Android Mobile Operated Car Control

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Information Technology

under the Supervision of

Mr. Punit Gupta

By

Abhishek Mankotia (Roll no. - 111416)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "Android Mobile Operated Car Control" , submitted by Abhishek Mankotia in partial fulfilment for the award of degree of Bachelor of Technology in Information Technology to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Mr. Punit Gupta

Assistant Professor

Acknowledgement

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of this B.Tech. project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

I express my warm thanks to Mr. Punit Gupta for their support and guidance.

Also, I'd like to thank all the people who provided me with the facilities being required and conductive conditions for this project.

Thank you.

Date: **Abhishek Mankotia**

111416

Table of Content

List of Figures

Abstract

I think one of the major achievements achieved by computers besides doing complex calculations and its contribution to scientific research is in the field of making everyday tasks simpler. I reckon, almost every person who owns/drives a car has always thought of one thing every once in a while, their vehicle's fuel economy. Well, I thought why not give a solution to their problem and have some kind of a system to help them keep track of it. Besides that, why not provide them with some kind of a car control in their hands.

Talking of the problem, I had to make some kind of a system to control/keep track of various features of a car, beginning with fuel economy.

The solution I came up with was an embedded system interfaced with an Android mobile application. Initially, we planned on implementing only a fuel economy calculator but later, we decided on adding some other features such as a gauge to show the engine's temperature and a light control system. The embedded system consists of a microcontroller board – Arduino Uno, which is based on the ATmega328(datasheet), a few sensors – primarily a sensor to measure analogue signals and to measure the fuel level.

These sensors are going to send information to the Arduino board which is interfaced with the Android phone with the help of a Bluetooth module – HC 05.

On receiving the distance and fuel signals, the board will send them to the android phone, which on receiving the signals will calculate the fuel economy and show it

on the screen. The user will also be able to keep a log of the various fuel economies calculated in the past. I further plan on implementing some kind of demographics on screen with which the user will be able to keep track of his fuel economy even better.

This project spans over various disciplines as I have understood the working of traditional odometer as well as digital odometers and how the distance is calculated whist a vehicle's wheel is rotating. I've made use of the available equipment and implemented the same.

I can expand the scope of the project and utilize similar approach to implement other functionalities in the car, as previously stated. I feel this project has really helped me in achieving a keen sense of observation and application. Also, it has helped me realize the importance of using engineering principles to solve everyday problems.

CHAPTER 1

An Introduction to the Problem

Ever since the rise of fuel prices in recent times, fuel economy has become a priority of almost every other car commuter. Whilst there have been a lot of sanctions imposed on inefficient vehicles by major environmental protection agencies, one major issue that still remains is how to optimize your cars fuel economy and effectively calculate it.

A large number of vehicles are devoid of a system that keeps track of a car's fuel economy and shows it in real time, making the user aware of how much fuel he's expending, in turn, how much the user is spending on his car every day.

Most of us do a simple calculation each time our fuel tank hits empty: number of miles on our odometer divided by the liters of fuel consumed. In theory, this provides an accurate assessment of how many kilometers we can drive per litre. In reality, the number we come up with is a mere estimate that fluctuates by the tank, and even by the day.

This is perfectly fine for coming up with a rough number, but things start to get a bit confusing when you start trying to improve your car's fuel econmy. How do you know if the techniques or gadgets you're using are working if a rough estimate is all you can get? If your car's mileage varies by about 10 percent based on driving conditions, how do you know if the improvement you see after, say, adding acetone to your gas tank is a result of the additive or of the weather?

There are a large number of factors that affect your car's mileage. For instance, snowy weather requires headlights during the day, wipers and defrosting your front and rear windshields. All this activity uses fuel. On the other hand, warm weather might mean activating the air conditioning, which also lowers your fuel economy. And then there's the fact that fuel takes up less volume in the cold and expands in the heat, so the amount of gas you can fit in your tank isn't even constant.

Weather, hills, road conditions and frequency of stoplights all affect how many kilometers you get per litre at any given time, so manual calculation after you've gone through an entire tank doesn't do much for determining an increase or decrease in MPG

In this project I endeavor to provide a solution to the eminent problem of keeping track of your car's fuel economy. Before analyzing the work done during the course of this project, I'd like to highlight the existing technology present in this field.

CHAPTER 2

Existing Technology

Here, we look at some of the existing pioneers in the field who have developed systems which calculate a vehicle's fuel economy in real time and let users utilize their fuel more efficiently.

Both the systems mainly revolve around taking the signals from the fuel tank and another sensor present on the axle of the wheels. They come with additional benefits which are further discussed:

1. ScanGaugeE

ScanGaugeE provides you with real-time information about your vehicle's fuel economy through an intuitive graphic display. According to a report issued by the US government, adjusting your driving habits can increase fuel economy by up to

33%. Use the instant feedback provided by ScanGaugeE™ to adjust your driving style and improve your fuel economy.

It helps you keep track of your fuel costs and fuel used in real-time. View information such as miles-per-gallon, gallons-per-hour and Trip Fuel Used. Track actual fuel costs with Trip-Fuel-Costs and Today's-Fuel-Costs digital gauges.

Track your vehicle's CO2 output in real-time. View information such as Current-CO2 and Trip-CO2. You can also track the days Total CO2 and Total CO2 for the tank within the built-in Trip Computers.

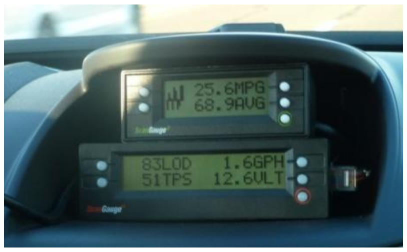


Figure 1 ScanGaugeE

ScanGaugeE installs in just minutes **without tools** and does not require batteries or an external power source. All data and power are derived from the single OBDII connection which is present on all 1996 and newer cars and light trucks.

The detachable six foot cord also allows ScanGaugeE to be mounted just about anywhere on the dash or console while staying connected to your vehicle.

Pros and Cons

The apparatus is kind of cumbersome to install. It comes with connector cables which can be kind of clumsy to install in your car. Also, the entire installation is kind of large so one needs to make sure they have the kind of space needed for such a system present in their vehicle.

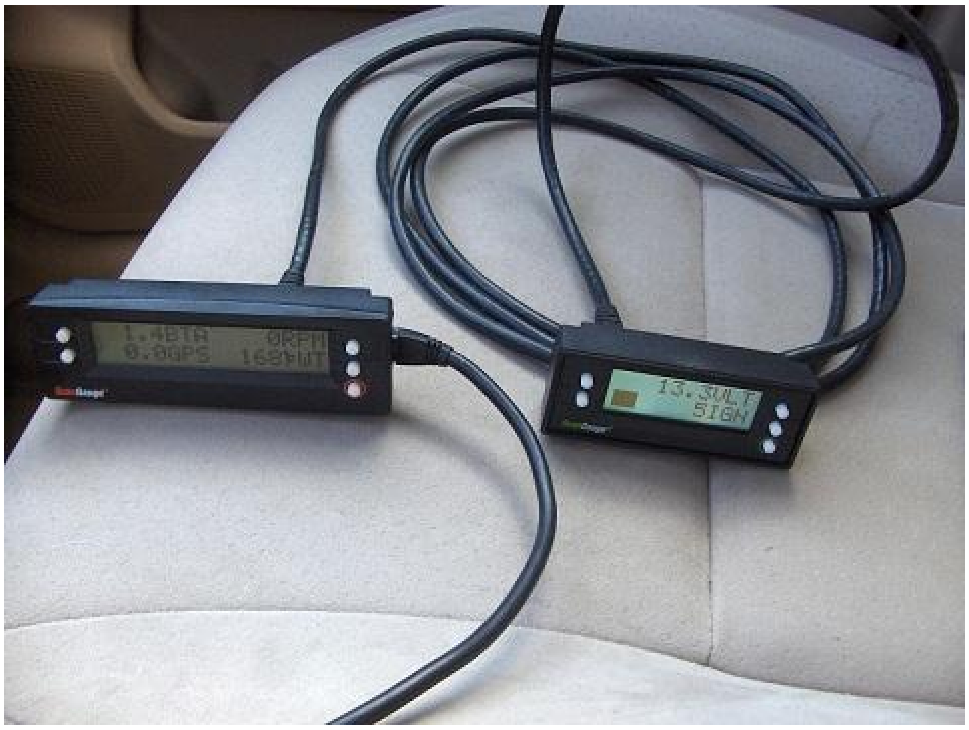


Figure 2 The entire ScanGaugeE appratus

Talking of advantages, the system has added a bar graph to the left of the display. The graph always tracks fuel economy over time, therefore, giving the user a real time idea of the fuel economy expended by the user. Also, the scale of the graph, or how it displays information is configurable. This is a pretty nice feature, in my opinion.

You can also set the graph's zero point to a set fuel economy. They call this "GOAL", and you can manually set whatever fuel economy you want as your zero point. With GOAL you can see if you are hitting the fuel economy you want or not.

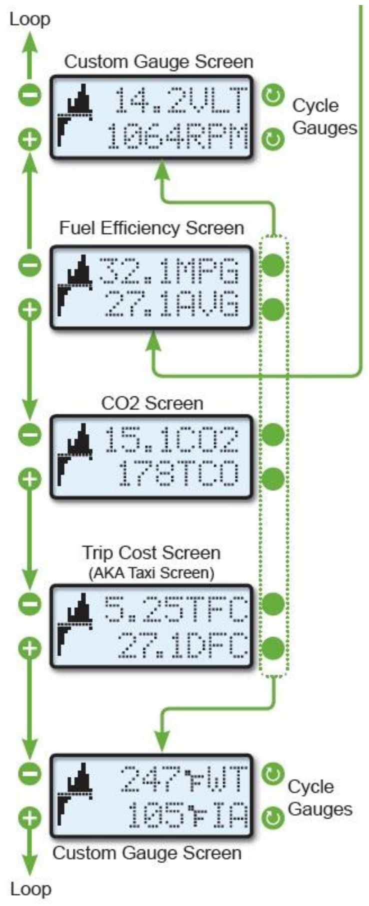


Figure 3 ScanGauge display

Also, they have added gauge sets. You can scroll through the gauge sets with the left buttons. There are three default (not customizable) gauge sets, and two customizable sets. The default gauge sets show instant and trip fuel economy, the next shows trip CO2 emissions and today's CO2, and the third shows trip fuel cost and today's fuel cost.

This implies that the system also helps environmentally conscious individuals.

Another downer is the cost, at $96 this apparatus fails to attract a lot of potential buyers who'd rather stick to their copies and pens than spend that kind of money on a fancy gizmo.

2. Fuel flow meters DFM

Inline fuel meter DFM can be applied for fuel accounting both autonomously and as a part of vehicle tracking and fuel monitoring system.
Fuel gauge DFM allows to solve the following tasks:

- fuel consumption control;
- fuel consumption rationing;
- fuel theft detecting and preventing;
- fuel comsumption optimization and real-time monitoring;
- engine fuel consumption testing.

Fuel meter DFM enables to receive objective information about **actual fuel consumption** and **vehicle working time**. It also permits to reduce fuel and repairing costs. It is possible to develop fuel consumption rates for selected routes and and technological operations.

The economic effect of using the fuel accounting devices is different at various companies, usually about 10 to 40%, depending on the baseline situation and management persistence.

Fuel gauge DFM has three-dimension ring type measuring chamber. DFM generates an impulse, when the volume of fuel (which is equal to volume of the measuring chamber) passes through it.
For detailed operating principle for every DFM model please visit product pages.

Figure 4 DFM Fuel Flow Meter

CHAPTER 3

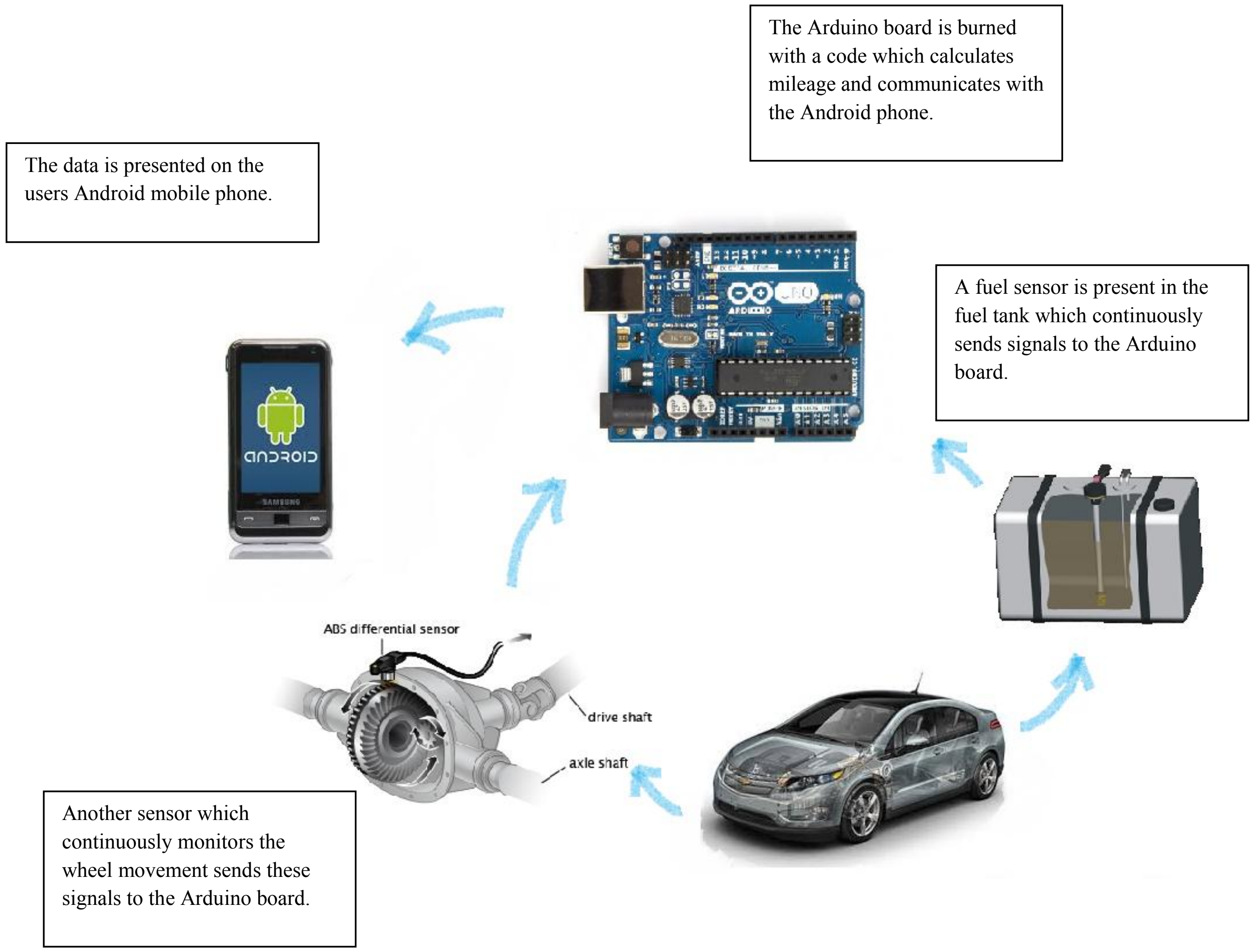
Proposed Solution

The solution I came up with along with my project supervisor revolves around exploiting the various functionalities offered by two upcoming technologies – Arduino Uno and Android. We'll talk of both these technologies later in this document. For now, let us think of the board as means of connecting readily available sensors and exploiting them through our handheld devices.

