**M S Ramaiah Institute of Technology**

(An Autonomous Institute, Affiliated to VTU)

MSR nagar, MSRIT post, Bangalore-54

A Dissertation Report on

**SMART HELMET FOR BIKE**

Submitted by

**Darshan K P** 1MS12CS023

**Anil Pawar** 1MS13CS400

**Sreekanth Naik R** 1MS13CS400

**Saddam Hussian** 1MS13CS418

*in partial fulfillment for the award of the degree of*

# *Bachelor of Engineering in Computer Science & Engineering*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**M.S.RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

[www.msrit.edu](http://www.msrit.edu), **May 2015**

**ABSTRACT**

Alcohol reduces concentration. It decreases reaction time of a human body. Limbs take more to react to the instructions of brain. It hampers vision due to dizziness. Alcohol dampens fear and incites humans to take risks. All these factors while driving cause accidents and many a times it proves fatal. For every increase of 0.05 blood alcohol concentration, the risk of accident doubles. Due to drunk and driving every year thousands of deaths occur particularly for two wheeler accidents and many more are injured. Due to injuries thousands of them will be left crippled with permanent disability.

This not only affects the persons involved in accidents but also to the immediate family

members. Many times those injured are sole bread winners for their family. This also

affects the economy in a great way. Many of such accidents prevented if drunk and

drive cases are detected and preventing such persons from riding the bike. Also head

injuries are major cause of death in two wheeler accidents and many of such cases,

death or severe injuries are prevented if persons wear helmets.

#### Contents

***Declaration i***

***Acknowledgements ii***

***Abstract iii***

***List of Figures***

***List of Tables***

1 **INTRODUCTION 1**

* 1. General Introduction……………….
  2. Statement of the Problem…………..
  3. Objectives of the project……………
  4. Project deliverables……………
  5. Current Scope………………………
  6. Future Scope……………………….

1. **PROJECT ORGANIZATION**
   1. Software Process Models
   2. Roles and Responsibilities
2. **LITERATURE SURVEY**
3. **SOFTWARE REQUIREMENT SPECIFICATIONS**

4.1 Product Overview

4.2 External Interface Requirements

5.2.1 User Interfaces

4.2.2 Hardware Interfaces

4.2.3 Software Interfaces

4.2.4 Communication Interfaces

4.3 Functional Requirements

4.3.1 Functional Requirement 1.1

:

4.3 Functional Requirement 1.n

4.4 Software System Attributes

4.4.1 Reliability

4.4.2 Availability

4.4.3 Security

4.4.4 Portability

4.4.5 Maintainability

4.4.6 Performance

4.5 Performance Requirements

4.6 Other Requirements

1. **DESIGN**
   1. Introduction
   2. Architecture Design
   3. Graphical User Interface
   4. Block Diagram
2. **IMPLEMENTATION**
   1. Tools Introduction
   2. Technology Introduction
   3. Overall view of the project in terms of implementation
   4. Information about the implementation of Modules
3. **TESTING**

**7.1** Results and Snapshots

1. **CONCLUSION & SCOPE FOR FUTURE WORK 44**
2. **REFERENCES**

**1 INTRODUCTION**

**1.1 General Introduction**

Interfacing alcohol detection equipments with helmets and the helmet is connected to bike wirelessly this project aims to prevent drunk and drive. Also the helmet is equipped with sensor to detect if person is not wearing the helmet. If person is drunk or not wearing the helmet then the bike will not start. Several approaches are devised to prevent such accidents like tightening the law and patrolling the streets regularly. But such methods have inherent flaw due to man power shortage and its not possible to patrol every street all the time. Hence our approach will detect such cases not only detect drunk and drive automatically but also report to authorities, ambulance and families of the concerned The helmet is equipped with microcontroller and connected with alcohol detection sensor and obstacle sensor. The alcohol sensor obviously detects the alcohol consumption by rider. The obstacle sensor will detectif the person is wearing the helmet or not.

**1.2 Statement of the problem**

To design a smart helmet and integrate with bike, this helmet will detect if a person or rider is consumed alcohol it will signal the bike and bike engine will be stopped. Also if the rider does not wear the helmet then the engine will stop functioning. This has couple of benefits first the rider will be prevented from riding the bike after consuming the bike. Second the smart helmet will prevent the person from riding the bike without helmet. This prevents accidents due to alcoholism as well as fatalities due to head injuries if rider is not wearing the helmet.

**1.3 Objectives of the Project**

* To design a smart helmet and integrate with bike.
* helmet will detect if a person or rider is consumed alcohol.
* This prevents accidents due to alcoholism.

* It will signal the bike and bike engine will be stopped.

**1.4 PROJECT DELIVERABLES**

The PPT has been prepared and the document has been made on this topic

and same thing has been presented which explains the above mentioned things.

The SRS has been prepared and the Design document has been made on this

topic and same thing has been presented which explains the above mentioned

things.

**1.5 CURRENT SCOPE**

Nowadays bikes are becoming craze for students and youths. Hence it is necessary to save their lives from the accidents due to drunken drive.The helmet is equipped with rasperrypi board and connected with alcohol detection sensor and obstacle sensor or eye sensor. The alcohol sensor obviously detects the alcohol consumption by rider. The obstacle sensor will detect if the person is wearing the helmet or not.

**1.6 FUTURE SCOPE**

In future we have a tendency to planned to construct our intelligent system during a compact size and additionally as globally acceptable to notify the No entry and No parking areas. Government should enforce laws to install such system in each 2 wheeler. y implementing such mechanism in 2 wheelers, the deaths attributable to due to driving and alternative road fatalities are often brought to zero p.c. And also indicates No parking area which would reduce the crowd of the vehicle in those areas. No entry area is mainly allocated during the development or repairing of the road, if the rider enters in such area this system would immediately intimate as No entry area and vehicle can stop automatically. in case of any accident it might send the messages to the friends continuously about the location of the accident happened until the first aid reaches the rider. Our system helps to knowthe location of the vehicle for rescuing in the case oftheft incidents.

1. **PROJECT ORGANIZATION**

**2.1 Software Process Models**

Software for business applications, whether it is intended to perform a single task or it is intended for use as a company-wide, integrated system, should be tailored to fit the company’s unique needs and goals. In the simplest terms, the software should be capable of performing all the functions necessary to perform a task efficiently. While the software should be inclusive, it should not be unduly cumbersome. Careful attention is required to develop software that is both functional and efficient. Once the software is in use, it must be maintained. In this chapter, we will discuss several processes which can serve as guidelines for software development. The term software development life cycle model is a way of describing the planning, designing, coding, and testing of a software system, as well as the method in which these steps are implemented. A variety of life cycle models exist, but they all include the same constituent parts.

All life cycle models take a project through several primary phases: a requirements-gathering phase, a design phase, a construction or implementation phase, and a testing phase. Each phase produces feedback that affects the next phase. For instance, the requirements gathered during the requirements phase influence the design, which is translated into working software code during the implementation phase. The software code is verified against the requirements during the testing phase. During the requirements-gathering phase, the needs of the company are outlined. Managers and users (and in some cases, clients) make their “wish-lists” about what they would like the software to do. Analysts ask questions about the intended use of the software, what type of data will be processed, how the software should handle the data, and how the data can be accessed once in the system. Following the requirements phase, the software development team should have a detailed list of functions that the system will perform. Emphasis is on the system's goals, rather than the way in which the system will achieve those goals.

In the design phase, the results of the requirements-gathering phase are translated into a software design plan. Focus shifts from the system's results to the way in which those results will be achieved and how the ideas of the requirements-gathering phase are accomplished. Designers consider many different criteria, from the hardware and operating system platform that hosting the software to the way subsystems will communicate with each other. In essence, during the design phase, the designers attempt to turn the dreams of the managers and users into reality. Emphasis during this phase is on making a practical, working design for what has been outlined in the requirements-gathering phase.

In the implementation phase, the results of the design phase are translated into program code. Software that does not meet the needs of the company is wasteful. During this phase the programmers should make it their central goal to fulfill the requirements of the company and to meet the design outlined in the design phase. The classes and class interactions developed in the design phase are very explicit. They translate directly into the code generated in the implementation phase. Later in this course, design tools will be introduced that actually automate code generation from the output of the design phase.

In the testing phase, the results of the implementation phase are run through a series of tests to verify that it functions and that it meets goals of the requirements phase. A testing plan is created to describe the unit tests and system tests that will be performed. Unit testing is performed on individual software components. The process of integration brings together all the software components to create a complete system. System testing is performed on the software system as a whole. All software development life cycle models share common phases of development: gathering of requirements, designing of the software system, coding of software, and the testing of the system. The Waterfall life cycle model is one of the simplest and easiest to use; it consists of five discrete phases that are executed sequentially. The V-Shaped life cycle model adds an emphasis on testing to the Waterfall model. The Incremental life cycle model applies a series of iterations to the Waterfall model. The Spiral life cycle model builds upon the Waterfall and Incremental models and focuses on risk analysis.

**2.2 Roles and responsibilities :**

Sreekanth Nayak R – Installations of operating System and configurations of hardware components.

Darshan K P - preparing the design of the system, coding and implementing the system.

Saddam Hussain - Connection of hardware components, Getting the results and testing it.

Anil Pawar – Collecting the dataset, performing the tests and

documenting the details.

