
Serial.print(" Going Straight ");
turnDirection = straight;
}
} // calcDesiredTurn()

```

```

//////////////////////////////////////-----
//////////////////////////////////////

```

```

////////////////////////////////////// Check Sonar & Obstacle Aviodance
System//////////////////////////////////////

```

```

void checkSonar(void)
{

```

```

    delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms
should be the shortest delay between pings.

```

```

    Serial.print("Ping: ");
    Serial.print(sonar.ping_cm()); // Send ping, get distance in cm and print result (0
= outside set distance range)
    Serial.println("cm");
    sonarDistance = sonar.ping_cm();

}

```

```

void checkTurn (void)
{
    if (turnDirection == right)
    {

```

```

{
    if (SR >= SSD)
        exit;}
    else{
        motork.run(left);}
}
if (turnDirection == left)
{
    if (SL >= SSD)
        { exit;}
    else
        {motork.run(right);}
}
}

```

```

void moveAndAvoid(void)

```

```

{
    if (sonarDistance >= SAFE_DISTANCE) // no close objects in front of car
    {
        if (turnDirection == straight)
            {checkStraight ();
            speed = FAST_SPEED;
            } else
            speed = TURN_SPEED;
        motor.setSpeed(speed);
        motor.run(FORWARD);
        checkTurn ();
        motork.run(turnDirection);
        Serial.print("\nMoving Forward");
        return;
    }
}

```

```

if (sonarDistance > TURN_DISTANCE && sonarDistance < SAFE_DISTANCE)
// not yet time to turn, but slow down

```

```

{
  if (turnDirection == straight)
    {
      checkStraight ();
      speed = NORMAL_SPEED;
    }
    else
    {
      checkTurn ();
      speed = TURN_SPEED;
      motork.run(turnDirection); // already turning to navigate
    }
  motor.setSpeed(speed);
  motor.run(FORWARD);
  Serial.print("\nSlowing Down & Moving Forward");
  return;
}

if (sonarDistance < TURN_DISTANCE && sonarDistance >
STOP_DISTANCE) // getting close, time to turn to avoid object
{
  speed = SLOW_SPEED;
  motor.setSpeed(speed); // slow down
  motor.run(FORWARD);
  switch (turnDirection)
  {
    case straight: // going straight currently, so start new turn
      {
        checkStraight ();
        if (headingError <= 0)
          turnDirection = left;
        else
          turnDirection = right;
        motork.run(turnDirection); // turn in the new direction
        break;
      }
    case left: // if already turning left, try right
      {
        checkTurn ();

```



```

        delay(100);
    } // while (sonarDistance < TURN_DISTANCE)
    motor.run(RELEASE);

    // stop backing up
    Serial.print("\nBreaking & Moving Backward");
return;
    } // end of IF TOO CLOSE

} // moveAndAvoid()

void checkStraight (void)
{
    if(turnDirection == straight && SR >= SSD && SL >= SSD)
        {exit;}
    else if(turnDirection == straight && SR >= SSD && SL < SSD)
        {speed = TURN_SPEED; motor.run(right);} else
    if(turnDirection == straight && SR < SSD && SL >= SSD)
        { speed = TURN_SPEED;
        motor.run(left);} else if(turnDirection == straight && SR <
SSD && SL < SSD)
        {
            motor.run(RELEASE); // stop
            motork.run(RELEASE); // straighten up turnDirection =
straight; motor.setSpeed(NORMAL_SPEED); // go back at
higher speet
            if (BckDist >= SAFE_DISTANCE)
                {
                    motor.run(BACKWARD);
                }
        }
    else
        {
            motor.run(RELEASE);
        }
}

```

```
}
```

```
// end of program routine, loops forever void
```

```
loopForever(void)
```

```
{ while
```

```
(1)
```

```
;
```

```
}
```

```
////////////////////////////////////-----End Of Code -----////////////////////////////////////
```