We shall be using a fuel sensor and a potentiometer to emulate the working of an actual car. We shall than proceed to make the user interact with the hardware using his mobile phone, where, not only can he view the distance traversed and fuel consumed in real time but also keep track of his past fuel economies which can help him in bettering his fuel consumption decisions.

Infrastructure diagrams in the following pages will assist in better understanding of the project.

1. Infrastructure Diagram



The Arduino board is burned with a code which calculates mileage and communicates with the Android phone.

The data is presented on the users Android mobile phone.

A fuel sensor is present in the fuel tank which continuously sends signals to the Arduino board.

Another sensor which continuously monitors the wheel movement sends these signals to the Arduino board.

Figure 5 Infrastructure diagram - 1

Now the question arises: how will we demonstrate the feasibility of this project in the confines of a computer lab or a small room? Don't worry we shall emulate the project using a few sensors compatible with Arduino Uno. The details of these sensors shall be discussed later.

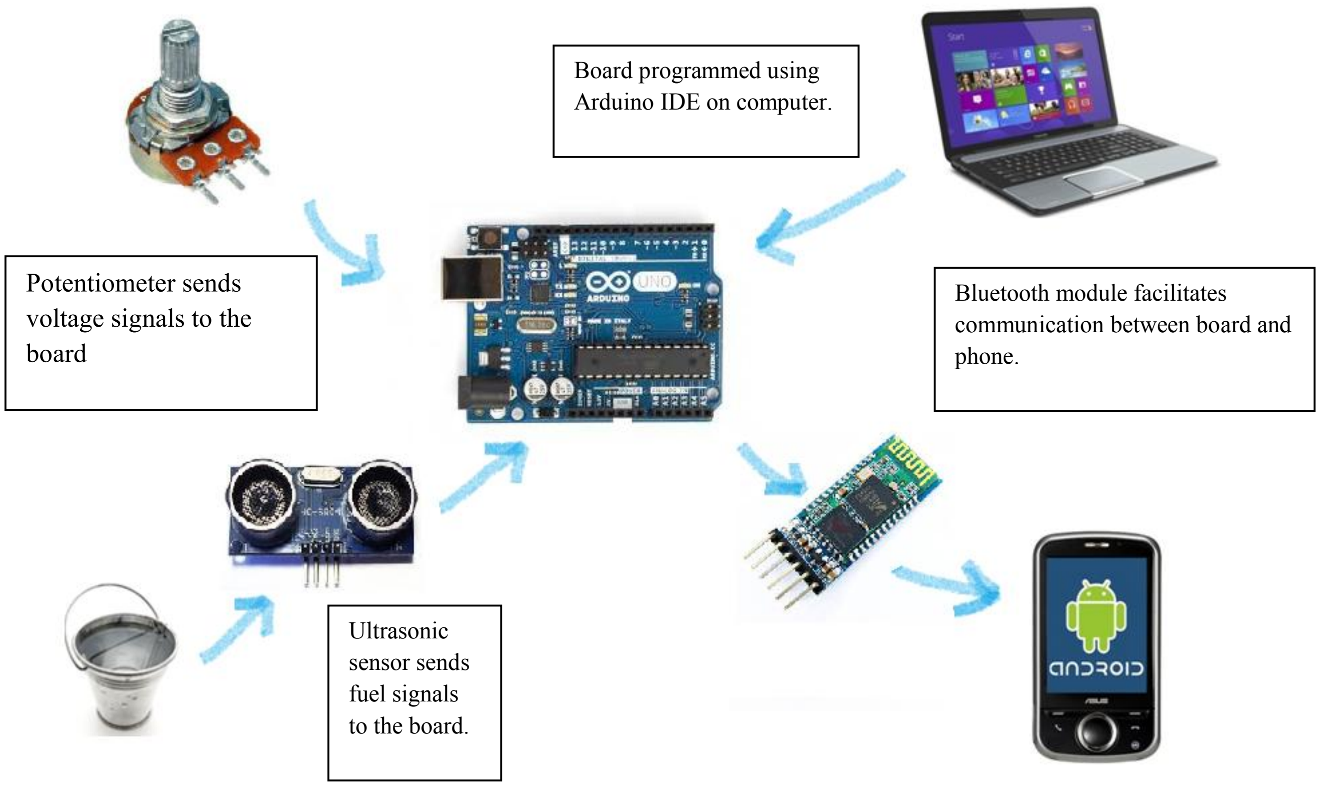

Board programmed using Arduino IDE on computer.
Potentiometer sends voltage signals to the board
Bluetooth module facilitates communication between board and phone.
Ultrasonic sensor sends fuel signals to the board.

Figure 6 Infrastructure diagram - 2

CHAPTER 4

An Introduction to Hardware

Now, we have a brief idea of how the solution will work, we shall progress to understand the various hardware used in the making of this project. Starting off with the development board:

1. Arduino Uno

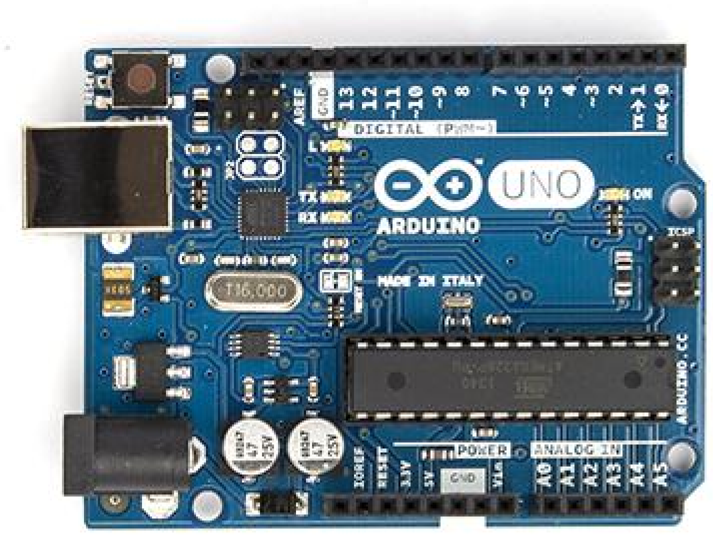


Figure 7 Arduino Uno

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

It can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

2. Important Features of Arduino Uno

There are many other microcontrollers and icrocontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino
- Open source and extensible software- The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand

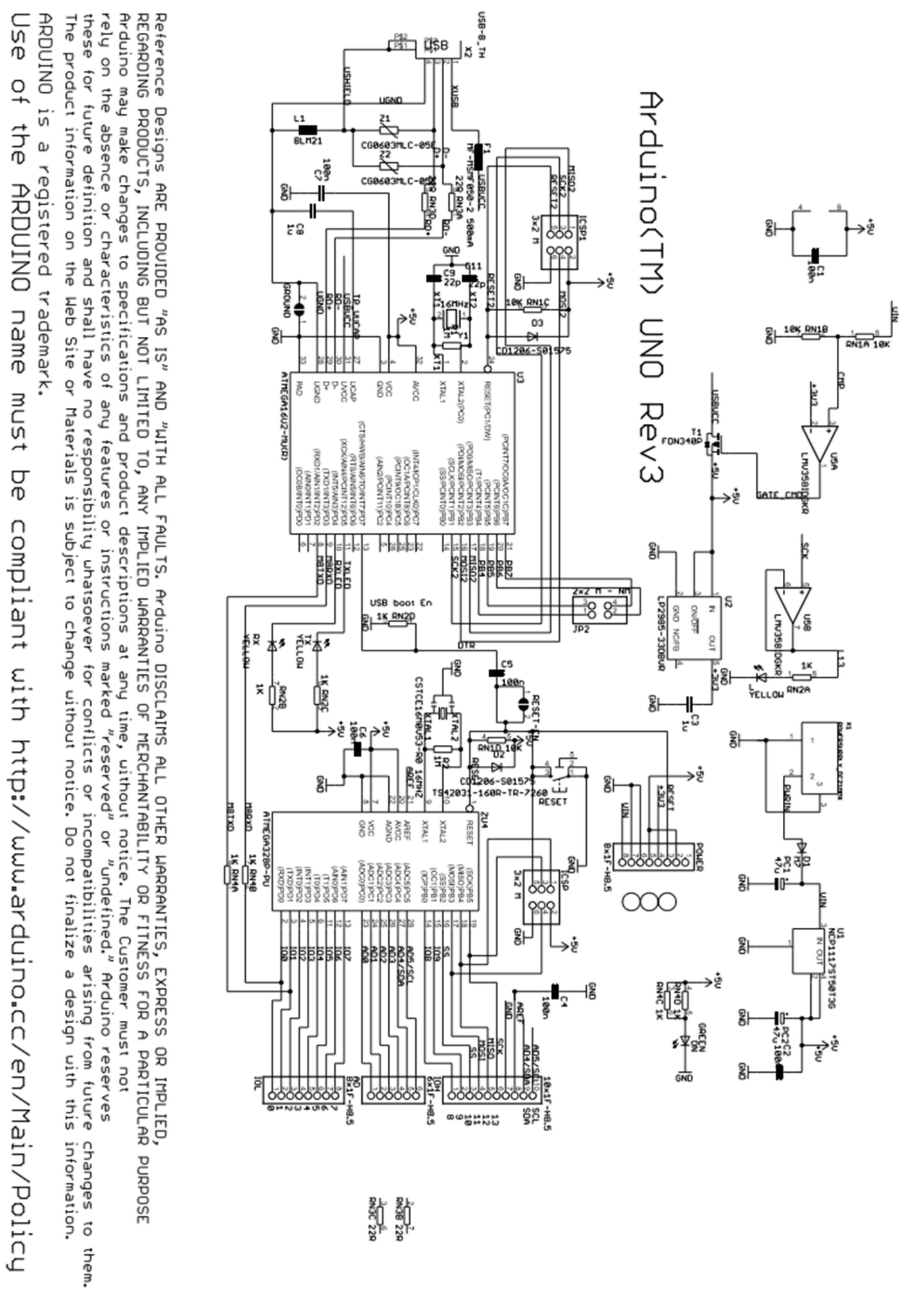
the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

- Open source and extensible hardware - The Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

3. Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

4. Datasheet



Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.
ARDUINO is a registered trademark.
Use of the ARDUINO name must be compliant with http://www.arduino.cc/en/Main/Policy

Figure 8 Datasheet

5. Pinout Diagram

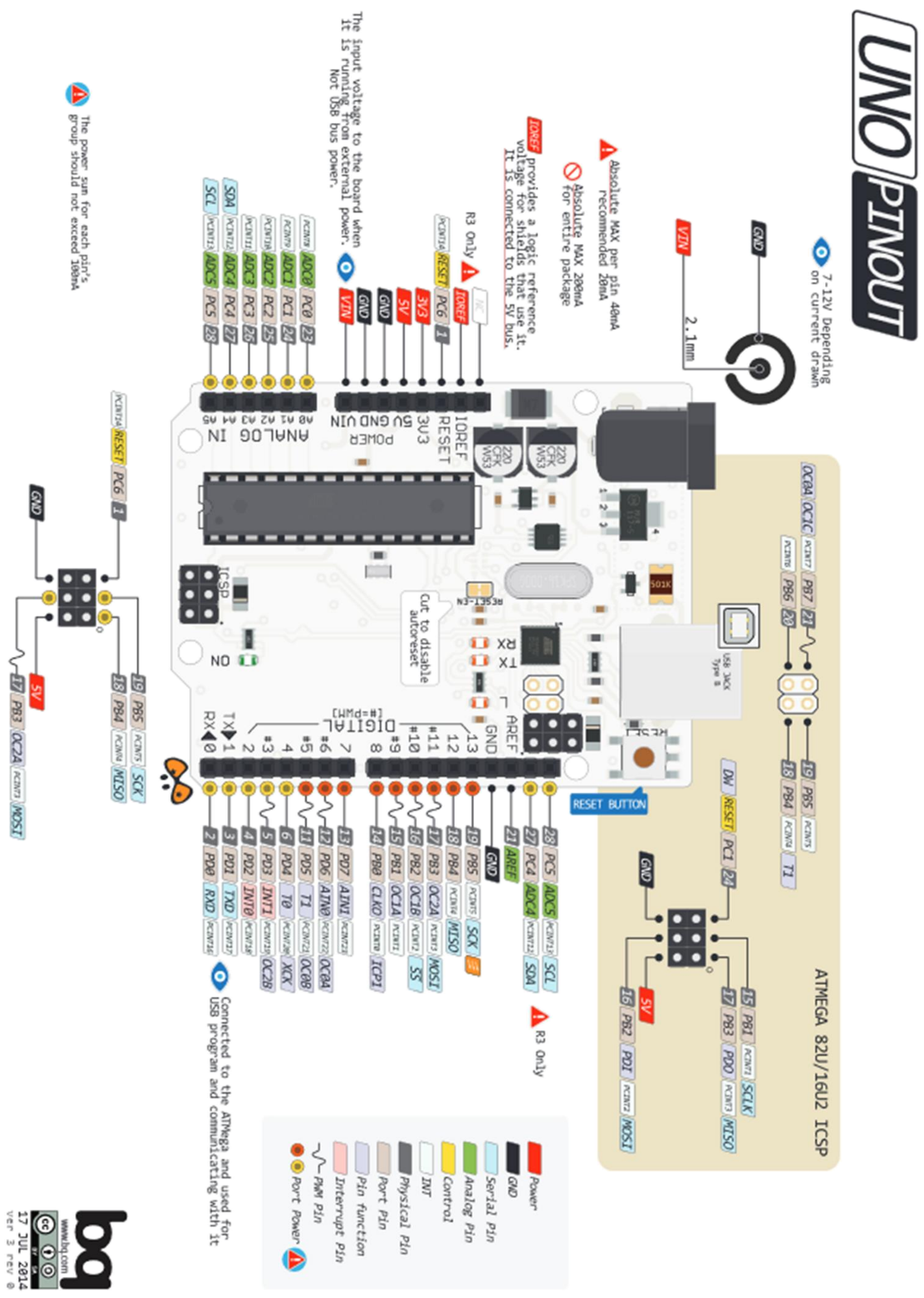

UNO PINOUT
7-12V Depending on current drawn
GND
VIN
2.1mm
Absolute MAX per pin 40mA recommended 20mA
Absolute MAX 200mA for entire package
IOREF provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
R3 Only
The input voltage to the board when it is running from external power. Not USB bus power.
The power sum for each pin's group should not exceed 100mA
ANALOG IN
POWER
DIGITAL (PWM=~)
ICSP
RESET BUTTON
Cut to disable autoreset
ATMEGA 82U/16U2 ICSP
Connected to the ATMega and used for USB program and communicating with it
Power
GND
Serial Pin
Analog Pin
Control
INT
Physical Pin
Port Pin
Pin function
Interrupt Pin
PWM Pin
Port Power
www.bq.com
17 JUL 2014
ver 3 rev 0