**3 LITERATURE SURVEY:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sr no | Paper Studied | |  |  | Old Feature |  | New Feature | |  |  |
|  |  |  |  |  | |  | |  | | |  |
|  | 1 | IJSRD | - | International | | A. Helmet Checking | | A.Helmet checking via ir | | |  |
|  |  | Journal | for | Scientific | | Application |  | transmittor |  |  |  |
|  |  | Research & Development | | | | B.Sobriety Test Application | | B.Alcohol detection | | |  |
|  |  |  |  |  |  | C..Exhaus | tChecking | C.No exhaust checking | | |  |
|  |  |  |  |  |  | Application |  |  |  |  |  |
|  |  |  | |  |  |  | |  |  |  |  |
|  | 2 | International | | Journal | of | A.Robotic helmet unit | | CODEC |  |  |  |
|  |  | Science | andResearch | | | B.headphones,speker | |  |  |  |  |
|  |  | (IJSR) |  |  |  | C.Gps satellite unit | |  |  |  |  |
|  |  | ISSN (Online): 2319‐7064 | | | | d.CODEC |  |  |  |  |  |
|  |  |  | |  |  |  | |  | | |  |
|  | 3 | International | | Journal | of | A.Helmet checking | | A.Helmet checking via ir | | |  |
|  |  | Scientific | & | Engineering | | B.No entry no parking | | transmittor |  |  |  |
|  |  | Research Volume 2, Issue | | | |  |  | B.Alcohol detection | | |  |
|  |  | 12,December-20111 | | | |  |  | C.Sms | to | family |  |
|  |  | ISSN 2229-5518 IJSER | | |  |  |  | members | in | case of |  |
|  |  |  |  |  |  |  |  | accident |  |  |  |
|  |  |  | | |  |  |  |  |  |  |  |

**4 Software Requirement Specification**

* 1. **Product Overview**

The Product smart helmet and integrate with bike, this helmet will detect if a person orrider is consumed alcohol it will signal the bike and bike engine will be stopped. Also if the rider does not wear the helmet then the engine will stop functioning. This has couple of benefits first the rider will be prevented from riding the bike after consuming the bike. Second the smart helmet will prevent the person from riding the bike without helmet. This prevents accidents due to alcoholism as well as fatalities due to head injuries if rider is not wearing the helmet.

**4.2 External Interface Requirements :**

# 4.2.1 User Interfaces

The product must be usable from the text command line, particularly under operating systems where the command line is a standard common user interface (such as Linux). However, some guidelines do apply:

**Help.** Help should be available for all tools. Command-line tools should print a helpful message (if appropriate) if invoked with no arguments (if any command line tools are designed to accept input as a pipe, it is acceptable to forego this rule). If multiple options are available, the product should present a text menu detailing the probable choices rather than forcing the user to retype everything up to that point.

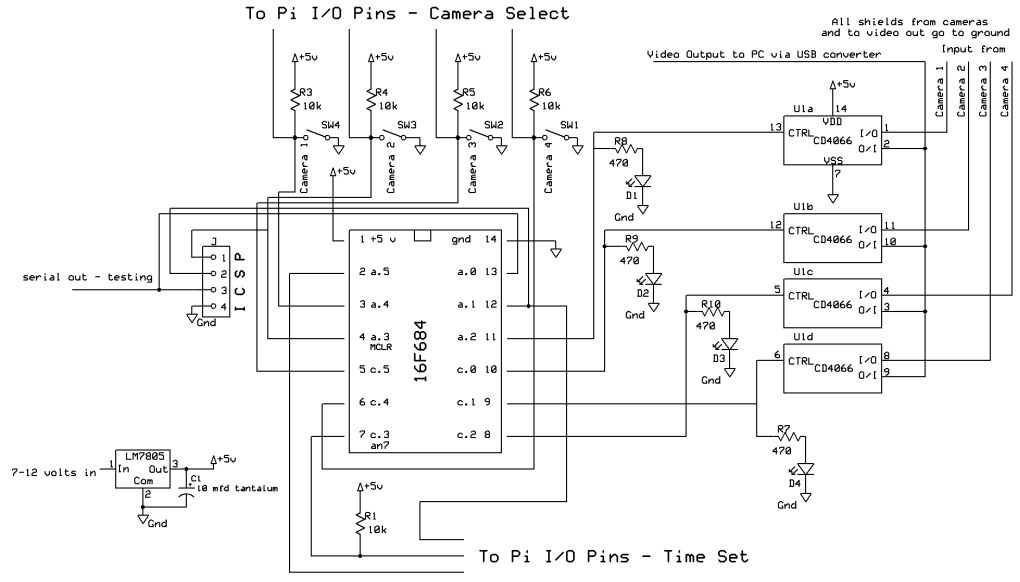
**Feedback.**Each tool should give feedback to the user describing the results of their last action; in other words, issue submissions should be acknowledged (with a response message from standard output, a follow-on web page, or return email) with some positive indication of the result.

**Command-line options.:**If command-line tools can accept arguments, they will support them according to the standards laid out in the GNU coding standards, and should support both short and long forms.

**4.2.2 Hardware description :**

**Raspberry Pi :**

In the above block diagram for model *A, B, A+, B+*; model *A* and *A+* have the lowest two blocks and the rightmost block missing (note that these three blocks are in a chip that actually contains a three-port USB hub, with a USB Ethernet adapter connected to one of its ports). In model *A* and *A+* the USB port is connected directly to the SoC. On model *B+* the chip contains a five-point hub, with four USB ports fed out, instead of the two on model *B*.



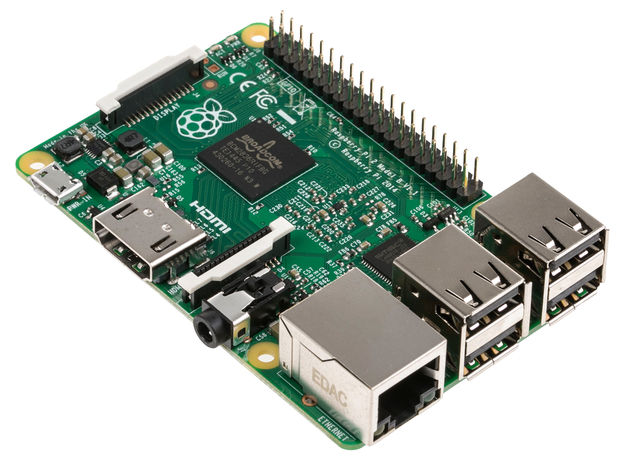
**Figure 4.1 Raspberry Pi circuit Diagram**

### Processor

The [SoC](http://en.wikipedia.org/wiki/System_on_a_chip) used in the first generation Raspberry Pi is somewhat equivalent to the chip used in older [smartphones](http://en.wikipedia.org/wiki/Smartphone) (such as[iPhone](http://en.wikipedia.org/wiki/IPhone) / [3G](http://en.wikipedia.org/wiki/IPhone_3G) / [3GS](http://en.wikipedia.org/wiki/IPhone_3GS)). The Raspberry Pi is based on the [Broadcom](http://en.wikipedia.org/wiki/Broadcom) BCM2835 [system on a chip](http://en.wikipedia.org/wiki/System_on_a_chip) (SoC),[[1]](http://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-Broadcom-BCM2835-Website-1) which includes an 700 [MHz](http://en.wikipedia.org/wiki/Hertz) [ARM1176JZF-S](http://en.wikipedia.org/wiki/ARM11) processor, [VideoCore](http://en.wikipedia.org/wiki/VideoCore) IV GPU,[[7]](http://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-grandmax_brose_2012-7) and RAM. It has a Level 1 [cache](http://en.wikipedia.org/wiki/CPU_cache) of 16 [KB](http://en.wikipedia.org/wiki/Kibibyte) and a Level 2 [cache](http://en.wikipedia.org/wiki/CPU_cache) of 128 [KB](http://en.wikipedia.org/wiki/Kibibyte). The Level 2 cache is used primarily by the GPU. The SoC is [stacked](http://en.wikipedia.org/wiki/Package_on_package) underneath the RAM chip, so only its edge is visible.

#### RAM

On the older beta model B boards, 128 MB was allocated by default to the GPU, leaving 128 MB for the CPU. On the first 256 MB release model B (and model A), three different splits were possible. The default split was 192 MB (RAM for CPU), which should be sufficient for standalone 1080p video decoding, or for simple 3D, but probably not for both together. 224 MB was for Linux only, with just a 1080p [framebuffer](http://en.wikipedia.org/wiki/Framebuffer), and was likely to fail for any video or 3D. 128 MB was for heavy 3D, possibly also with video decoding (e.g. XBMC). Comparatively the Nokia 701 uses 128 MB for the Broadcom VideoCore IV. For the new model B with 512 MB RAM initially there were new standard memory split files released( arm256\_start.elf, arm384\_start.elf, arm496\_start.elf) for 256 MB, 384 MB and 496 MB CPU RAM (and 256 MB, 128 MB and 16 MB video RAM). But a week or so later the RPF released a new version of start elf that could read a new entry in config.txt (gpu\_mem=xx) and could dynamically assign an amount of RAM (from 16 to 256 MB in 8 MB steps) to the GPU, so the older method of memory splits became obsolete, and a single start elf worked the same for 256 and 512 MB Pis. The second generation has 1 GB of RAM.



**Figure 4.2 Raspberry PI**

The Raspberry Pi is a series of [credit card](http://en.wikipedia.org/wiki/Credit_card)-sized [single-board computers](http://en.wikipedia.org/wiki/Single-board_computer) developed in the [UK](http://en.wikipedia.org/wiki/United_Kingdom) by the [Raspberry Pi Foundation](http://en.wikipedia.org/wiki/Raspberry_Pi_Foundation) with the intention of promoting the teaching of basic [computer science](http://en.wikipedia.org/wiki/Computer_science) in schools.

The original Raspberry Pi and Raspberry Pi 2 are manufactured in several board configurations through licensed manufacturing agreements with [Newark element14](http://en.wikipedia.org/wiki/Newark_element14) ([Premier Farnell](http://en.wikipedia.org/wiki/Premier_Farnell)), [RS Components](http://en.wikipedia.org/wiki/RS_Components) and Egoman. These companies sell the Raspberry Pi online.Egoman produces a version for distribution solely in China and Taiwan, which can be distinguished from other Pis by their red colouring and lack of [FCC](http://en.wikipedia.org/wiki/FCC_Declaration_of_Conformity)/[CE marks](http://en.wikipedia.org/wiki/CE_marking). The hardware is the same across all manufacturers.