Figure 9 Pin Out diagram

6. HC-05 module

HC-05 embedded Bluetooth serial communication module (can be short for module) has two work modes: order-response work mode and automatic connection work mode. And there are three work roles (Master, Slave and Loopback) at the automatic connection work mode. When the module is
at the automatic connection work mode, it will follow the default way set lastly to transmit the data automatically. When the module is at the order-response work mode, user can send the AT command to the module to set the control parameters and sent control order. The work mode of module can be switched by controlling the module PIN (PIO11) input level.

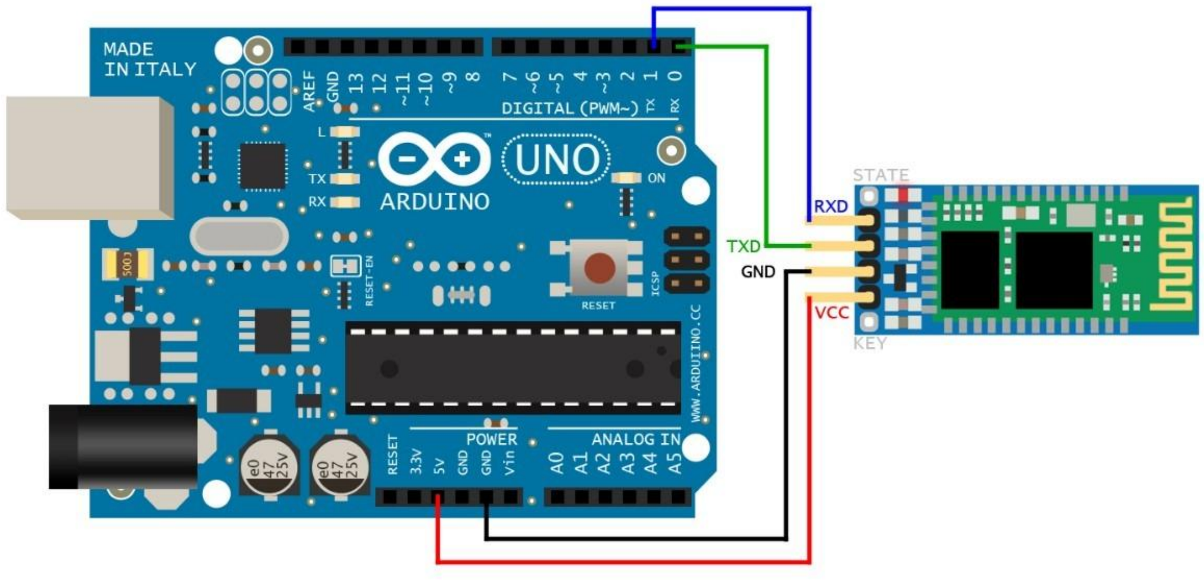


Figure 10 Connections from Bluetooth module HC-05 to Arduino Uno

7. HC-SR04 Ultrasonic Sensor

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

Features

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

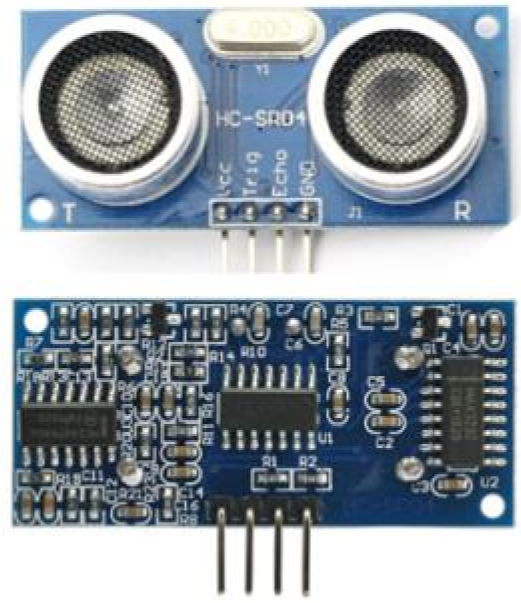


Figure 11 HC-SR04 Ultrasonic Sensor

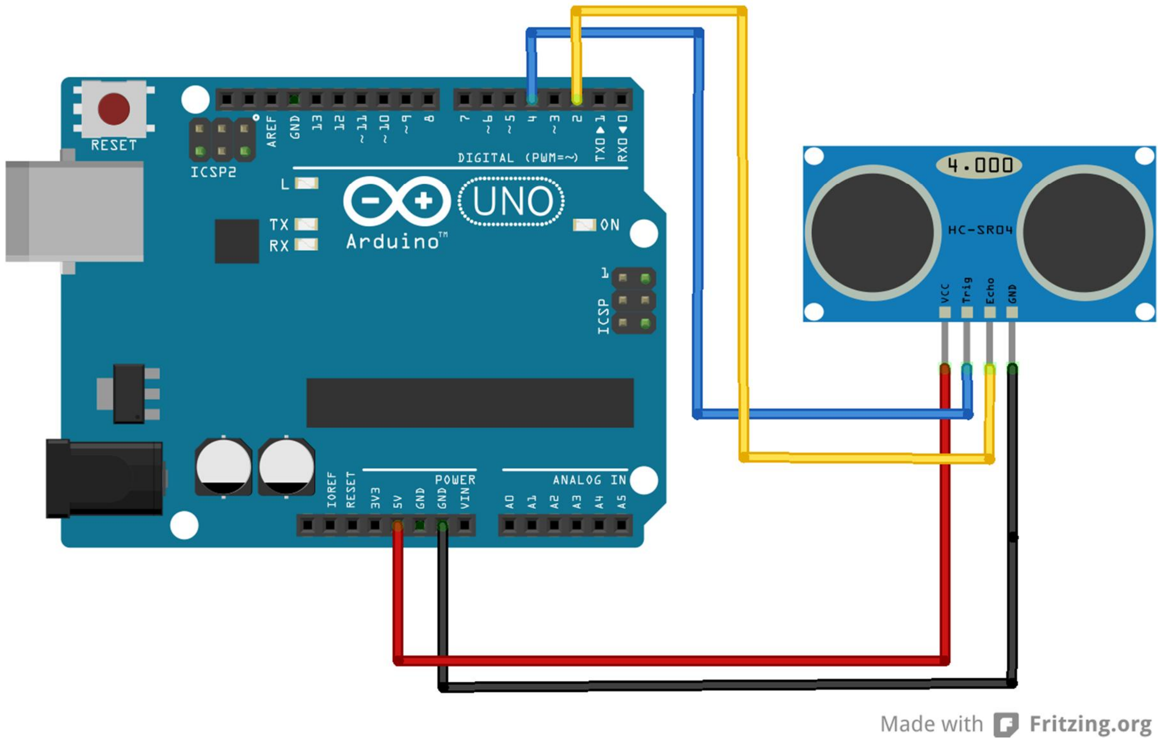


Figure 12 Connections from HC-SR04 Ultrasonic Sensor to Arduino Uno

8. Liquid Crystal Display

The LiquidCrystal library allows you to control LCD displays. There are many of them out there, and you can usually tell them by the 16-pin interface.

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

A Read/Write (R/W) pin that selects reading mode or writing mode

An Enable pin that enables writing to the registers

8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

There's also a display constrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.

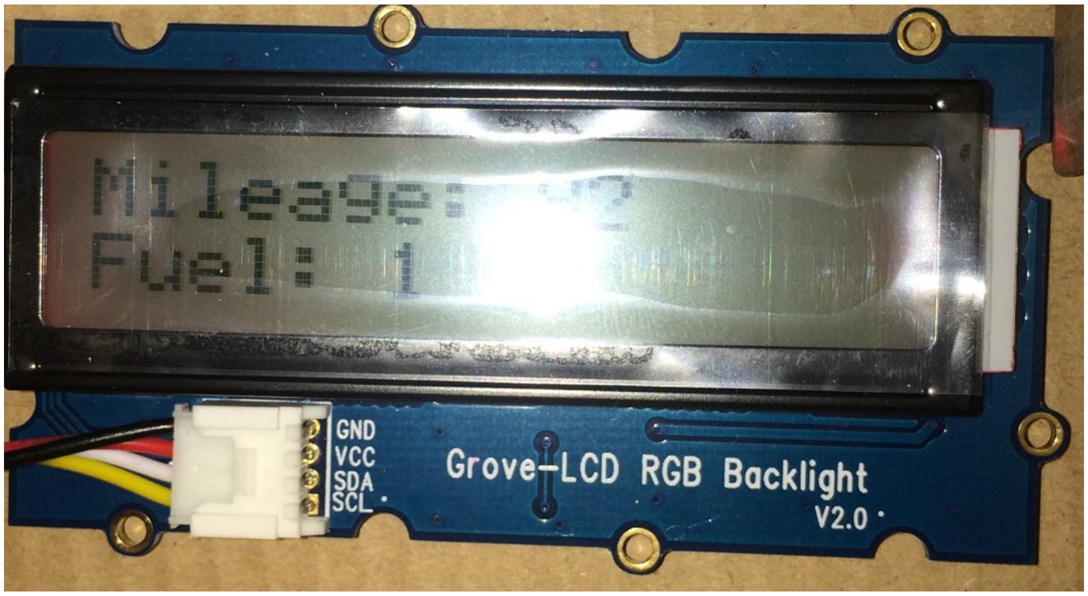


Figure 13 LCD interfaced with Arduino

CHAPTER - 5

An Introduction to IDEs and SDKs Used

We now have a brief idea of all the kinds of hardware used in the making of this project. In order to harness the potential of the various sensors, we now need to embed an ingenious code which will help us to achieve the desired motive.

Arduino Uno can be embedded with a code, written and compiled in a light IDE known as Arduino IDE. The version used in the making of this project is 1.0.6.

Also, talking of interaction with the mobile phone; an application has been developed in Android. The choice of platform relies on a number of benefits provided by this particular environment, the one that tops the list being the ability to freely upload your own application on your phone bypassing numerous formalities on other available platforms such as iOS and Windows.

1. Arduino 1.0.6

The Arduino integrated development environment (IDE) is a cross-platform application written in Java, and derives from the IDE for the Processing programming language and the Wiring projects. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. A program or code written for Arduino is called a *sketch*.

Arduino programs are written in C or C++. The Arduino IDE comes with a software library called "Wiring" from the original Wiring project, which makes many common input/output operations much easier. Users only need define two functions to make a runnable cyclic executive program:

- setup(): a function run once at the start of a program that can initialize settings
- loop(): a function called repeatedly until the board powers off

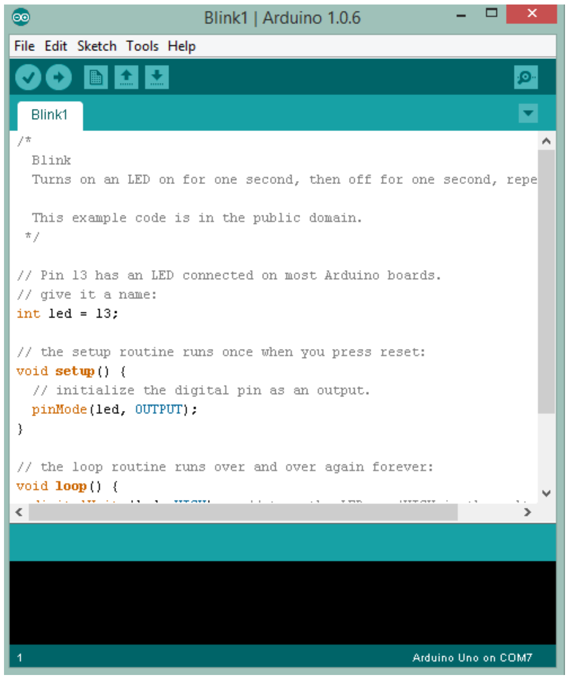
A typical first program for a microcontroller simply blinks an LED on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13

void setup () {
  pinMode (LED_PIN, OUTPUT); // Enable pin 13 for digital output
}

void loop () {
  digitalWrite (LED_PIN, HIGH); // Turn on the LED
  delay (1000); // Wait one second (1000 milliseconds)
  digitalWrite (LED_PIN, LOW); // Turn off the LED
  delay (1000); // Wait one second
}
```

Screen grab of the IDE is as follows:



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
```

Figure 14 A Screen grab of the Arduino IDE

1.1. Structure of Arduino Code

Sketch

A *sketch* is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board.