The original Raspberry Pi is based on the [Broadcom](http://en.wikipedia.org/wiki/Broadcom) BCM2835 [system on a chip](http://en.wikipedia.org/wiki/System_on_a_chip) (SoC), which includes an [ARM1176JZF-S](http://en.wikipedia.org/wiki/ARM11)700 [MHz](http://en.wikipedia.org/wiki/Hertz) processor, [VideoCore](http://en.wikipedia.org/wiki/VideoCore) IV GPU, and was originally shipped with 256 megabytes of [RAM](http://en.wikipedia.org/wiki/Random-access_memory), later upgraded (models B and B+) to 512 [MB](http://en.wikipedia.org/wiki/Megabyte). The system has [Secure Digital](http://en.wikipedia.org/wiki/Secure_Digital) (SD) (models A and B) or MicroSD (models A+ and B+) sockets for boot media and persistent storage

In 2014, the Raspberry Pi Foundation launched the Compute Module, which packages a BCM2835 with 512 MB RAM and an[eMMC](http://en.wikipedia.org/wiki/MultiMediaCard#eMMC) flash chip into a module for use as a part of embedded systems.

The Foundation provides Debian and Arch Linux ARM [distributions](http://en.wikipedia.org/wiki/Linux_distribution) for download. Tools are available for [Python](http://en.wikipedia.org/wiki/Python_(programming_language)) as the main programming language, with support for [BBC BASIC](http://en.wikipedia.org/wiki/BBC_BASIC) (via the [RISC OS](http://en.wikipedia.org/wiki/RISC_OS) image or the Brandy Basic clone for Linux), [C](http://en.wikipedia.org/wiki/C_(programming_language)), [C++](http://en.wikipedia.org/wiki/C%2B%2B), [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), [Perl](http://en.wikipedia.org/wiki/Perl) and [Ruby](http://en.wikipedia.org/wiki/Ruby_(programming_language)).

As of 18 February 2015, over five million Raspberry Pis have been sold. While already the fastest selling British [personal computer](http://en.wikipedia.org/wiki/Personal_computer), it has also shipped the second largest number of units behind the [Amstrad PCW](http://en.wikipedia.org/wiki/Amstrad_PCW), the "Personal Computer [Word-processor](http://en.wikipedia.org/wiki/Word-processor)", which sold eight million.

In early February 2015, the next-generation Raspberry Pi, Raspberry Pi 2, was officially announced. The new computer board will initially be available only in one configuration (model B) and features a Broadcom BCM2836 SoC, with a [quad-core](http://en.wikipedia.org/wiki/Multi-core_processor)[ARM Cortex-A7](http://en.wikipedia.org/wiki/ARM_Cortex-A7) CPU and a VideoCore IV dual-core GPU; 1 GB of RAM with remaining specifications being similar to those of the previous generation model B+. Crucially, the Raspberry Pi 2 will retain the same US$35 price point of the model B, with the US$25 model A remaining on sale.

**Liquid crystal display**

Main articles: Liquid-crystal display and Thin-film-transistor liquid-crystal display

There are multiple technologies that have been used to implement liquid crystal displays (LCD). Throughout the 1990s, the primary use of LCD technology as computer monitors was in laptops where the lower power consumption, lighter weight, and smaller physical size of LCDs justified the higher price versus a CRT. Commonly, the same laptop would be offered with an assortment of display options at increasing price points: (active or passive) monochrome, passive color, or active matrix color (TFT). As volume and manufacturing capability have improved, the monochrome and passive color technologies were dropped from most product lines.

TFT-LCD is a variant of LCD which is now the dominant technology used for computer monitors.

The first standalone LCD displays appeared in the mid-1990s selling for high prices. As prices declined over a period of years they became more popular, and by 1997 were competing with CRT monitors. Among the first desktop LCD computer monitors was the Eizo L66 in the mid-1990s, the Apple Studio Display in 1998, and the Apple Cinema Display in 1999. In 2003, TFT-LCDs outsold CRTs for the first time, becoming the primary technology used for computer monitors.The main advantages of LCDs over CRT displays are that LCDs consume less power, take up much less space, and are considerably lighter. The now common active matrix TFT-LCD technology also has less flickering than CRTs, which reduces eye strain.On the other hand, CRT monitors have superior contrast, have superior response time, are able to use multiple screen resolutions natively, and there is no discernible flicker if the refresh rate is set to a sufficiently high value. LCD monitors have now very high temporal accuracy and can be used for vision research.

**4.2.3 Software Interface:**

**LINUX (UBUNTU):**

**Features**

A [default](http://en.wikipedia.org/wiki/Default_(computer_science)) installation of Ubuntu contains a wide range of software that includes [Libre Office](http://en.wikipedia.org/wiki/LibreOffice), [Firefox](http://en.wikipedia.org/wiki/Mozilla_Firefox), [Thunderbird](http://en.wikipedia.org/wiki/Mozilla_Thunderbird), [Transmission](http://en.wikipedia.org/wiki/Transmission_(BitTorrent_client)), and several lightweight games such as [Sudoku](http://en.wikipedia.org/wiki/Sudoku) and [chess](http://en.wikipedia.org/wiki/GNOME_Chess). Many additional software packages, including titles no longer in the default installation such as [Evolution](http://en.wikipedia.org/wiki/Evolution_(software)), [GIMP](http://en.wikipedia.org/wiki/GIMP), [Pidgin](http://en.wikipedia.org/wiki/Pidgin_(software)), and [Synaptic](http://en.wikipedia.org/wiki/Synaptic_(software)), are accessible from the built in [Ubuntu Software Center](http://en.wikipedia.org/wiki/Ubuntu_Software_Center) as well as any other [APT](http://en.wikipedia.org/wiki/Advanced_Packaging_Tool) based [package management](http://en.wikipedia.org/wiki/Package_management) tool. Execution of [Microsoft Office](http://en.wikipedia.org/wiki/Microsoft_Office) and other [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows) applications can be facilitated via the [Wine com](http://en.wikipedia.org/wiki/Wine_(software)) through the use of a virtual machine such as [Virtual Box](http://en.wikipedia.org/wiki/VirtualBox) or [VMware kstation](http://en.wikipedia.org/wiki/VMware_Workstation)

Following are some of the important features of Linux Operating System*.*

* Portable - Portability means softwares can works on different types of hardwares in same way.
* Open Source - Linux source code is freely available and it is community based development project.
* Multi-User - Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
* Multiprogramming - Linux is a multiprogramming system means multiple applications can run at same time.
* Hierarchical File System - Linux provides a standard file structure in which system files/ user files are arranged.
* Shell - Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
* Security - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

**Security**

Ubntu's goal is to be secure "out-of-the box". By default, the user's programs run with [low previleges](http://en.wikipedia.org/wiki/Principle_of_least_privilege) and cannot corrupt the operating system or other user's files. For increased security,\* the [sudo](http://en.wikipedia.org/wiki/Sudo) tool is used to assign temporary privileges for performing adative tasks, which allows the [root account](http://en.wikipedia.org/wiki/Root_account) to remain locked and helps prevent inexerienced users from inadvertently making catastrophic system changes or opening secrityholes.[Policy Kit](http://en.wikipedia.org/wiki/PolicyKit) is also being widely implemented into the desktop to further [harden](http://en.wikipedia.org/wiki/Security-focused_operating_system) he system. Most network ports are closed by default to prevent hacking. built-infiewall allows end-users who install network servers to control access. A [GUI](http://en.wikipedia.org/wiki/GUI) (GUI for Uncomplicated Firewall) is available to configure it.  Ubuntu compiles its packages using [GCC](http://en.wikipedia.org/wiki/GNU_Compiler_Collection) features such as [PIE](http://en.wikipedia.org/wiki/Position-independent_code) and [harden](http://en.wikipedia.org/wiki/Hardening_(computing)) its software. These extra features greatly increase security at the performance expense of 1% in [32 bit](http://en.wikipedia.org/wiki/X86) and 0.01% in [64 bit](http://en.wikipedia.org/wiki/X86-64). The home and Private directories can be encrypted.

**EMBEDDED C**

In 2008, the C Standards Committee extended the C language to address these issues by providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as, fixed-point arithmetic, named address spaces, and basic I/O hardware addressing.

Embedded C use most of the syntax and semantics of standard C, e.g., main() function, variable definition, data type declaration, conditional statements (if, switch. case), loops (while, for), functions, arrays and strings, structures and union, bit operations, macros, unions, etc.

**Embedded systems programming**

Embedded systems programming is different from developing applications on a desktop computers. Key characteristics of an embedded system, when compared to PCs, are as follows:

* Embedded devices have resource constraints (limited ROM, limited RAM, limited stack space, less processing power)
* Components used in embedded system and PCs are different; embedded systems typically uses smaller, less power consuming components. Embedded systems are more tied to the hardware.

Two salient features of Embedded Programming are code speed and code size. Code speed is governed by the processing power, timing constraints, whereas code size is governed by available program memory and use of programming language.  Goal of embedded system programming is to get maximum features in minimum space and minimum time.

Embedded systems are programmed using different type of languages:

* Machine Code
* Low level language, i.e., assembly
* High level language like C, C++, Java, Ada, etc.
* Application level language like Visual Basic, scripts, Access, etc.

**Necessity**

During infancy years of microprocessor based systems, programs were developed using assemblers and fused into the EPROMs. There used to be no mechanism to find what the program was doing. LEDs, switches, etc. were used to check for correct execution of the program. Some ‘very fortunate’ developers had In-circuit Simulators (ICEs), but they were too costly and were not quite reliable as well. As time progressed, use of microprocessor-specific assembly-only as the programming language reduced and embedded systems moved onto C as the embedded programming language of choice. C is the most widely used programming language for embedded processors/controllers. Assembly is also used but mainly to implement those portions of the code where very high timing accuracy, code size efficiency, etc. are prime requirements.