Comments

The first few lines of the Blink sketch are a *comment*:

/ Hello */*

Everything between the /* and */ is ignored by the Arduino when it runs the sketch (the * at the start of each line is only there to make the comment look pretty, and isn't required). It's there for people reading the code: to explain what the program does, how it works, or why it's written the way it is. It's a good practice to comment your sketches, and to keep the comments up-to-date when you modify the code. This helps other people to learn from or modify your code.

There's another style for short, single-line comments. These start with // and continue to the end of the line. For example, in the line:

```
int ledPin = 13;            // LED connected to digital pin 13
```

the message "LED connected to digital pin 13" is a comment.

Variables

A *variable* is a place for storing a piece of data. It has a name, a type, and a value. For example, the line from the Blink sketch above declares a variable with the name ledPin, the type int, and an initial value of 13. It's being used to indicate which Arduino pin the LED is connected to. Every time the name ledPin appears in the code, its value will be retrieved. In this case, the person writing the program could have chosen not to bother creating the ledPin variable and instead have simply written 13 everywhere they needed to specify a pin number. The advantage of using a variable is that it's easier to move the LED to a different pin: you only need to edit the one line that assigns the initial value to the variable.

Functions

A *function* (otherwise known as a *procedure* or *sub-routine*) is a named piece of code that can be used from elsewhere in a sketch. For example, here's the definition of the setup() function from the Blink example:

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

The first line provides information about the function, like its name, "setup". The text before and after the name specify its return type and parameters: these will be explained later. The code between the { and } is called the *body* of the function: what the function does.

You can *call* a function that's already been defined (either in your sketch or as part of the Arduino language). For example, the line pinMode(ledPin, OUTPUT); calls the pinMode() function, passing it two *parameters*: ledPin and OUTPUT. These parameters are used by the pinMode() function to decide which pin and mode to set.

pinMode(), digitalWrite(), and delay()

The **pinMode()** function configures a pin as either an input or an output. To use it, you pass it the number of the pin to configure and the constant INPUT or OUTPUT.

When configured as an input, a pin can detect the state of a sensor like a pushbutton. As an output, it can drive an actuator like an LED.

The **digitalWrite()** functions outputs a value on a pin. For example, the line:

```
digitalWrite(ledPin, HIGH);
```

set the ledPin (pin 13) to HIGH, or 5 volts. Writing a LOW to pin connects it to ground, or 0 volts.

The **delay()** causes the Arduino to wait for the specified number of milliseconds before continuing on to the next line. There are 1000 milliseconds in a second, so the line:

```
delay(1000);
```

creates a delay of one second.

setup() and loop()

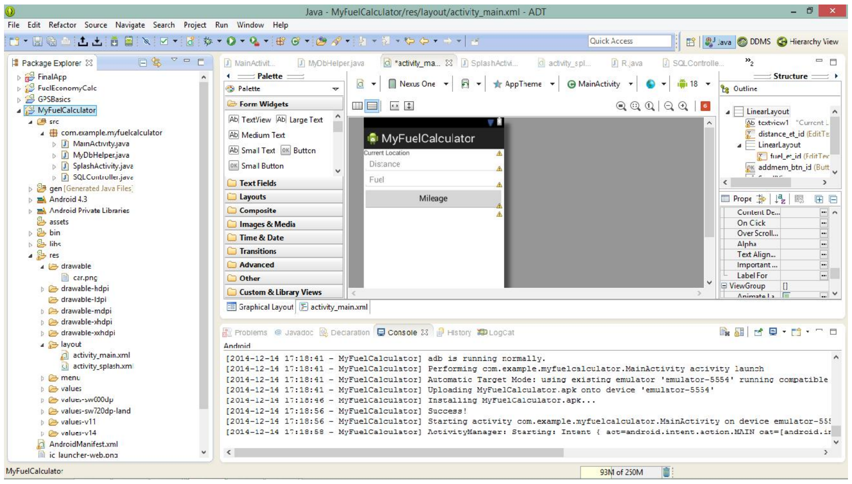
There are two special functions that are a part of every Arduino sketch: setup() and loop(). The setup() is called once, when the sketch starts. It's a good place to do setup tasks like setting pin modes or initializing libraries. The loop()function is called over and over and is heart of most sketches. You need to include both functions in your sketch, even if you don't need them for anything.

2. Android Development Tools

The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, and Windows XP or later. For the moment one can also develop Android software on Android itself by using the AIDE - Android IDE - Java, C++ app and the Java editor app. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) Plugin, though IntelliJ IDEA IDE (all editions) fully supports Android development out of the box, and NetBeans IDE also supports Android development via a plugin. Additionally, developers may use any text editor to edit Java and XML files, then use command line tools (Java Development Kit and Apache Ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).

Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Android applications are packaged in .apk format and stored under `/data/app` folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex file (compiled byte code files called Dalvik executables), resource files, etc.



```
[2014-12-14 17:18:41 - MyFuelCalculator] adb is running normally.
[2014-12-14 17:18:41 - MyFuelCalculator] Performing com.example.myfuelcalculator.MainActivity activity launch
[2014-12-14 17:18:41 - MyFuelCalculator] Automatic Target Mode: using existing emulator 'emulator-5554' running compatible
[2014-12-14 17:18:41 - MyFuelCalculator] Uploading MyFuelCalculator.apk onto device 'emulator-5554'
[2014-12-14 17:18:46 - MyFuelCalculator] Installing MyFuelCalculator.apk...
[2014-12-14 17:18:56 - MyFuelCalculator] Success!
[2014-12-14 17:18:56 - MyFuelCalculator] Starting activity com.example.myfuelcalculator.MainActivity on device emulator-555
[2014-12-14 17:18:58 - MyFuelCalculator] ActivityManager: Starting: Intent { act=android.intent.action.MAIN cat=[android.ir
```

Figure 15 Android Development Tools

CHAPTER – 6

Understanding Different Modules

(Exploring the Hardware)

1. Blink Example

We started off with implementing a simple program on the arduino board. To build the circuit, attach a 220-ohm resistor to pin 13. Then attach the long leg of an LED (the positive leg, called the anode) to the resistor. Attach the short leg (the negative leg, called the cathode) to ground.

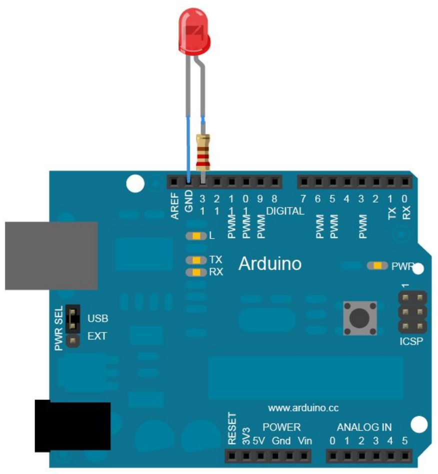


Figure 16 Connecting the LED and resistor to the Arduino board.

Code snippet:

```
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

2. Analog Read Voltage

Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 2 to the middle pin of the potentiometer.

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kilohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the **analog voltage** that you're reading as an input.

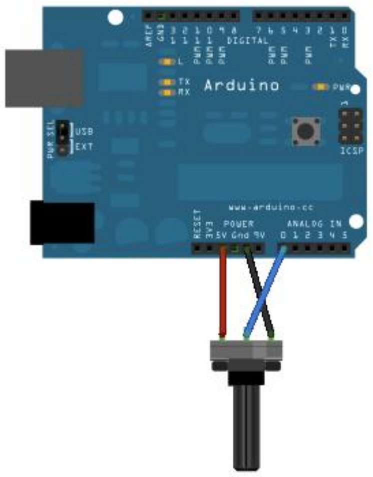


Figure 17 Connecting the potentiometer to the Arduino board

Code snippet:

```
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

3. Connecting Arduino Board with the HC – 05 Module

Firstly, we go to the bluetooth icon , right click and select Add a Device . On searching for new device , our bluetooth module will appear as HC-05 , and add it. The pairing code will be 1234. After make a pairing, we can now program the arduino and upload a sketch to send or receive data from Computer.

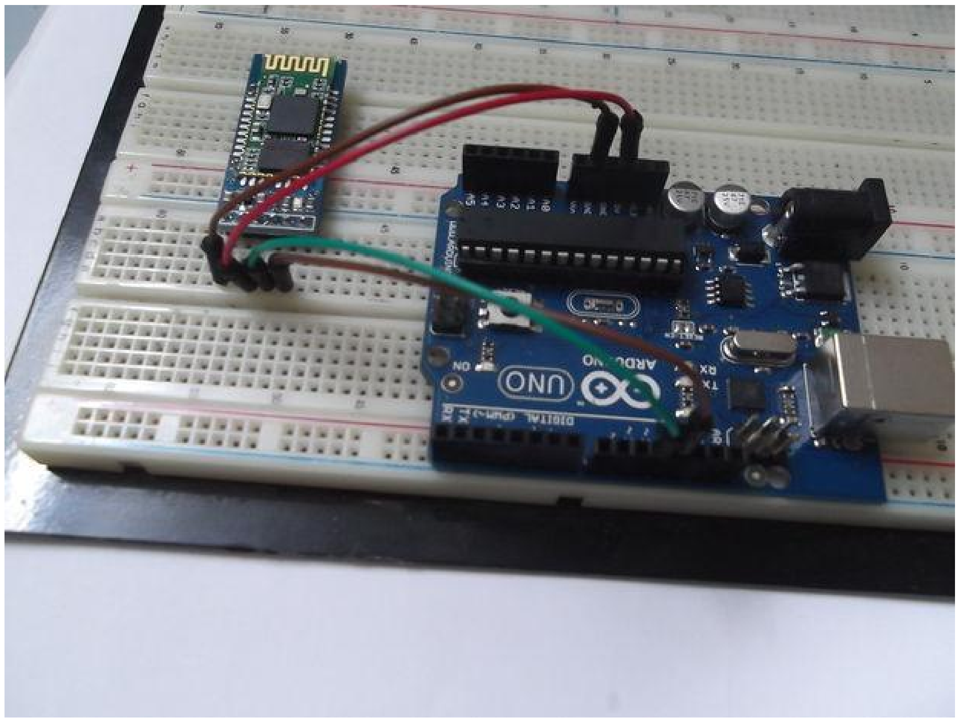


Figure 18 Connections on the Arduino Board

Code Snippet:

```
// see if there's incoming serial data:
if (Serial.available() > 0) {
// read the oldest byte in the serial buffer:
incomingByte = Serial.read();
// depending on the incoming byte do the needful action
if (incomingByte == '1') {
digitalWrite(11, HIGH);
//delay(500);
//Serial.println("ON");
state=1;
}
else if (incomingByte =='0') {
digitalWrite(11, LOW);
//delay(500);
//Serial.println("OFF");
state = 0;
}
```

4. Connecting Arduino Board with the HC – SR04 Module

This sensor is really easy to use and hence, popular among the Arduino Tinkerers. So I've decided to use this example using this sensor. In this project the ultrasonic sensor read and write the distance in the serial monitor. It's really simple.

Our goal is to help understand how this sensor works and then we can use this example in our major project

Note: There's an Arduino library called NewPing that comes really handy whilst using this sensor.

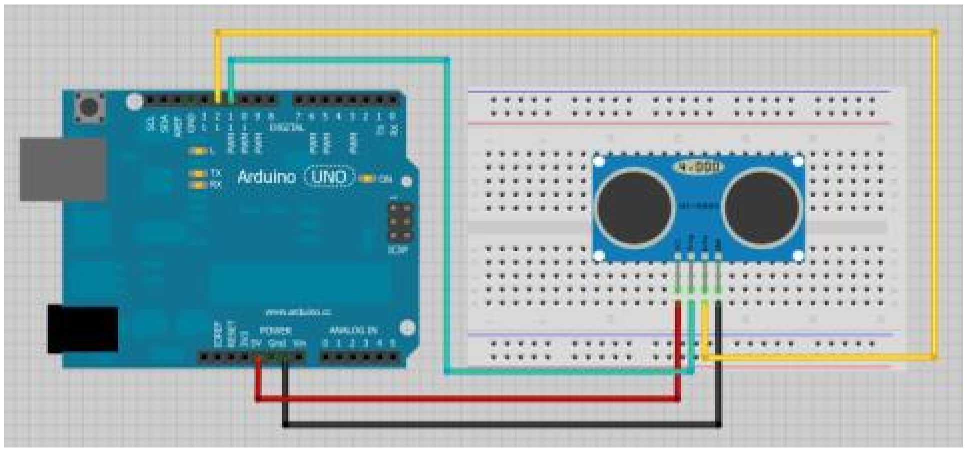


Figure 19 Connections on the Arduino Board

Code Snippet:

```
// Read the signal from the sensor: a HIGH pulse whose

// duration is the time (in microseconds) from the sending

// of the ping to the reception of its echo off of an object.

  pinMode(echoPin, INPUT);

  duration = pulseIn(echoPin, HIGH);

  // convert the time into a distance

  cm = (duration/2) / 29.1;

  inches = (duration/2) / 74;

  Serial.print(inches);

  Serial.print("in, ");

  Serial.print(cm);
```

5. Connecting Arduino Board with A LCD

Luckily, I was provided with a Grove sensor kit which had a backlit LCD display with a predefined library at disposal, the connections are pretty simple: GND and VCC fit in where they belong, SDA and SCL fit in on the Arduino Board.

We use various functions contained in the predefined library: Grove_LCD_RGB_Backlight-master.