As assembly language programs are specific to a processor, assembly language didn’t offer portability across systems. To overcome this disadvantage, several high level languages, including C, came up. Some other languages like PLM, Modula-2, Pascal, etc. also came but couldn’t find wide acceptance. Amongst those, C got wide acceptance for not only embedded systems, but also for desktop applications. Even though C might have lost its sheen as mainstream language for general purpose applications, it still is having a strong-hold in embedded programming. Due to the wide acceptance of C in the embedded systems, various kinds of support tools like compilers & cross-compilers, ICE, etc. came up and all this facilitated development of embedded systems using C. Assembly language seems to be an obvious choice for programming embedded devices. However, use of assembly language is restricted to developing efficient codes in terms of size and speed. Also, assembly codes lead to higher software development costs and code portability is not there. Developing small codes are not much of a problem, but large programs/projects become increasingly difficult to manage in assembly language. Finding good assembly programmers has also become difficult nowadays. Hence high level languages are preferred for embedded systems programming.

**MultiThreading:**

* A Thread is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables.
* Each application has at least one thread running when it is started, the main thread, in the main ThreadGroup .
* The runtime keeps its own threads in the systemthread group.

**4.2.4 Communication Interfaces :**

Alcohol Sensor

GPIO interface

Microcontroller to read and process the data. The data is transmitted thorough Wi-Fi

Wi-Fi for

Tx/Rx

Wi-Fi for

Tx/Rx

ARM 7 Controllers

Helmet Controller

Signal Receive& processing

Convert signal to Commands

Vehicle

Tx commands to Vehicle

BLOCK DIAGRAM FOR MOUTH BLOW AIR & Eye blink sensor

IR Eye Blink Sensor

GPIO interface

GPIO interface

Relay switches

The thought of developing this project comes to do some good things towards the society. Day by day the two wheeler accidents are increasing and leads to loss of many lives. Accord to a survey of India there are around 698 accidents occurring due to bike crashes per year. The reasons may be many such as no proper driving knowledge, no fitness of the bike, fast riding of bike, drunken and drive etc. Some time the person injured, the accident may not be directly responsible for the accident, it may be fault of rider, but end of the day it’s both the drivers involved in the accidents who is going to suffer. If accidents are one issue, lack of treatment in proper time is another reason for deaths. According to the survey India 698 accidents occur per year, nearly half the injured people die due to lack of treatment in proper time. The many reasons for this such as late arrival of ambulance, no persons at place where the accident occur to give information to the ambulance or parents. This is a situation we observe our day to day life, a thought of finding some solution to resolve this problem come up with this idea of giving the information about accident as soon as possible and in TIME….!!!!Because after all time matters a lot, if everything is done in time, at least we can save half the lives that are lost due to bike accidents. Considering three major factors for avoiding the accident causes such as I. Make wearing the helmet compulsory. II. Avoid drunk and drive. III. If person met with an accident, no one is there to help him. Simply leaving or ignoring the person he may die. In such situation, informing to ambulance or family members through mobile to rescue him for an extent. The idea of this work is to give information about the rider wearing the helmet or not, whether the rider drunken or not and also, he met with an accident it gives an information about location where he is met with an accident through GSM module to mobile numbers family members, so I have chosen GSM technology to give the information by sending SMS, using GSM module which has SIM card slot to place the SIM and send SMS. Sending SMS alone can’t help the driver, if we send and an SMS saying that accident had occurred where the ambulance will come without knowing the location of the accident. So to trace out the location where exactly accident occur using GPS module, and gives to microcontroller, then it sends the SMS which contains the latitude and longitude of a area to family members mobile numbers For this we use GPS module to extract the location of the accident, the GPS data will contain the latitude and longitude values using which we can find the accurate position of the accident place.

**4.3 FUNCTIONAL REQUIREMENTS :**

The functional requirements describe the core functionality of the application. This section includes the data and functional process requirements. Generally, the ConOps is the source for specified or derived requirements. 4.1 Functional Process Requirements Process requirements describe what the application must do. Process requirements relate the entities and attributes from the data requirements to the users’ needs. State the functional process requirements in a manner that enables the reader to see broad concepts decomposed into layers of increasing detail. Exhibit 1-2 is a sample of process requirements levels stated in such a manner. It is presented only as an example of requirements headings decomposed from generalities into levels of increasing detail.

Some of the more typical functional requirements include:

* + - 1. Alcohol Detection
      2. Sleeping monitoring

4.3.1 Helmet mounted Alcohol Detection sensor

The helmet will contain an ARM-7 controller like Raspberry Pi board. The microcontroller will be connected with alcohol detection sensor using GPIO pins. The bike will have another microcontroller. The bike microcontroller is interfaced with engine start stop function. The helmet mounted microcontroller is connected with bike using either Wi-Fi or Bluetooth devices. When bike is switched on the bike microcontroller waits for the signal from helmet mounted microcontroller. If bike microcontroller receives alcohol signal from helmet controller then it will switch off the bike. In case bike controller receives non-alcoholic signal from helmet controller then it will enable the engine.

4.3.2 Obstacle sensor

Many government authorities enacted law to make wearing helmet compulsory but implementing same is proving to be difficult as it need large pool of police to patrol the streets. Hence an attempt is made to make such implementation automatic. The helmet mounted controller is connected to obstacle sensor to detect if person is wearing the helmet or not prior to starting the bike. The controller mounted on bike will wait for the signal from helmet mounted controller before starting the engine. In case the person is not wearing the bike then helmet mounted controller will send negative signal to bike controller then the controller will not start the bike. In case the helmet controller sends positive signal indicating the person is wearing the helmet then the bike controller will start the engine. In this way we can make wearing the helmet compulsory.

**4.4 SOFTWARE SYSTEM ATTRIBUTES :**

**Safety and Security** :

Many software-intensive systems have significant safety and security ramifications and need to have their associated safety- and security-related requirements properly engineered. For example, it has been observed by several consultants, researchers, and authors that inadequate requirements are a major cause of accidents involving software-intensive systems. Yet in practice, there is very little interaction between the requirements, safety and security disciplines and little collaboration between their respective communities. Most requirements engineers know little about safety and security engineering, and most safety and security engineers know little about requirements engineering. Also, safety and security engineering typically concentrates on architectures and designs rather than requirements because hazard and threat analysis typically depend on the identification of vulnerable hardware and software components, the failure or exploitation of which can cause accidents and enable successful attacks. This leads to safety- and security-related requirements that are often ambiguous, incomplete, and even missing.

**Software Quality Attributes:**

**Following factors are used to measure software development quality:**

Each attribute can be used to measure the product performance. These attributes can be used for Quality assurance as well as Quality control. Quality Assurance activities are oriented towards prevention of introduction of defects and Quality control activities are aimed at detecting defects in products and services.

* + 1. **Reliability**   
       Measure if product is reliable enough to sustain in any condition. give consist ently correct results.Product reliability is measured in terms of working of project under different working environment and different conditions.
    2. **Maintainability**   
       Different versions of the product should be easy to maintain. For development its should be easy to add code to existing system, should be easy to upgrade for new features and new technologies time to time. Maintenance should be cost effective and easy. System be easy to maintain and correcting defects or making a change in the software.
    3. **Usability**  
       this can be measured in terms of ease of use. Application should be user friendly. The system must be:
* Easy to use for input preparation, operation, and interpretation of output.
* Provide consistent user interface standards or conventions with our other frequently used systems.
* Easy for new or infrequent users to learn to use the system.
  + 1. **Portability**  
       This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioral issues related to porting.
    2. **Correctness**  
       Application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means application should adhere to functional requirements.
    3. **Efficiency**  
       To Major system quality attribute. Measured in terms of time required to complete any task given to the system. For example system should utilize processor capacity, disk space and memory efficiently. If system is using all the available resources then user will get degraded performance failing the system for efficiency. If system is not efficient then it can not be used in real time applications.

**4.5 PERFORMANCE REQUIREMENTS :**

At a minimum, a set of performance requirements should document the following: The maximum satisfactory response time to be experienced most of the time for each distinct type of user-computer interaction, along with a definition of *most of the time*. Response time is measured from the time that the user performs the action that says "Go" until the user receives enough feedback from the computer to continue the task. It is the user's subjective wait time. It is not from entry to a subroutine until the first write statement.

If the user denies interest in response time and indicates that only the result is of interest, you can ask whether "ten times your current estimate of stand-alone execution time" would be acceptable. If the answer is "yes," you can proceed to discuss throughput. Otherwise, you can continue the discussion of response time with the user's full attention.

* The response time that is minimally acceptable the rest of the time. A longer response time can cause users to think the system is down. You also need to specify *rest of the time*; for example, the peak minute of a day, 1 percent of interactions. Response time degradations can be more costly or painful at a particular time of the day.
* The typical throughput required and the times it will be taking place. This is not a casual consideration. For example, the requirement for one program might be that it runs twice a day: at 10:00 a.m. and 3:15 p.m. If this is a CPU-limited program that runs for 15 minutes and is planned to run on a multiuser system, some negotiation is in order.
* The size and timing of maximum-throughput periods.
* The mix of requests expected and how the mix varies with time.
* The number of users per machine and total number of users, if this is a multiuser application. This description should include the times these users log on and off, as well as their assumed rates of keystrokes, completed requests, and think times. You may want to investigate whether think times vary systematically with the preceding and following request.
* Any assumptions that the user is making about the machines the workload will run on. If the user has a specific existing machine in mind, make sure you know that early on. Similarly, if the user is assuming a particular type, size, cost, location, interconnection, or any other variable that will constrain your ability to satisfy the preceding requirements, that assumption also becomes part of the requirements. Satisfaction will probably not be assessed on the system where the program is developed, tested, or first installed.