Hardware Required

- Arduino Board
- LCD Screen
- hook-up wire

Circuit Diagram

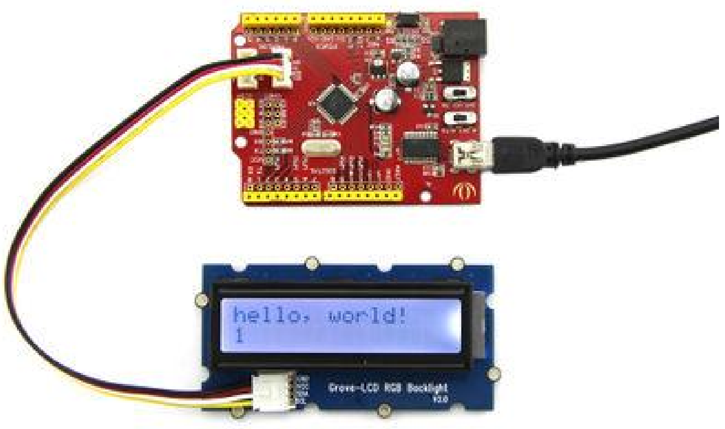


Figure 20 Connections from LCD to Arduino board

Code Snippet:

```
void setup()
{
   lcd.begin(16, 2);
   lcd.setRGB(colorR, colorG, colorB);
   lcd.print("hello, world!");
   delay(1000);
}
void loop()
{
   lcd.setCursor(0, 1);
   lcd.print(millis()/1000);
   delay(100);
}
```

CHAPTER - 7

Work Done

1. Arduino Code for sending Distance and Fuel Signals via Bluetooth

In this module, I have designed a circuit which is similar to the one present in a car where distance and fuel signals are continuously fed to the digital odometer present on the dashboard. We shall be using a potentiometer to emulate the movement of wheels and an ultrasonic sensor, HC-SR04, interfaced with the Arduino board shall measure changes in the fuel level if any. Also, we shall be displaying the mileage and fuel on an LCD.

The values obtained from the different sensors are sent with the help of a Bluetooth module, namely HC-05. It is interfaced with the development board as shown in a schematic (figure 3.4).

1.1. Requirements

- Arduino Uno
- A potentiometer
- A backlit LCD display
- HC-SR04 Ultrasonic Sensor
- HC-05 Bluetooth Module
- A Breadboard
- Connecting Wires

1.2. Circuit Diagram

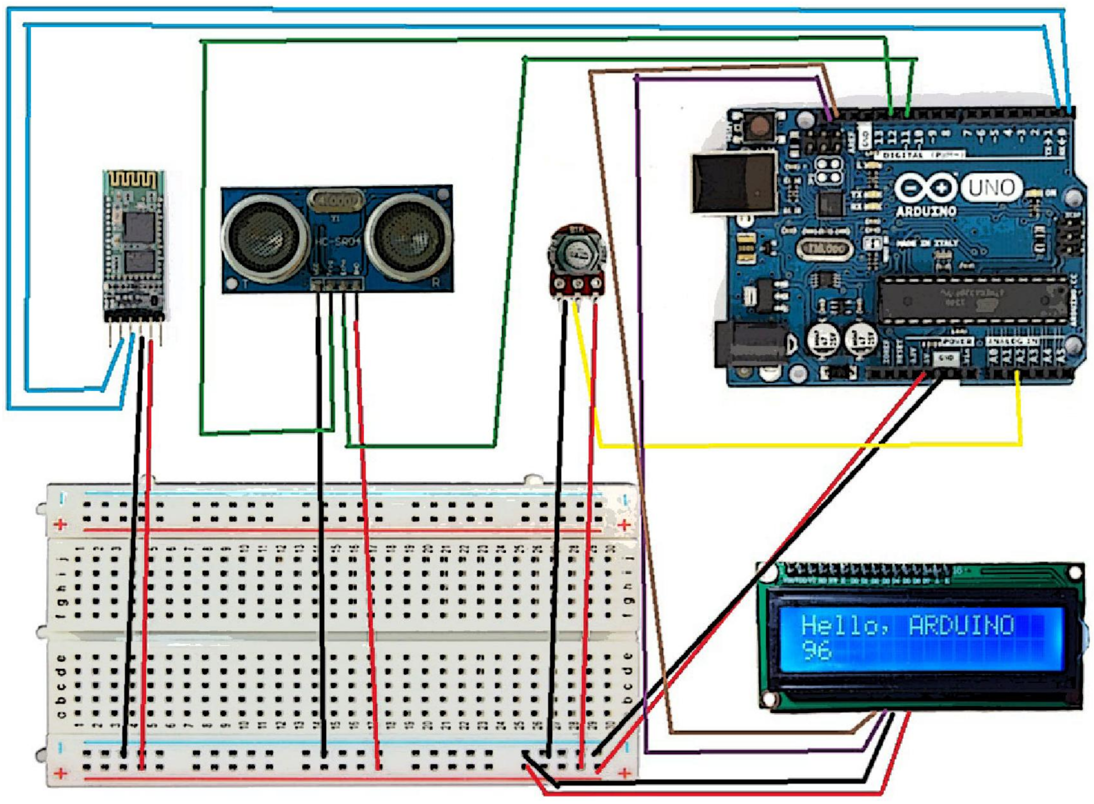


Figure 21 Circuit Diagram

1.3. Explanation of Code

#include <NewPing.h>

#include <Wire.h>

#include "rgb_lcd.h"

The **NewPing** library increases the performance of the ultrasonic sensor used to measure the fuel level in the project. The NewPing library totally fixes initial problems faced by the sensor, adds many new features, and breathes new life into these very affordable distance sensors. Some of the features are:

- Works with many different ultrasonic sensor models: SR04, SRF05 etc.
- Option to interface with all but the SRF06 sensor using only one Arduino pin.
- Doesn't lag for a full second if no ping echo is received like all other ultrasonic libraries.
- Ping sensors consistently and reliably at up to 30 times per second.
- Timer interrupt method for event-driven sketches.

The **Wire** library allows you to communicate with I2C / TWI devices. On the Arduino boards with the R3 layout (1.0 pinout), the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. The Arduino Due has two I2C / TWI interfaces SDA1 andSCL1 are near to the AREF pin and the additional one is on pins 20 and 21.

Here “rgb_lcd.h” file contains certain functions to utilize the LCD display has been provided by the sensor manufacturer.

#define TRIGGER_PIN 12 // Arduino pin tied to trigger pin on the ultrasonic sensor.

#define ECHO_PIN   11 // Arduino pin tied to echo pin on the ultrasonic sensor.

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

The above statements identify the various connection from the HC-SR04 Ultrasonic sensor.

The following variables decide the color of the backlit LCD:

const int colorR = 255;

const int colorG = 0;

const int colorB = 0;

Two arrarys are defined in order to store the obtained values from the sensors and the final values:

int sensorValue[4] = {0,0};

int voltageValue[4] = {0,0};

Now, the setup() function is called when a sketch starts. We use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

Code Snippet:

```
Serial.begin(9600); //change to 16, 2 for smaller 16x2 screens

  pinMode(led, OUTPUT);

  digitalWrite(led, LOW);

  counter=0;

  lcd.begin(16, 2);

  lcd.setRGB(colorR, colorG, colorB);

  lcd.print("ODOMETER");
```

Moving on to the loop function, it does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. We use it to actively control the Arduino board.

Code Snippet:

```
if (inbyte == '0')
  {
   //LED off
   digitalWrite(led, LOW);
  }
  if (inbyte == '1')
  {
   unsigned int uS = sonar.ping();
   do
    {
     digitalWrite(led, HIGH);
     val = analogRead(potPin);
     readSensor(uS);
     sendAndroidValues();
     delay(val);
     digitalWrite(led, LOW);
     delay(val);
    }while(inbyte == '1');
```

In this part of the code, we're sending the values from the sensor depending on the command communicated by the Android mobile application over Bluetooth.

If the value received is '1', the LED is turned ON and the distance and fuel signals are continuously sent to the application, if '0', the LED is turned OFF and the sensors come to a hault.

Also we use two functions, namely, readSensor() and sendAndroidValues() to read the value from the sensors and send these values over Bluetooth respectively. The pseudo codes are as follows:

sendAndroidValues()

```
Serial.print(sensorValue[0]);

Serial.print('+');

Serial.print('<');

Serial.print(sensorValue[1]);

Serial.print('>');

Serial.print('~');

Serial.println();
```

readSensor()

```
counter=counter+3.4;

sensorValue[0] = round(counter);

unsigned int uS4 = sonar.ping();

unsigned int uS2 = uS4 - uS0;

sensorValue[1] = (uS2/ US_ROUNDTRIP_CM);
```

On uploading the code on the development board, we initially observe the LCD turning on, displaying the text “ODOMETER”.

As soon as the user turns ON the engine using the mobile application, the LED turns on. The ultrasonic sensor continuously scans for any changes in the fuel level.

2. Android Application

After burning the code on the Arduino Uno, we move on to the next pedestal in communication between the user and hardware – The Android application.

The Android application I have developed connects your mobile phone to the embedded system and continuously scans for signals received. It then processes the received values of distance and fuel to calculate the mileage. All of the data is then stored in a table which the user can view for future reference. Also, I further plan on implementing some kind of an analytic tool where the user can visualize his fuel economy in form of a graph.

The application basically revolves around taking distance and fuel measure from the Arduino board via Bluetooth. The mileage is calculated and is stored in a database using SQLite, the details of which will be discussed later.

The application makes use of six classes, which communicate together to fulfill the functionality expected from the application.

Five major classes have been briefly discussed in the following pages:

2.1 DeviceListActivity.java

We shall start our discussion with the first page shown after the Splash Screen.

This activity displays a list of available devices from all the paired devices on your Android phone. The user can connect his device accordingly and proceed to retrieve signals from his car.

To understand how this is done we initially check the state of the Bluetooth adapter. If it is OFF we prompt the user to turn it on. This is done using the **checkBTState()** function:

```
if(mBtAdapter==null) {
          Toast.makeText(getBaseContext(), "Car does not support Bluetooth",
Toast.LENGTH_SHORT).show();
      } else {
        if (mBtAdapter.isEnabled()) {
          Log.d(TAG, "...Bluetooth ON...");
        } else {
          //Prompt user to turn on Bluetooth
          Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
          startActivityForResult(enableBtIntent, 1);
```

We then proceed to display the list of available devices, storing and displayed in an array of paired devices:

```
if (pairedDevices.size() > 0) {

          findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);//m
ake title viewable
                    for (BluetoothDevice device : pairedDevices) {
                              mPairedDevicesArrayAdapter.add(device.getName() +
"\n" + device.getAddress());
                    }
```

Now, we connect our device to the desired device using **OnItemClickListener()**

```
String info = ((TextView) v).getText().toString();
          String address = info.substring(info.length() - 17);

          // Make an intent to start next activity while taking an extra which is
the MAC address.
                    Intent i = new Intent(DeviceListActivity.this,
MainActivity.class);
          i.putExtra(EXTRA_DEVICE_ADDRESS, address);
                    startActivity(i);
```

The following class mainly deals with the Bluetooth functionality of this application. Various functions have been defined which help the Android application to connect with the development board and receive fuel and distance signals from the sensors.

bluetoothIn = new Handler()

A Bluetooth handler is used which facilitates communication between the hardware and the Android mobile phone.

On receiving the desired message, the application now removes the desired information.

recDataString.append(readMessage);

int endOfLineIndex = recDataString.indexOf("~");

int endOfLineIndexa = recDataString.indexOf("+");

int beginningOfLineIndex = recDataString.indexOf("<");

int endOfLineIndexb = recDataString.indexOf(">");

The aforementioned statements are used to scan for the symbols present in the received data. This helps to easily decode the received fuel and distance signals.

String sensor0 = recDataString.substring(1, endOfLineIndexa);

String sensor1 = recDataString.substring(beginningOfLineIndex+1, endOfLineIndexb);

These statements record the values of fuel level and distance and store them into variables sensor1 and sensor 0 respectively using recDataString.substring().

Now that we have understood how the distance and fuel signals are retrieved, we now calculate the fuel economy.

OnClickListener() is used to define what clicking the two buttons does.

On pressing the “Engine Start” button:

```
mConnectedThread.write("1");    // Send "1" via Bluetooth
        Toast.makeText(getBaseContext(), "ENGINE ON",
Toast.LENGTH_SHORT).show();
```

The LED is turned on and the fuel and distance signals are continuously scanned and retrieved.

When the “Engine Stop” button is pressed:

```
mConnectedThread.write("0");    // Send "0" via Bluetooth

Toast.makeText(getBaseContext(), "ENGINE STOP",

String message = "#"+FinalValFuel+"<"+FinalValDist+">";

intent.putExtra(EXTRA_MESSAGE, message);

StartActivity(intent);
```

We send the current values of distance and fuel to the next activity via the intent.

2.3 DisplayMessageActivity.java

This activity retrieves data from the Main Activity after the engine is turned off. The fuel and distance signals are used to calculate the car’s fuel economy and are then stored in a table.

This Activity contains various functions to retrieve fuel and distance signals, calculate the car’s fuel economy and store them in a table.

A function **BuildTable()** is used to create a table to store and show the recorded log of distance, fuel and fuel economy values.

A pseudo code has been shown in the following page:

```
int rows = c.getCount();
            int cols = c.getColumnCount();

            c.moveToFirst();

            // outer for loop
            for (int i = 0; i < rows; i++) {

             TableRow row = new TableRow(this);
             row.setLayoutParams(new LayoutParams(LayoutParams.MATCH_PARENT,
               LayoutParams.WRAP_CONTENT));

             // inner for loop
             for (int j = 0; j < cols; j++) {

              TextView tv = new TextView(this);
              tv.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
                LayoutParams.WRAP_CONTENT));
              tv.setGravity(Gravity.CENTER);
              tv.setTextSize(18);
              tv.setPadding(0, 5, 0, 5);

              tv.setText(c.getString(j));
```

Besides that, the job of calculating the mileage and creating the table is done by SQLController.java and MyDbHelper.java respectively

2.4 MyDbHelper.java

A class has been implemented to create the database. It uses the **SQLiteOpenHelper** which is a helper class to manage database creation and version management.

A database called “FUEL_ECONOMY.DB” is created. It consists of a table called “TABLE_MILEAGE” and columns such as “MILEAGE_ID”, “MILEAGE_DISTANCE”, “MILEAGE_FUEL” and “MILEAGE_VALUE”.

A table is created using the following statement:

```
private static final String CREATE_TABLE = "create table " + TABLE_MILEAGE

   + "(" + MILEAGE_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "

   + MILEAGE_DISTANCE + " TEXT NOT NULL ," + MILEAGE_FUEL

   + " TEXT NOT NULL,"

   +MILEAGE_VALUE+ " TEXT NOT NULL);";
```

The above mentioned query creates a table “MILEAGE_TABLE” with columns, namely – MILEAGE_ID, MILEAGE_DISTANCE., MILEAGE_FUEL and MILEAGE_VALUE.

2.5 SQLController.java

This class contains houses the most important but extremely simple function for mileage calculation and data insertion.

The function looks like:

```
ContentValues cv = new ContentValues();

cv.put(MyDbHelper.MILEAGE_DISTANCE,
dist_val);cv.put(MyDbHelper.MILEAGE_FUEL, fuel_val);

int dist_int_val = Integer.parseInt(dist_val);

int fuel_int_val = Integer.parseInt(fuel_val);

int mileage_int_val = dist_int_val/fuel_int_val;

String mileage_str = String.valueOf(mileage_int_val);

cv.put(MyDbHelper.MILEAGE_VALUE, mileage_str);

database.insert(MyDbHelper.TABLE_MILEAGE, null, cv);
```

CHAPTER - 8

Experimentation and Results

1. Hardware Implementation

1.1 Potentiometer

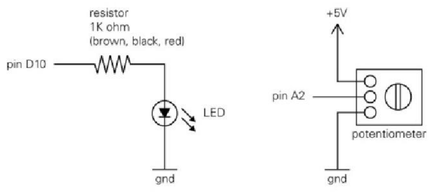


Figure 22 Schematic of a Potentiometer

In order to simulate the working of an accelerating car, we connect a potentiometer to our Arduino board. The three pins are connected to VCC, A2 and GND respectively in the following manner:

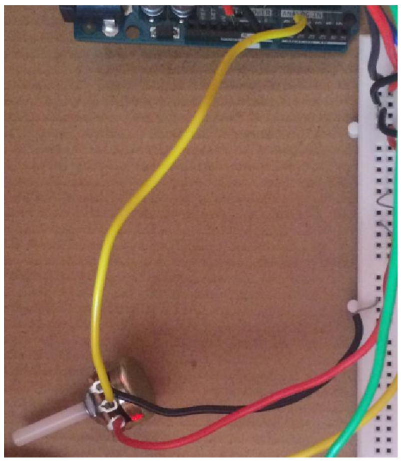


Figure 23 Connecting a Potentiometer to the circuit

1.2 HC-05 Bluetooth Module

Communication between the two major modules – the software and hardware has been facilitated by the HC-05 module. To understand its proper working, a schematic would come in handy:

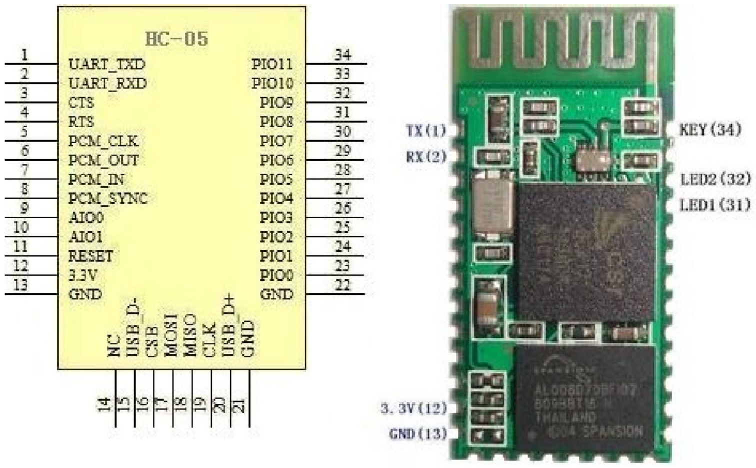


Figure 24 Schematic of Bluetooth Module HC-05

Connections from the HC-05 module to the Arduino Board are pretty simple: You need to connect the VCC and GND pins to where they belong, RX and TX pins switch sides on the board:

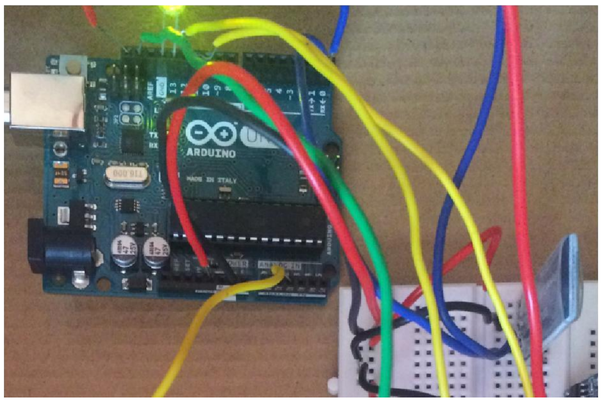


Figure 25 Bluetooth HC-05 Module connected on Arduino Uno

The module is connected at a baud rate of 9600.

1.3 HC-SR04 Ultrasonic Sensor

Now that we’ve furnished two criterions – distance and communication, we further proceed to calculate fuel changes using the HC-SR04 module. To understand the connections more properly, a schematic has been illustrated below:

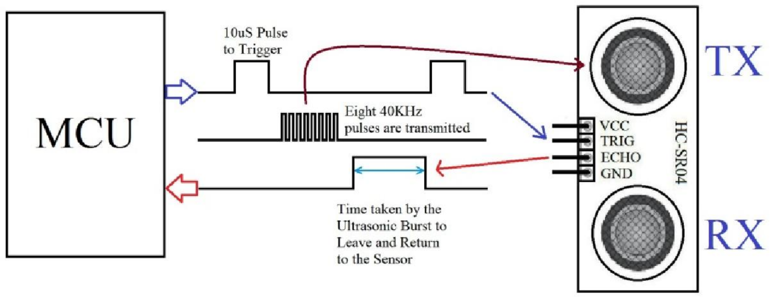

MCU
10uS Pulse to Trigger
Eight 40KHz pulses are transmitted
Time taken by the Ultrasonic Burst to Leave and Return to the Sensor
VCC
TRIG
ECHO
GND
HC-SR04
TX
RX

Figure 26 HC-SR04 Ultrasonic Module Schematic

Connections to the board are pretty much similar to that of the Bluetooth module:

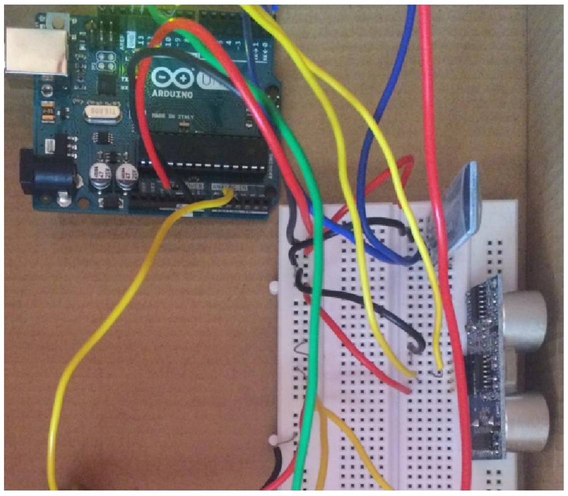


Figure 27 HC-SR04 Ultrasonic Module connected to Arduino Uno

This sensor will help us sense the changes in the fuel level via sending and retrieving ultrasonic waves.

1.4 Grove backlit LCD

In addition to the already mentioned sensors, to take simulation to a next level: we implemented a digital odometer. LCD provided in the Grove kit has been used with a predefined library, the connections are pretty simple:

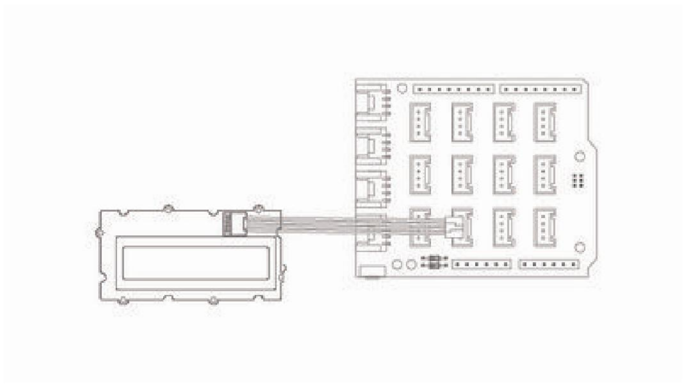


Figure 28 Schematic of the Grove backlit LCD

The above illustration shows the connections to the shielding provided with the kit. I, however, connected the display directly to the Arduino board as follows:

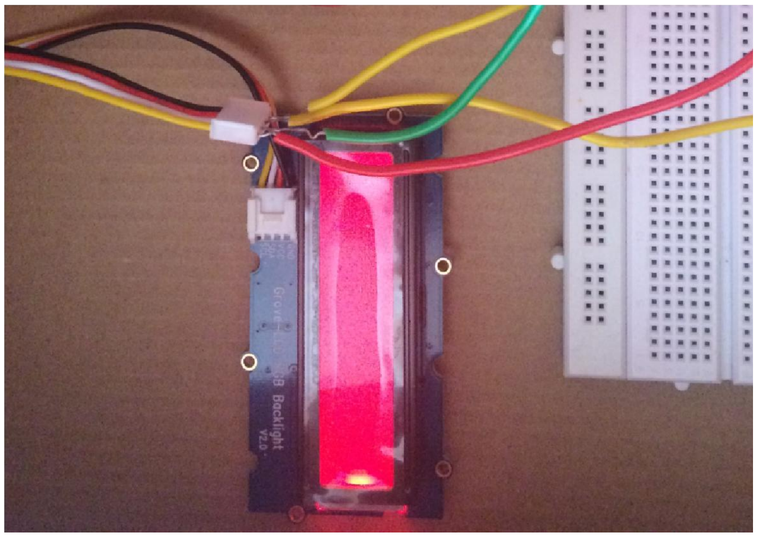


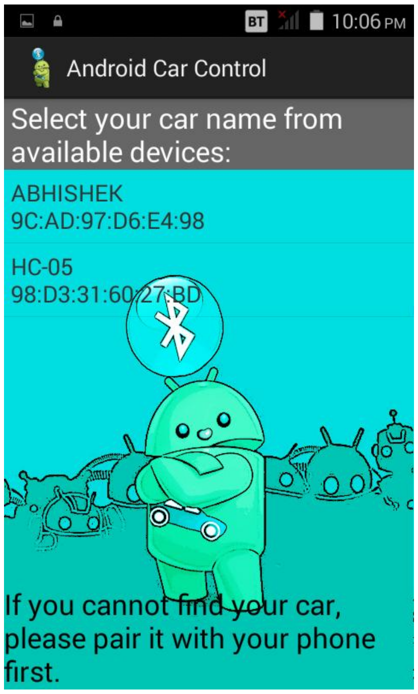
Figure 29 Grove backlit LCD connected smugly to the Board

2. Software Implementation

We shall now proceed to discuss the Android application via illustrations. All major classes have been discussed in detail in previous chapters. Now, we shall look at where these interfaces fit in and help the user to conveniently keep track of his car's fuel economy.

2.1 Selecting Your Bluetooth Device from the List of Available Devices

Firstly, we need to figure a way to connect our Android handheld to the hardware. To do the same, I have created an Activity where a list of paired devices with the phone is shown, on selecting the desired device, its MAC address is passed on to the next Activity. The interface looks like:



Select your car name from available devices:

ABHISHEK
9C:AD:97:D6:E4:98

HC-05
98:D3:31:60:27:BD

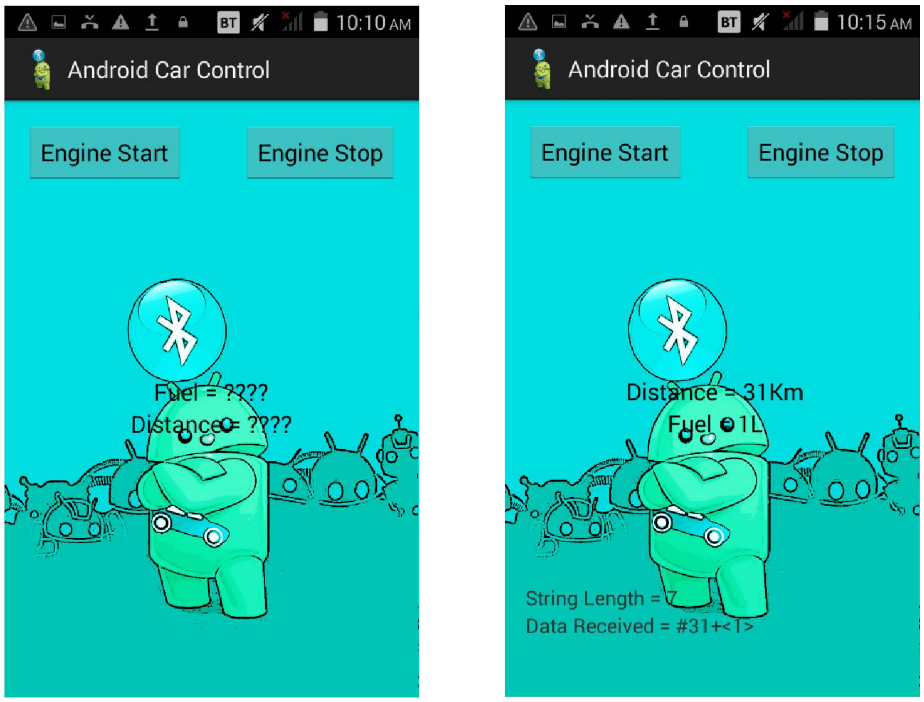
If you cannot find your car, please pair it with your phone first.

Figure 30 Interface Showing the Various Paired Devices

2.2 Viewing Distance and Fuel Values on Your Phone

The retrieved signals from the hardware are shown on an interface on your mobile phone. Two buttons have been provided: “Engine Start” and “Engine Stop”. On pressing the “Engine Start” button, a signal is sent to the device to start receiving signals from the sensor and broadcast them to the handheld.

On pressing the “Engine Stop”, the final values of the fuel and distance are stored and sent to the next Activity in order to be processed and calculate the fuel economy.



Fuel = ????
Distance = ????