**4.6 OTHER REQUIREMENTS :**

Requirements are statements of what the system must do, how it must behave, the properties it must exhibit, the qualities it must possess, and the constraints that the system and its development must satisfy a condition or capability needed by a user to solve a problem or achieve an objective

1. a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document
2. a documented representation of a condition or capability as in definition 1 or 2 [[IEEE 1990a](http://www.sei.cmu.edu/productlines/frame_report/frame_bib.htm#IEEE1990a)]

Requirements engineering emphasizes the use of systematic and repeatable techniques that ensure the completeness, consistency, and relevance of the system requirements [[Sommerville 1997a](http://www.sei.cmu.edu/productlines/frame_report/frame_bib.htm#Sommerville1997a)]. Specifically, requirements engineering encompasses requirements elicitation, analysis, specification, verification, and management, where

* **Requirements elicitation** is the process of discovering, reviewing, documenting, and understanding the user's needs and constraints for the system.
* **Requirements analysis** is the process of refining the user's needs and constraints.
* **Requirements specification** is the process of documenting the user's needs and constraints clearly and precisely.
* **Requirements verification** is the process of ensuring that the system requirements are complete, correct, consistent, and clear.
* **Requirements management** is the process of scheduling, coordinating, and documenting the requirements engineering activities (that is, elicitation, analysis, specification, and verification) [[Dorfman 1997a](http://www.sei.cmu.edu/productlines/frame_report/frame_bib.htm#Dorfman1997a)].

1. **DESIGNS**

**5.1 Introduction**

Interfacing alcohol detection equipments with helmets and the helmet is connected to bike wirelessly this project aims to prevent drunk and drive. Also the helmet is equipped with sensor to detect if person is not wearing the helmet. If person is drunk or not wearing the helmet then the bike will not start.

* Several approaches are devised to prevent such accidents like tightening the law and patrolling the streets regularly. But such methods have inherent flaw due to man power shortage and its not possible to patrol every street all the time.

Hence our approach will detect such cases not only detect drunk and drive automatically but also report to authorities, ambulance and families of the concerned person in case of accident.

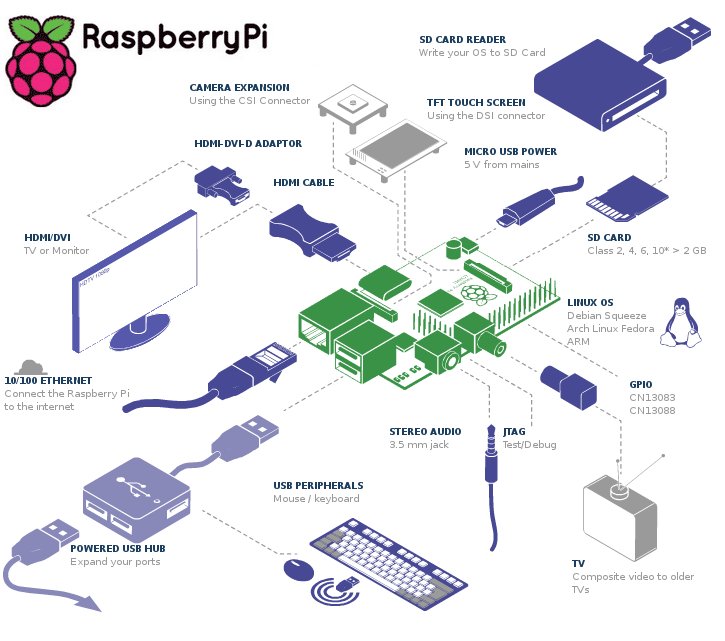
The helmet is equipped with microcontroller and connected with alcohol detection sensor and obstacle sensor or eye sensor. The alcohol sensor obviously detects the alcohol consumption by rider. The obstacle sensor or eye sensor will detect if the person is wearing the helmet or not. The GPS module will report about the location of the accident to family and hospital and police.

* 1. **Architecture Design**

The Foundation provides Debian and Arch Linux ARM [distributions](http://en.wikipedia.org/wiki/Linux_distribution) for download. Tools are available for [Python](http://en.wikipedia.org/wiki/Python_(programming_language)) as the main programming language, with support for [BBC BASIC](http://en.wikipedia.org/wiki/BBC_BASIC) (via the [RISC OS](http://en.wikipedia.org/wiki/RISC_OS) image or the Brandy Basic clone for Linux), [C](http://en.wikipedia.org/wiki/C_(programming_language)), [C++](http://en.wikipedia.org/wiki/C%2B%2B), [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), [Perl](http://en.wikipedia.org/wiki/Perl) and [Ruby](http://en.wikipedia.org/wiki/Ruby_(programming_language)).

As of 18 February 2015, over five million Raspberry Pis have been sold. While already the fastest selling British [personal computer](http://en.wikipedia.org/wiki/Personal_computer), it has also shipped the second largest number of units behind the [Amstrad PCW](http://en.wikipedia.org/wiki/Amstrad_PCW), the "Personal Computer [Word-processor](http://en.wikipedia.org/wiki/Word-processor)", which sold eight million.

In early February 2015, the next-generation Raspberry Pi, Raspberry Pi 2, was officially announced. The new computer board will initially be available only in one configuration (model B) and features a Broadcom BCM2836 SoC, with a [quad-core](http://en.wikipedia.org/wiki/Multi-core_processor)[ARM Cortex-A7](http://en.wikipedia.org/wiki/ARM_Cortex-A7) CPU and a VideoCore IV dual-core GPU; 1 GB of RAM with remaining specifications being similar to those of the previous generation model B+. Crucially, the Raspberry Pi 2 will retain the same US$35 price point of the model B, with the US$25 model A remaining on sale.



**5.3 Graphical User Interface**

**[](http://en.wikipedia.org/wiki/File:Nexus_S_with_Ubuntu.jpg)**

The system requirements vary among Ubuntu products. For the Ubuntu desktop release 14.04, a PC with at least 768 [MB](http://en.wikipedia.org/wiki/Megabyte) of [RAM](http://en.wikipedia.org/wiki/Random-access_memory) and 5 [GB](http://en.wikipedia.org/wiki/Gigabyte) of disk space is recommended. For less powerful computers, there are other Ubuntu distributions such as [Lubuntu](http://en.wikipedia.org/wiki/Lubuntu) and [Xubuntu](http://en.wikipedia.org/wiki/Xubuntu). As of version 12.04,Ubuntu supports the [ARM architecture](http://en.wikipedia.org/wiki/ARM_architecture).Ubuntu is also available on [PowerPC](http://en.wikipedia.org/wiki/PowerPC),and [SPARC](http://en.wikipedia.org/wiki/SPARC) platforms,although these platforms are not officially supported.

[Live images](http://en.wikipedia.org/wiki/Live_image) are the typical way for users to assess and subsequently install Ubuntu. These can be downloaded as a disk image ([.iso](http://en.wikipedia.org/wiki/ISO_image)) and subsequently burnt to a DVD and booted, or run via [UNetbootin](http://en.wikipedia.org/wiki/UNetbootin) directly from a USB drive (making, respectively, a [live DVD](http://en.wikipedia.org/wiki/Live_DVD) or [live USB](http://en.wikipedia.org/wiki/Live_USB) medium). Running Ubuntu in this way is typically slower than running it from a [hard drive](http://en.wikipedia.org/wiki/Hard_drive), but does not alter the computer unless specifically instructed by the user. If the user chooses to boot the live image rather than execute an installer at boot time, there is still the option to then use an installer called [Ubiquity](http://en.wikipedia.org/wiki/Ubiquity_(software)) to install Ubuntu once booted into the live environment. [Disk images](http://en.wikipedia.org/wiki/Disk_image) of all current and past versions are available for download at the Ubuntu web site. Various third-party programs such as [remastersys](http://en.wikipedia.org/wiki/Remastersys) and [Reconstructor](http://en.wikipedia.org/wiki/Reconstructor) are available to create customized copies of the Ubuntu Live DVDs (or CDs). "Minimal CDs" are available (for server use) that fit on a CD.

Additionally, [USB flash drive](http://en.wikipedia.org/wiki/USB_flash_drive) installations can be used to boot Ubuntu and Kubuntu in a way that allows permanent saving of user settings and portability of the USB-installed system between physical machines (however, the computers' [BIOS](http://en.wikipedia.org/wiki/BIOS) must support booting from USB).

In newer versions of Ubuntu, the [Ubuntu Live USB creator](http://en.wikipedia.org/wiki/Ubuntu_Live_USB_creator) can be used to install Ubuntu on a USB drive (with or without a live CD or DVD). Creating a bootable USB drive with [persistence](http://en.wikipedia.org/wiki/Persistence) is as simple as dragging a slider to determine how much space to reserve for persistence; for this, Ubuntu employs [casper](http://en.wikipedia.org/wiki/Casper_(persistency)).

**5.4 Block Diagram**

This paper mainly focuses on avoidance of drunken driving. Hence this system will not turn on the vehicle, when the user is in drunken condition. In addition to this, it will not allow the user to park drive the vehicle in the no parking or no entry area respectively. The system will send short message service to the friends or relatives when an accident occurs. It also employs theft detection. Our system consists of two major parts. 1) Helmet unit and 2) Vehicle unit as shown in fig.1 & 2.

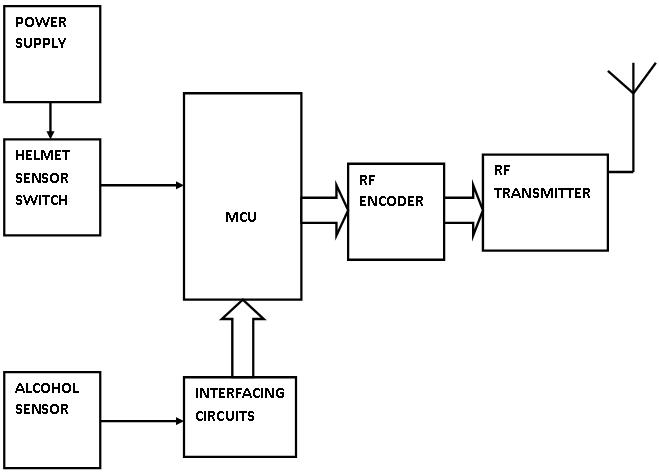


Figure 1. Helmet Unit.

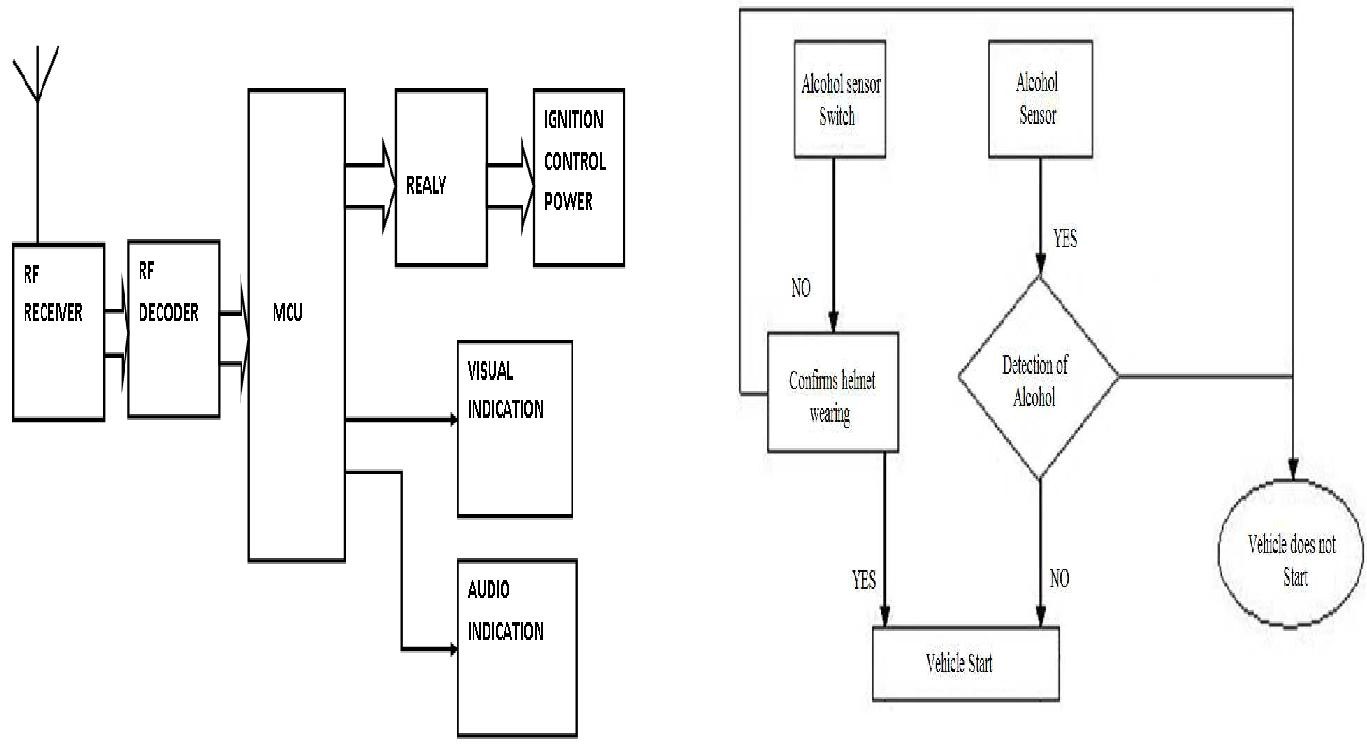
****

Fig 2. Vehicle Unit Fig 3. Flow chart of electronic smart helmet system.

**6 IMPLEMENTATION**

**6.1 Tools Introduction**

**EMBEDDED C**

In 2008, the C Standards Committee extended the C language to address these issues by providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as, fixed-point arithmetic, named address spaces, and basic I/O hardware addressing.

Embedded C use most of the syntax and semantics of standard C, e.g., main() function, variable definition, data type declaration, conditional statements (if, switch. case), loops (while, for), functions, arrays and strings, structures and union, bit operations, macros, unions, etc.

**Embedded systems programming**

Embedded systems programming is different from developing applications on a desktop computers. Key characteristics of an embedded system, when compared to PCs, are as follows:

* Embedded devices have resource constraints (limited ROM, limited RAM, limited stack space, less processing power)
* Components used in embedded system and PCs are different; embedded systems typically uses smaller, less power consuming components. Embedded systems are more tied to the hardware.

Two salient features of Embedded Programming are code speed and code size. Code speed is governed by the processing power, timing constraints, whereas code size is governed by available program memory and use of programming language.  Goal of embedded system programming is to get maximum features in minimum space and minimum time.

Embedded systems are programmed using different type of languages:

* Machine Code
* Low level language, i.e., assembly
* High level language like C, C++, Java, Ada, etc.
* Application level language like Visual Basic, scripts, Access, etc.

**NECESSITY**

During infancy years of microprocessor based systems, programs were developed using assemblers and fused into the EPROMs. There used to be no mechanism to find what the program was doing. LEDs, switches, etc. were used to check for correct execution of the program. Some ‘very fortunate’ developers had In-circuit Simulators (ICEs), but they were too costly and were not quite reliable as well. As time progressed, use of microprocessor-specific assembly-only as the programming language reduced and embedded systems moved onto C as the embedded programming language of choice. C is the most widely used programming language for embedded processors/controllers. Assembly is also used but mainly to implement those portions of the code where very high timing accuracy, code size efficiency, etc. are prime requirements.As assembly language programs are specific to a processor, assembly language didn’t offer portability across systems. To overcome this disadvantage, several high level languages, including C, came up. Some other languages like PLM, Modula-2, Pascal, etc. also came but couldn’t find wide acceptance. Amongst those, C got wide acceptance for not only embedded systems, but also for desktop applications. Even though C might have lost its sheen as mainstream language for general purpose applications, it still is having a strong-hold in embedded programming. Due to the wide acceptance of C in the embedded systems, various kinds of support tools like compilers & cross-compilers, ICE, etc. came up and all this facilitated development of embedded systems using C. Assembly language seems to be an obvious choice for programming embedded devices. However, use of assembly language is restricted to developing efficient codes in terms of size and speed. Also, assembly codes lead to higher software development costs and code portability is not there. Developing small codes are not much of a problem, but large programs/projects become increasingly difficult to manage in assembly language. Finding good assembly programmers has also become difficult nowadays. Hence high level languages are preferred for embedded systems programming.

**MultiThreading:**

* A Thread is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables.
* Each application has at least one thread running when it is started, the main thread, in the main ThreadGroup .
* The runtime keeps its own threads in the systemthread group.
  1. **Technology Introduction**

**Raspberry Pi**

In the above block diagram for model *A, B, A+, B+*; model *A* and *A+* have the lowest two blocks and the rightmost block missing (note that these three blocks are in a chip that actually contains a three-port USB hub, with a USB Ethernet adapter connected to one of its ports). In model *A* and *A+* the USB port is connected directly to the SoC. On model *B+* the chip contains a five-point hub, with four USB ports fed out, instead of the two on model *B*.

### Processor

The [SoC](http://en.wikipedia.org/wiki/System_on_a_chip) used in the first generation Raspberry Pi is somewhat equivalent to the chip used in older [smartphones](http://en.wikipedia.org/wiki/Smartphone) (such as[iPhone](http://en.wikipedia.org/wiki/IPhone) / [3G](http://en.wikipedia.org/wiki/IPhone_3G) / [3GS](http://en.wikipedia.org/wiki/IPhone_3GS)). The Raspberry Pi is based on the [Broadcom](http://en.wikipedia.org/wiki/Broadcom) BCM2835 [system on a chip](http://en.wikipedia.org/wiki/System_on_a_chip) (SoC),[[1]](http://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-Broadcom-BCM2835-Website-1) which includes an 700 [MHz](http://en.wikipedia.org/wiki/Hertz) [ARM1176JZF-S](http://en.wikipedia.org/wiki/ARM11) processor, [VideoCore](http://en.wikipedia.org/wiki/VideoCore) IV GPU,[[7]](http://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-grandmax_brose_2012-7) and RAM. It has a Level 1 [cache](http://en.wikipedia.org/wiki/CPU_cache) of 16 [KB](http://en.wikipedia.org/wiki/Kibibyte) and a Level 2 [cache](http://en.wikipedia.org/wiki/CPU_cache) of 128 [KB](http://en.wikipedia.org/wiki/Kibibyte). The Level 2 cache is used primarily by the GPU. The SoC is [stacked](http://en.wikipedia.org/wiki/Package_on_package) underneath the RAM chip, so only its edge is visible.

#### RAM

On the older beta model B boards, 128 MB was allocated by default to the GPU, leaving 128 MB for the CPU. On the first 256 MB release model B (and model A), three different splits were possible. The default split was 192 MB (RAM for CPU), which should be sufficient for standalone 1080p video decoding, or for simple 3D, but probably not for both together. 224 MB was for Linux only, with just a 1080p [framebuffer](http://en.wikipedia.org/wiki/Framebuffer), and was likely to fail for any video or 3D. 128 MB was for heavy 3D, possibly also with video decoding (e.g. XBMC). Comparatively the Nokia 701 uses 128 MB for the Broadcom VideoCore IV. For the new model B with 512 MB RAM initially there were new standard memory split files released( arm256\_start.elf, arm384\_start.elf, arm496\_start.elf) for 256 MB, 384 MB and 496 MB CPU RAM (and 256 MB, 128 MB and 16 MB video RAM). But a week or so later the RPF released a new version of start elf that could read a new entry in config.txt (gpu\_mem=xx) and could dynamically assign an amount of RAM (from 16 to 256 MB in 8 MB steps) to the GPU, so the older method of memory splits became obsolete, and a single start elf worked the same for 256 and 512 MB Pis. The second generation has 1 GB of RAM.