Distance = 31Km
Fuel = 1L
String Length = 7
Data Received = #31+<1>

Figure 31 Interface Showing the Fuel and Distance values

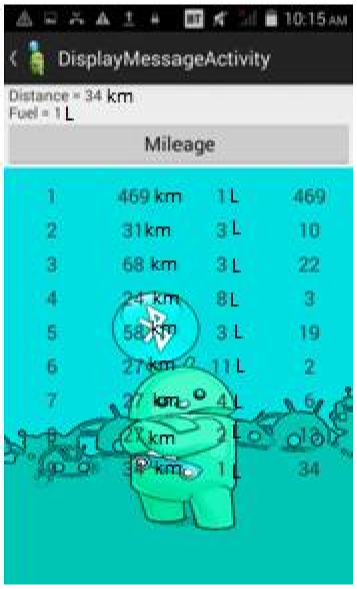
After successful completion of this activity, we proceed to the next step:

2.3 Calculation of Car Mileage and Storing the Values in a Table

After turning off the engine, you will be left with two values – the distance traversed and the fuel level consumed. Now we are left with one final task: calculation of fuel economy.

A simple activity, as already discussed, takes care of this. A database has been created using SQLlite which promptly stores all of the data in a convenient table.

The following illustrations demonstrate the same:



Distance = 34 km
Fuel = 1 L

1	469 km	1 L	469
2	31km	3 L	10
3	68 km	3 L	22
4	24 km	8 L	3
5	58 km	3 L	19
6	27 km	11 L	2
7	27 km	4 L	6
8	[illegible] km	2 L	[illegible]
[illegible]	34 km	1 L	34

Figure 32 Screen Grab of Display Message Activity

The application dutifully completes its responsibility of communication with the hardware, giving a convenient interface and storing the data in a neat table.

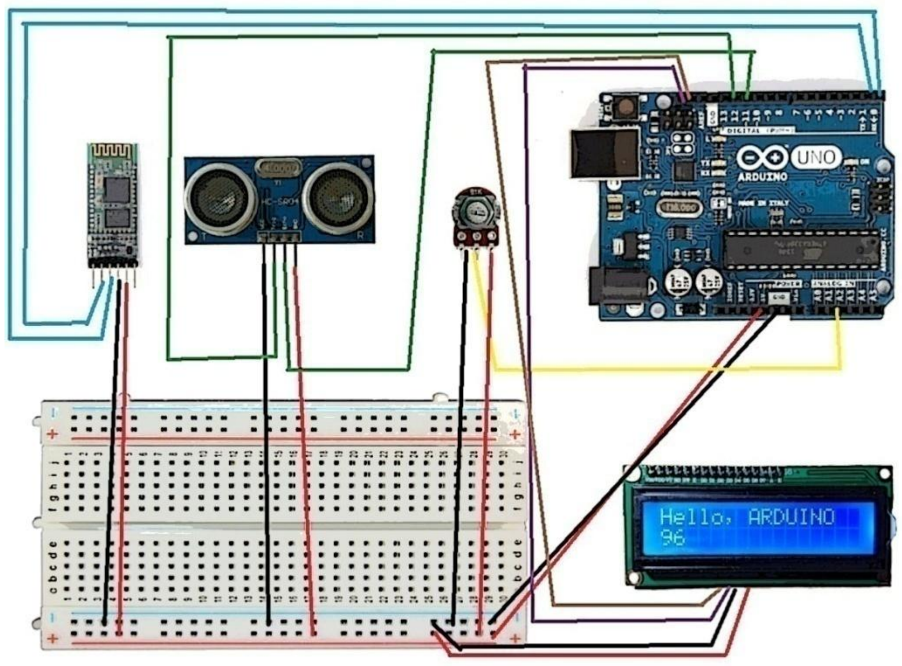
3. Final Run-through of the Project

Now that we've fully understood what went into the making of the project, we shall finally be able to appreciate it working in real time. We'll proceed step-by-step.

Firstly, we'll set-up the apparatus which consists of the following:

- Arduino Uno
- A potentiometer
- A backlit LCD display
- HC-SR04 Ultrasonic Sensor
- HC-05 Bluetooth Module
- A Breadboard
- Connecting Wires

For our convenience, I have put up the circuit diagram for the various connections again:



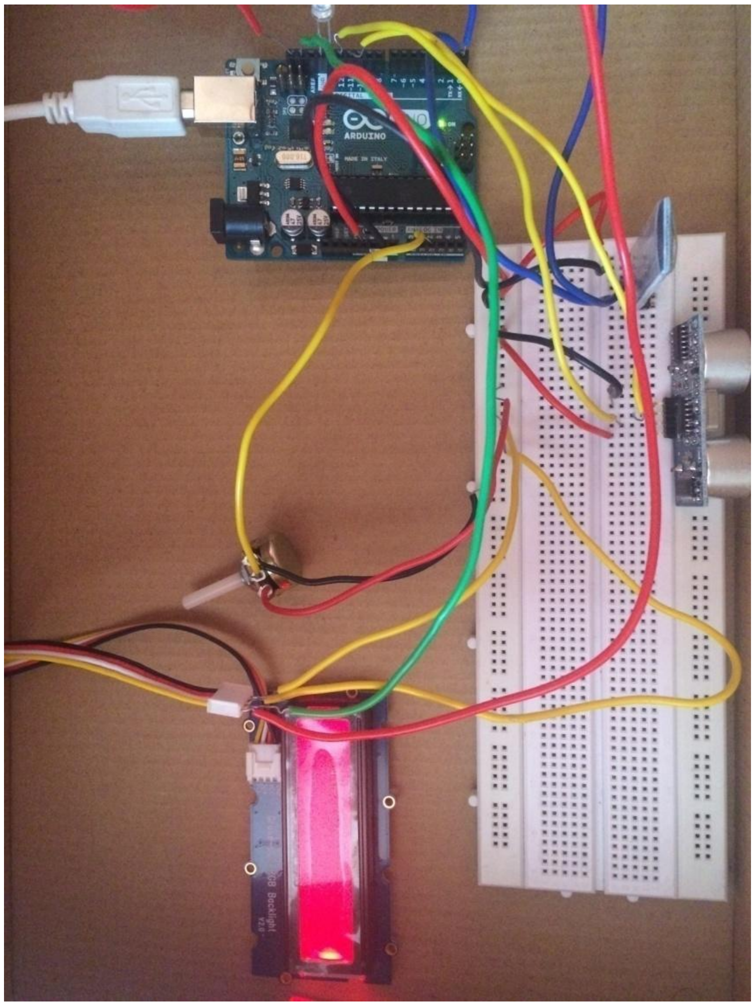


Figure 33 Snapshot of the apparatus

On successfully assembling the circuit, it should look something like this. As soon as you connect it to the appropriate power source, we shall see the backlit LCD light up displaying "ODOMETER".

Now, we're ready to move on to the next step i.e. uploading and loading our Android application on our handheld device.

You can transfer the .apk file directly to your mobile phone and install the application using the steps shown on the screen.

On successful installation and loading the application will look something like this:

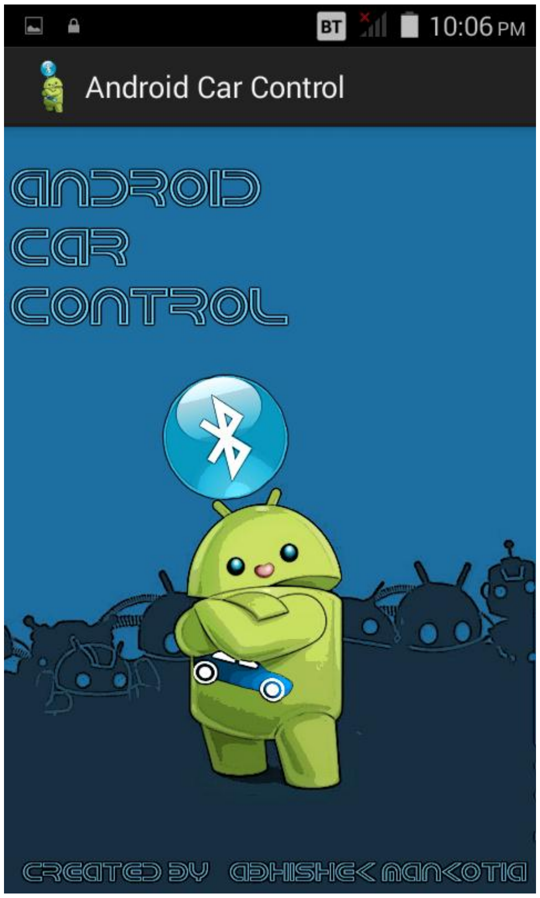
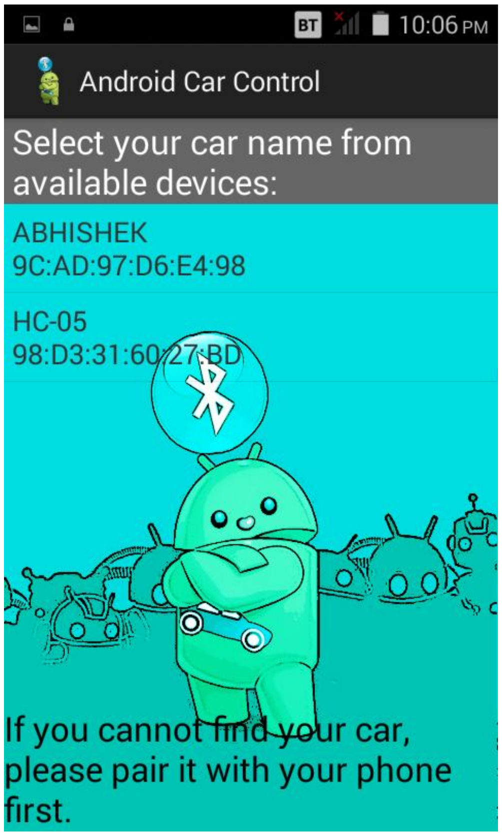


Figure 34 Splash Screen showing the App name and Logo

Figure 8.1.3 shows how the application will look momentarily on being loaded. Splash Screen being a graphical control element consisting of an image, text etc. shown for a fraction of seconds whilst the application prepares for action.

The logo has been created with Adobe Photoshop. The layout has been created in Android.

In a fraction of seconds we shall move to the next screen, the DeviceListActivity.



10:06 PM

Android Car Control

Select your car name from available devices:

ABHISHEK
9C:AD:97:D6:E4:98

HC-05
98:D3:31:60:27:BD

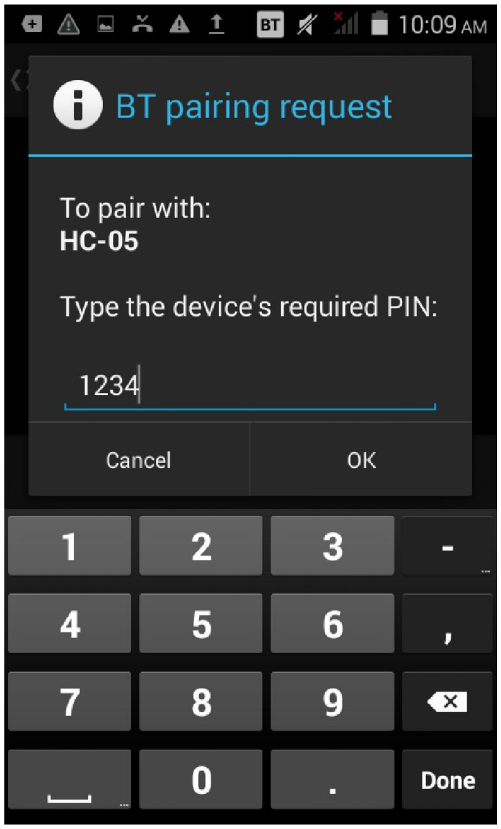
If you cannot find your car, please pair it with your phone first.

Figure 35 Screenshot showing the DeviceListActivity

The following activity shows the list of all the available devices. The device needs to be paired with your Android mobile phone. If you are unable to view your device, try restarting the hardware and pair it with your phone again.

The module we have interfaced with our device is shown on the screen as HC-05.

It has already been paired with our handheld. This can be done by connecting your device in the Bluetooth devices and entering the default passcode “1234” as follows:


BT pairing request

To pair with:
HC-05

Type the device's required PIN:

1234

Figure 36 Screenshot showing the module pairing

On successful pairing the LED on the HC-05 module will blink less frequently, also the device will be shown in the DeviceListActivityScreen.

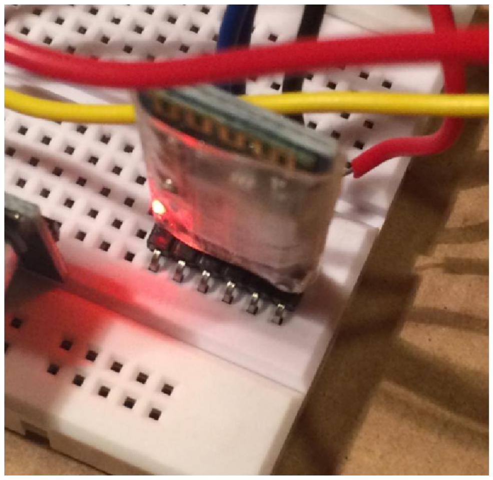
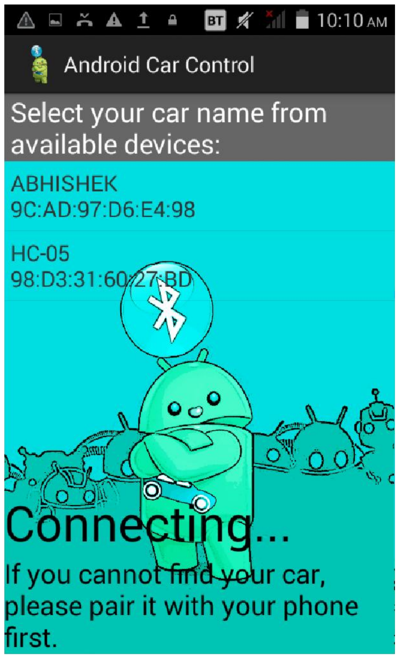


Figure 37 Screenshot of blinking LED on HC-05 module


10:10 AM

Android Car Control

Select your car name from available devices:

ABHISHEK
9C:AD:97:D6:E4:98

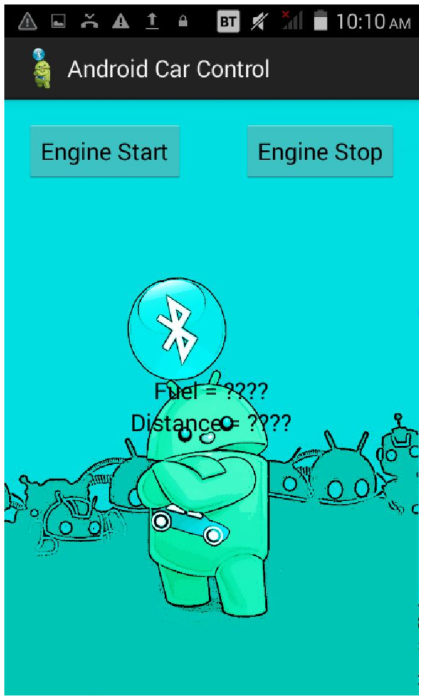
HC-05
98:D3:31:60:27:BD

Connecting...