**LINUX (UBUNTU):**

**Features**

A [default](http://en.wikipedia.org/wiki/Default_(computer_science)) installation of Ubuntu contains a wide range of software that includes [Libre Office](http://en.wikipedia.org/wiki/LibreOffice), [Firefox](http://en.wikipedia.org/wiki/Mozilla_Firefox), [Thunderbird](http://en.wikipedia.org/wiki/Mozilla_Thunderbird), [Transmission](http://en.wikipedia.org/wiki/Transmission_(BitTorrent_client)), and several lightweight games such as [Sudoku](http://en.wikipedia.org/wiki/Sudoku) and [chess](http://en.wikipedia.org/wiki/GNOME_Chess). Many additional software packages, including titles no longer in the default installation such as [Evolution](http://en.wikipedia.org/wiki/Evolution_(software)), [GIMP](http://en.wikipedia.org/wiki/GIMP), [Pidgin](http://en.wikipedia.org/wiki/Pidgin_(software)), and [Synaptic](http://en.wikipedia.org/wiki/Synaptic_(software)), are accessible from the built in [Ubuntu Software Center](http://en.wikipedia.org/wiki/Ubuntu_Software_Center) as well as any other [APT](http://en.wikipedia.org/wiki/Advanced_Packaging_Tool) based [package management](http://en.wikipedia.org/wiki/Package_management) tool. Execution of [Microsoft Office](http://en.wikipedia.org/wiki/Microsoft_Office) and other [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows) applications can be facilitated via the [Wine com](http://en.wikipedia.org/wiki/Wine_(software)) through the use of a virtual machine such as [Virtual Box](http://en.wikipedia.org/wiki/VirtualBox) or [VMware Kstation](http://en.wikipedia.org/wiki/VMware_Workstation).

* 1. **Overall view of the project in terms of implementation**

Implementation Done on our project using C Programming Code.

<----------smart\_helmet.c---------->

#include <stdio.h>

#include <wiringPi.h>

// LED Pin - wiringPi pin 0 is BCM\_GPIO 17.

#define LED 0 // GPIO 17

#define LED1 1 // GPIO 18

#define LED3 3 // GPIO 22

int main (void)

{

printf ("Raspberry Pi - Gertboard Blink\n") ;

int pin = 0;

int pin1 = 0;

wiringPiSetup () ;

pinMode (LED, INPUT) ;

pinMode (LED1, INPUT) ;

pinMode (LED3, OUTPUT) ;

for (;;)

{

// delay (500) ; // mS

pin = digitalRead (LED ) ; // Off

pin1 = digitalRead (LED1 ) ; // Off

printf ("Pin - Obstacle OUTPUT: %d \n", pin) ;

printf ("Pin - Alchohol OUTPUT: %d \n", pin1) ;

if(pin == 1 && pin1 == 0)

{

digitalWrite (LED3, 1) ; // On

printf ("\n\n <===== SWITCHING ON MOTORCYCLE ====>\n\n") ;

delay (1000) ; // mS

}

else

{

digitalWrite (LED3, 0) ; // Off

printf ("\n\n <===== WARNING:::: SWITCHING OFF MOTORCYCLE ====>\n\n") ;

delay (1500) ; // mS

}

delay (500) ;

}

return 0 ;

}

<----------smart\_helmet\_bk.c---------->

#include <stdio.h>

#include <wiringPi.h>

// LED Pin - wiringPi pin 0 is BCM\_GPIO 17.

#define LED 0 // GPIO 17

#define LED1 1 // GPIO 18

#define LED3 3 // GPIO 22

int main (void)

{

printf ("Raspberry Pi - Gertboard Blink\n") ;

int pin = 0;

int pin1 = 0;

wiringPiSetup () ;

pinMode (LED1, INPUT) ;

pinMode (LED, OUTPUT) ;

for (;;)

{

digitalWrite (LED, 1) ; // On

delay (500) ; // mS

digitalWrite (LED, 0) ; // Off

//pin = digitalRead (LED ) ; // Off

// pin1 = digitalRead (LED1 ) ; // Off

printf ("Pin - Obstacle OUTPUT: %d \n", pin) ;

printf ("Pin - Alchohol OUTPUT: %d \n", pin1) ;

//if(pin == 1 && pin1 == 1)

if(pin == 1 )

{

digitalWrite (LED3, 1) ; // On

delay (1500) ; // mS

printf ("\n\n <===== SWITCHING ON MOTORCYCLE ====>\n\n") ;

}

else

{

digitalWrite (LED3, 0) ; // Off

delay (1500) ; // mS

printf ("\n\n <===== WARNING:::: SWITCHING OFF MOTORCYCLE ====>\n\n") ;

}

delay (500) ;

}

return 0 ;

}

<----------blink.c---------->

#include <stdio.h>

#include <wiringPi.h>

// LED Pin - wiringPi pin 0 is BCM\_GPIO 17.

#define LED 0 // GPIO 17

#define LED1 1 // GPIO 18

#define LED3 3 // GPIO 22

int main (void)

{

printf ("Raspberry Pi - Gertboard Blink\n") ;

int pin = 0;

int pin1 = 0;

wiringPiSetup () ;

pinMode (LED1, OUTPUT) ;

pinMode (LED, INPUT) ;

for (;;)

{

digitalWrite (LED1, 1) ; // On

delay (500) ; // mS

digitalWrite (LED1, 0) ; // Off

pin = digitalRead (LED ) ; // Off

pin1 = digitalRead (LED1 ) ; // Off

printf ("Pin - Obstacle OUTPUT: %d \n", pin) ;

printf ("Pin - Alchohol OUTPUT: %d \n", pin1) ;

//if(pin == 1 && pin1 == 1)

if(pin == 1 )

{

digitalWrite (LED3, 1) ; // On

delay (1500) ; // mS

printf ("\n\n <===== SWITCHING ON MOTORCYCLE ====>\n\n") ;

}

else

{

digitalWrite (LED3, 0) ; // Off

delay (1500) ; // mS

printf ("\n\n <===== WARNING:::: SWITCHING OFF MOTORCYCLE ====>\n\n") ;

}

delay (500) ;

}

return 0 ;

}

<----------Bluetooth.c---------->

#include <stdio.h>

#include <stdlib.h>

#include <netdb.h>

#include <netinet/in.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <bluetooth/bluetooth.h>

#include <bluetooth/hci.h>

#include <bluetooth/hci\_lib.h>

#include <bluetooth/rfcomm.h>

int sendData(const char \*addr, char \* name);

int sendEth(char \* buffer);

unsigned int pedId = 0;

//char dest[18] = "00:1B:10:00:2A:EC";

char dest[18] = "34:BB:26:8A:9D:01";

char buf[90];

void blue\_scan();

struct location{

char mac[19];

char name[50];

unsigned int floor;

unsigned int room;

unsigned int devid;

};

static struct hostent \* server;

static struct sockaddr\_in serveraddr;

int main(int argc, char\* argv[] )

{

pthread\_t blue;

bzero((char \*) &serveraddr, sizeof(serveraddr));

if(argc < 2)

{

fprintf(stderr,"usage %s hostname port\n", argv[0]);

return 0;

}

server = gethostbyname(argv[1]);

if (server == NULL) {

fprintf(stderr,"ERROR, no such host\n");

exit(0);

}

bcopy( server->h\_addr, &serveraddr.sin\_addr.s\_addr, server->h\_length);

// pthread\_mutex\_init(&mutex,NULL);

printf("\nCreating Theads ...\n\n");

pthread\_create( &blue, NULL, (void\*) blue\_scan, NULL);

while(1);

printf(" end of thread \n");

}

//int main(int argc, char \*\*argv)

void blue\_scan()

{

inquiry\_info \*ii = NULL;

int max\_rsp, num\_rsp;

int dev\_id, sock, len, flags;

int i;

char addr[19] = { 0 };

char name[248] = { 0 };

struct location loc;

memset(&loc, 0, sizeof(struct location));

printf("Enter Floor Num: ");

scanf("%u", &loc.floor);

printf("Enter Room Num: ");

scanf("%u", &loc.room);

while(1)

{

printf("\n<----- Starting New SCAN ---->\n\n");

dev\_id = hci\_get\_route(NULL);

sock = hci\_open\_dev( dev\_id );

if (dev\_id < 0 || sock < 0) {

perror("opening socket");

//exit(1);

}

len = 8;

max\_rsp = 255;

flags = IREQ\_CACHE\_FLUSH;

ii = (inquiry\_info\*)malloc(max\_rsp \* sizeof(inquiry\_info));

num\_rsp = hci\_inquiry(dev\_id, len, max\_rsp, NULL, &ii, flags);

if( num\_rsp < 0 ) perror("hci\_inquiry");

for (i = 0; i < num\_rsp; i++)

{

//ba2str(&(ii+i)->bdaddr, addr);

ba2str(&(ii+i)->bdaddr,addr);

memset(name, 0, sizeof(name));

if (hci\_read\_remote\_name(sock, &(ii+i)->bdaddr, sizeof(name),

name, 0) < 0)

strcpy(name, "[unknown]");

printf("Detected ::: %s %s\n", addr, name);

memset(buf, 0, sizeof(buf));

memcpy(&loc.mac, addr, 19);

memcpy(&loc.name, name, 50);

memcpy(buf, &loc, 90);

// sendBD( buff)

sendEth(buf);

}

free( ii );

close( sock );

printf("End of Scan ....\n\n\n");

sleep(30);

} // end of while

return ;

}

int sendBD(char \* buffer)

{

struct sockaddr\_rc addr = { 0 };

int s, status;

char dest[18] = "00:1B:10:00:2A:EC";

struct location loc;

printf("sending data to main ...\n");

memcpy( &loc, buffer, 90);

//char dest[18] = "00:15:83:0C:BF:EB";

// allocate a socket

s = socket(AF\_BLUETOOTH, SOCK\_STREAM, BTPROTO\_RFCOMM);

// set the connection parameters (who to connect to)

addr.rc\_family = AF\_BLUETOOTH;

addr.rc\_channel = (uint8\_t) 1;

str2ba( dest, &addr.rc\_bdaddr );

// connect to server

status = connect(s, (struct sockaddr \*)&addr, sizeof(addr));

printf("Connect status: %d\n", status);