If you cannot find your car, please pair it with your phone first.

Figure 38 Screenshot showing the application connecting to the module

The application should successfully connect to your mobile phone and the following screen will be displayed with currently no values for distance and fuel:



Fuel = ????
Distance = ????

Figure 39 Screenshot showing the application prior to turning the Engine On

Now, we can finally proceed to turn on the engine. As soon as we press the “Engine Start” button, we notice the LED on the board gradually blinking:

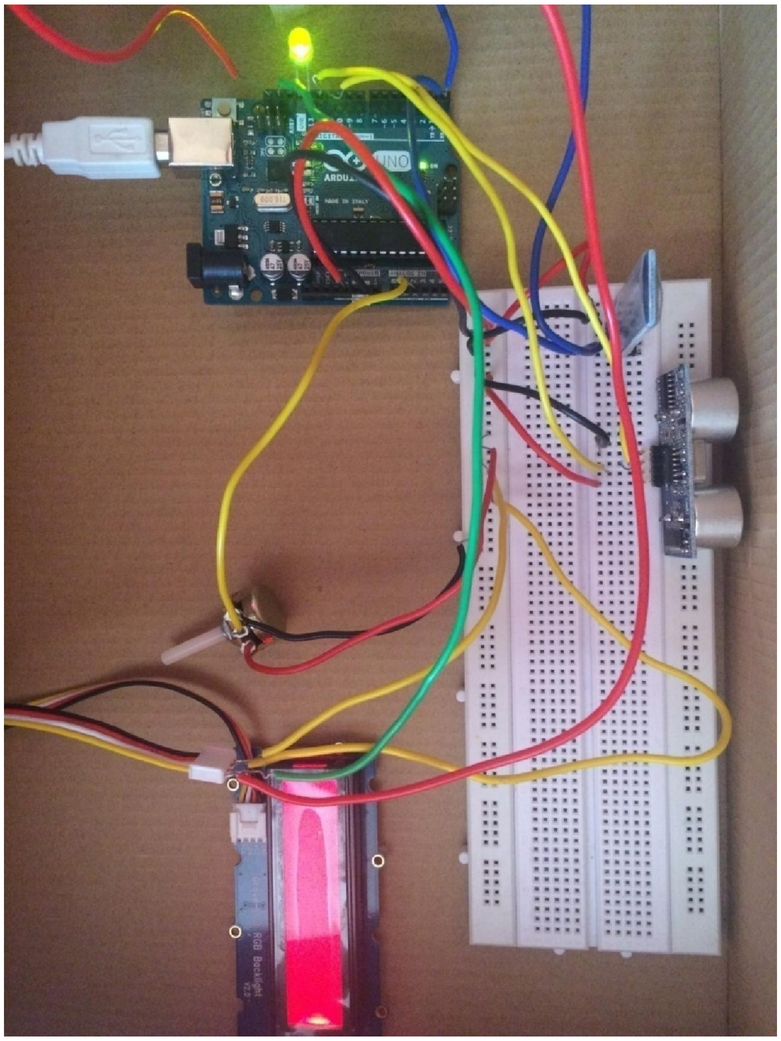


Figure 40 Circuit snapshot after the engine is turned on

Also, we notice the LED no longer shows “ODOMETER”, instead we now view the values of mileage and fuel level.

The distance and fuel signals are continuously emitted from the hardware via the HC-05 module and are received by the Android device. The values are displayed on the mobile screen.

In order to emulate what happens in a real car, we have attached a potentiometer; on rotating the knob we shall notice the value of distance increases faster, symbolizing an increase in the speed of the car:

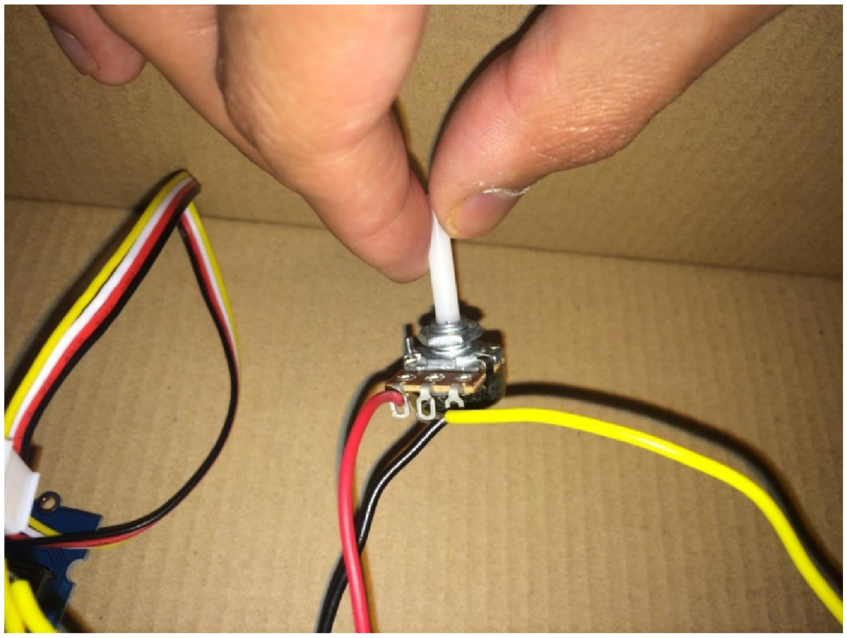


Figure 41 Snapshot of the potentiometer's knob being rotated

On varying the distance in front of the HC-SR04 sensor, we shall notice a change in the fuel level, we therefore successfully emulate the working of a fuel tank sensor:

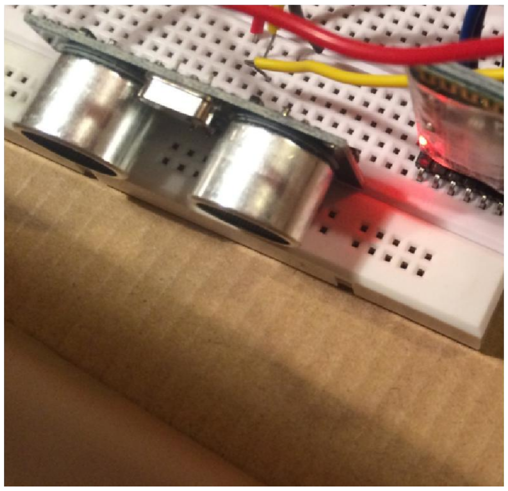


Figure 42 Snapshot of the HC-SR04 Ultrasonic sensor in action

The LCD display acts like a digital odometer showing the present value of mileage and fuel level:

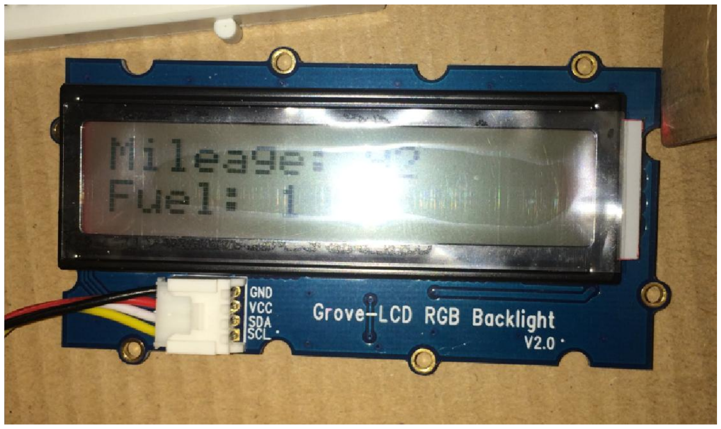
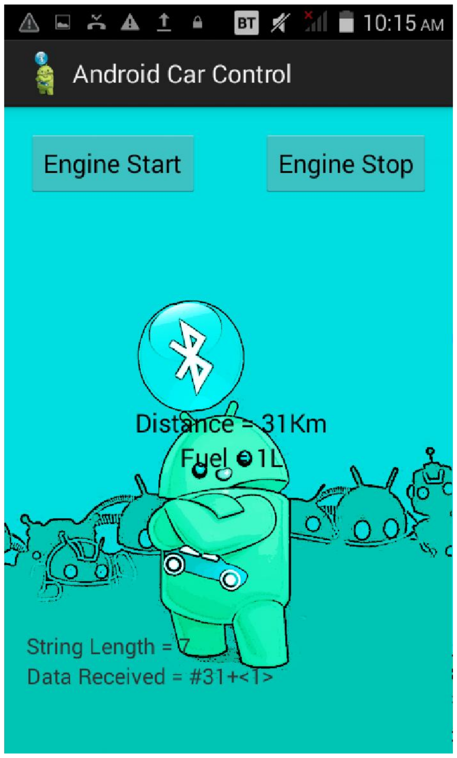


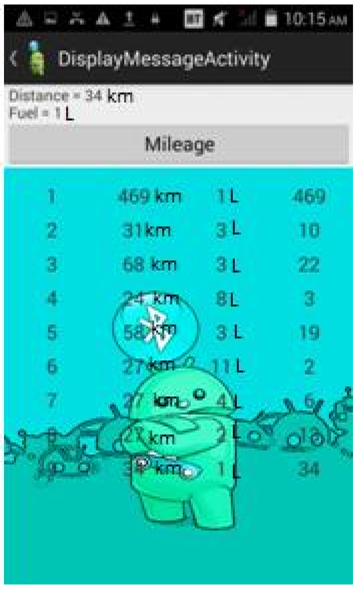
Figure 43 Snapshot of the backlit LCD in action



Distance = 31Km
Fuel = 1L
String Length = 7
Data Received = #31+<1>

Figure 44 Snapshot of the Android application showing mileage and fuel values

We find the corresponding values on the Android application. Now, we proceed to press the “Engine Stop” button, we notice the LED goes OFF, and we are taken to a new screen:



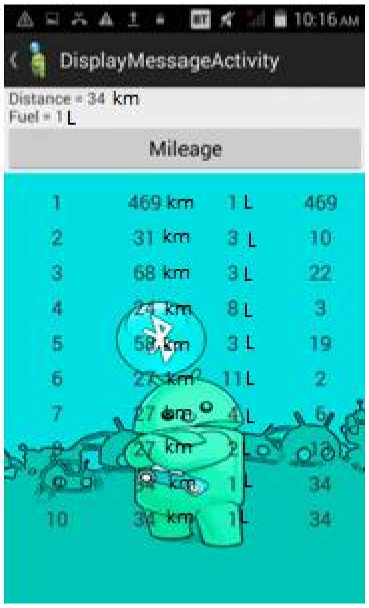
Distance = 34 km
Fuel = 1 L

1	469 km	1 L	469
2	31km	3 L	10
3	68 km	3 L	22
4	24 km	8 L	3
5	58 km	3 L	19
6	27 km	11 L	2
7	27 km	4 L	6
8	[illegible] km	[illegible] L	[illegible]
9	34 km	1 L	34

Figure 45 Screenshot showing the Display Message Activity prior to calculating the mileage

As soon as you click the “ENGINE STOP” button; the existing values of the fuel and distance are sent to the next activity. They are sent using the “intent” functionality of Android.

In order to calculate the mileage, you need to click on the “MILEAGE” button.



Distance = 34 km
Fuel = 1 L

1	469 km	1 L	469
2	31 km	3 L	10
3	68 km	3 L	22
4	24 km	8 L	3
5	58 km	3 L	19
6	27 km	11 L	2
7	27 km	4 L	6
8	27 km	2 L	13
[illegible]	[illegible] km	1 L	34
10	34 km	1 L	34

Figure 46 Screenshot of the Display Message Activity after the mileage has been calculated

On clicking the "MILEAGE" button the fuel and distance signals are fed to a function which calculates the mileage. This value is then stored in a variable and sent to the table to be stored.

This functionality helps the user to keep track of all his previous mileages. This data can be further manipulated for better statistical records.

CHAPTER - 9

Conclusion and Future Work

1. Conclusion

During the course of this project, I understood the importance of embedded systems and its application in solving everyday problems. The problem I took up was to make some kind of a system to control/keep track of various features of a car, beginning with fuel economy.

The solution I came up with was an embedded system interfaced with an Android mobile application. Initially, we planned on implementing only a fuel economy calculator but later, we decided on adding some other features such as a gauge to show the engine's temperature and a light control system. I successfully implemented the fuel economy module, the remaining modules can be taken up in future.

The system successfully generates distance and fuel signals which are sent to the Android mobile phone using the HC-05 module; the fuel economy is then calculated and stored in a table for future reference.

2. Future Work

2.1 Light Control

Another feature that I wish to implement is that of controlling the headlights and tail lamps using infrared sensors. It will make the lights to automatically go on whenever it gets dark and therefore, will provide a handy feature.

2.2 Graphical Visualization

The user can keep track of their vehicle's fuel economy by visualizing graphs on the app.

2.3 Parking Assistance

On completion of the aforementioned additions, I plan on making use of a SONAR sensor for the Arduino board which is readily available in the market and implement some kind of a parking assistance system which the user can visualize on his phone screen.

Reference:

1. Brian W. Evans, “Structure”, Arduino Programming Notebook, First Edition, Creative Commons, 7-29

2. Tim Fulton, “ScanGauge-E Review and Comparison”, Ecomodder

3. Minsk, Belarus, “Engine fuel flow meters DFM”, Technoton

4. The New Boston, “Java/Android Development”, Community forums, thenewboston