// send a message

if( status == 0 )

{

status = write(s, buffer, 90);

}

printf("Send status: %d\n", status);

if( status < 0 ) perror("uh oh");

sleep(3);

close(s);

return 0;

}

int sendEth(char \* buffer)

{

int n = 0;

int byte\_count;

// struct sockaddr\_in serveraddr;

char \*servername;

char buf[256];

socklen\_t addr\_size;

int sockfd;

struct location loc;

printf("Sending data to Main ...\n");

memcpy( &loc, buffer, 90);

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

bzero(&serveraddr,sizeof(serveraddr));

serveraddr.sin\_family=AF\_INET;

serveraddr.sin\_port=htons(5001);

//bcopy((char\*) server->h\_addr, (char\*)&serveraddr.sin\_addr.s\_addr, server->h\_length );

// serveraddr.sin\_addr.s\_addr = inet\_addr("192.168.1.30");

serveraddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr\_size=sizeof(serveraddr);

if(connect(sockfd,(struct sockaddr \*)&serveraddr,addr\_size)==-1)

{

perror("connect");

return 1;

}

n = send(sockfd , buffer, 90, 0);

if( n < 0)

{

printf("Send failed");

return 1;

}

close(sockfd);

}

* 1. **Information about the implementation of Modules**

Helmet mounted Alcohol Detection sensor

The helmet will contain an ARM-7 controller like Raspberry Pi board. The microcontroller will be connected with alcohol detection sensor using GPIO pins. The bike will have another microcontroller. The bike microcontroller is interfaced with engine start stop function. The helmet mounted microcontroller is connected with bike using either Wi-Fi or Bluetooth devices. When bike is switched on the bike microcontroller waits for the signal from helmet mounted microcontroller. If bike microcontroller receives alcohol signal from helmet controller then it will switch off the bike. In case bike controller receives non-alcoholic signal from helmet controller then it will enable the engine.

Eyes blink sensor or obstacle sensor

Many government authorities enacted law to make wearing helmet compulsory but implementing same is proving to be difficult as it need large pool of police to patrol the streets. Hence an attempt is made to make such implementation automatic. The helmet mounted controller is connected to eye-blink sensor or obstacle sensor to detect if person is wearing the helmet or not prior to starting the bike. The controller mounted on bike will wait for the signal from helmet mounted controller before starting the engine. In case the person is not wearing the bike then helmet mounted controller will send negative signal to bike controller then the controller will not start the bike. In case the helmet controller sends positive signal indicating the person is wearing the helmet then the bike controller will start the engine. In this way we can make wearing the helmet compulsory.

GSM module for sending SMS in case of accident

The bike controller will regularly exchange messages with helmet controller. In normal circumstance the exchanges will take place without any interruption. Also in case the person switches off the bike then both bike controller and helmet controller will properly terminate the connection. But in case of accident the eye sensor and alcohol sensor will lose connection with bike controller hence an alarm will be raised. This alarm will activate the GSM module which will send SMS to family and friends. Also it will send message to ambulance and police authorities regarding accident along with location of the accident. Also the SMS will contain information about the person like male/female, blood group and other ailments the person might be having. This will make hospital authorities to prepare for the treatment of the patience.

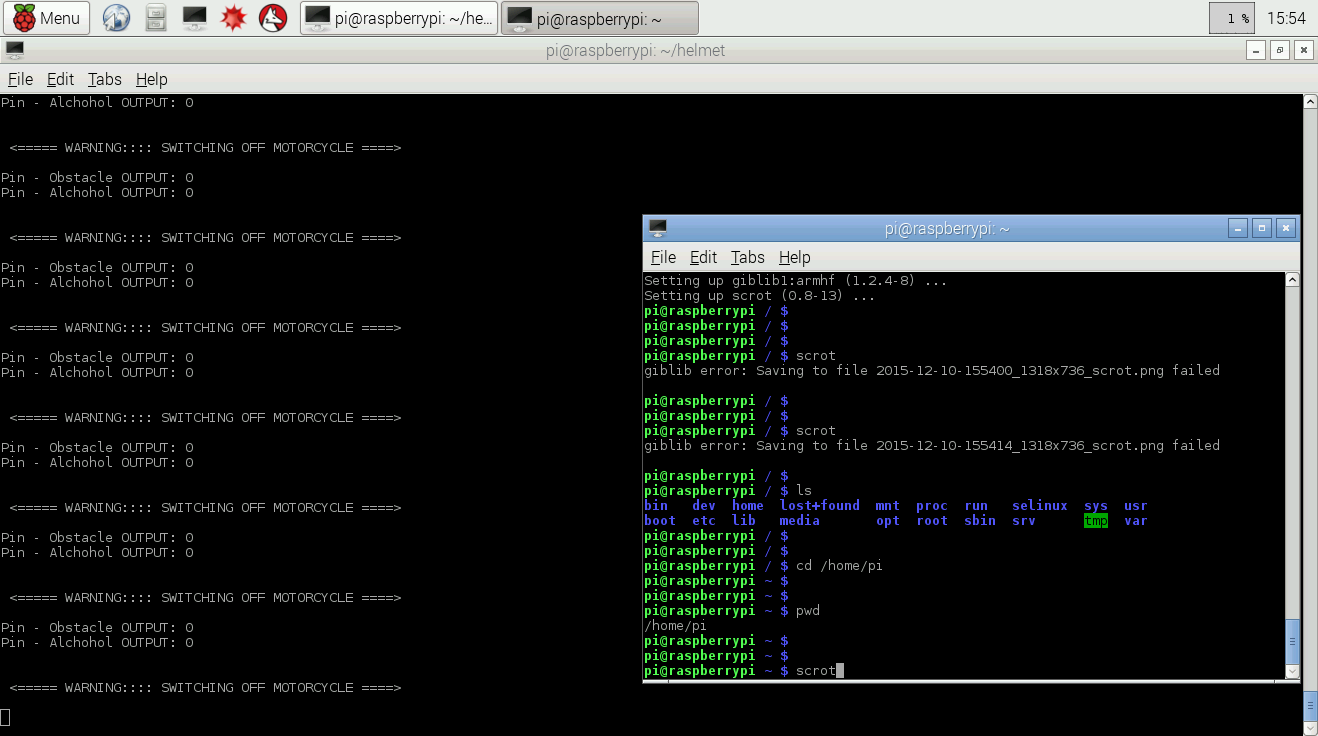
**7 TESTING**

**7.1 Results and Snapshots**

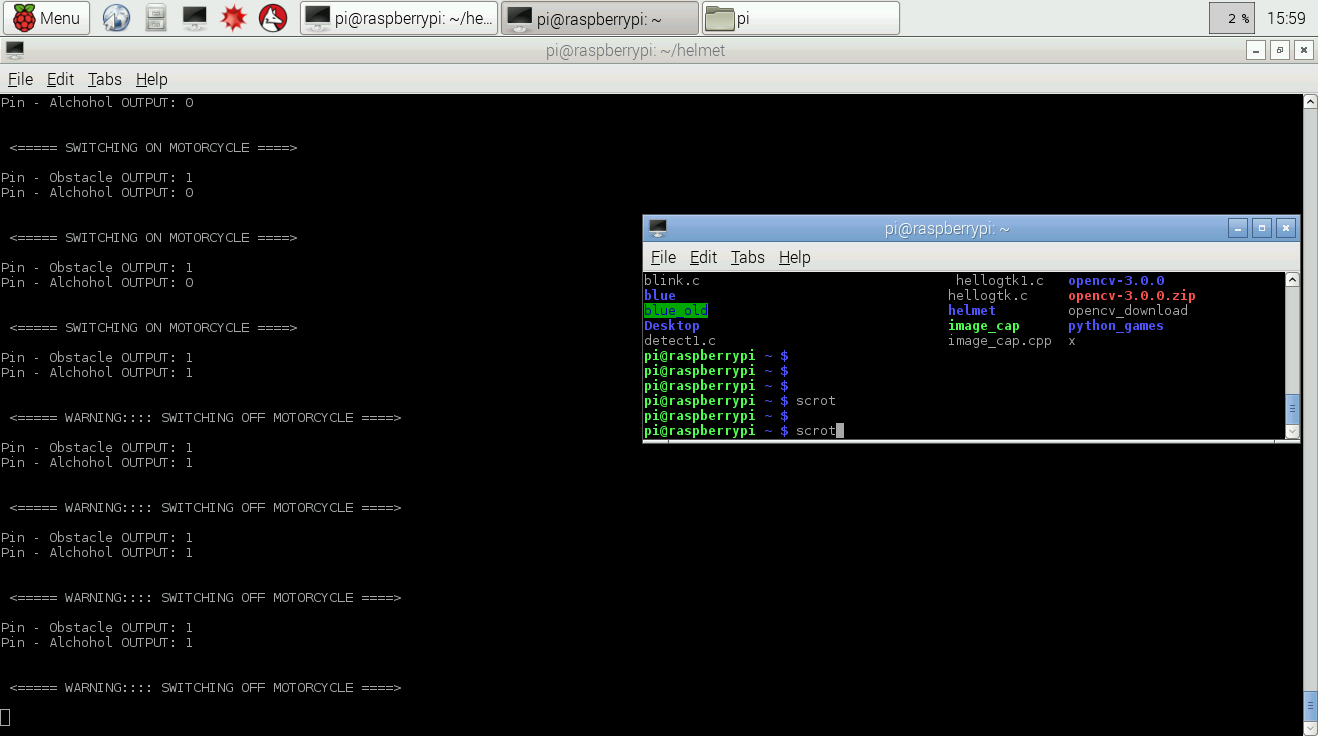
**Result**

Smart helmet for safe rider is designed with radio frequency link. , as user wear helmet a rf signal radiate from transmitter and these RF signal get sensed and synchronized with the help of address matching by the receiver section placed in the ignition switch of the bike and bike get started and bike stopped working as helmet keep out from head. This means bike work properly till helmet keep on head.

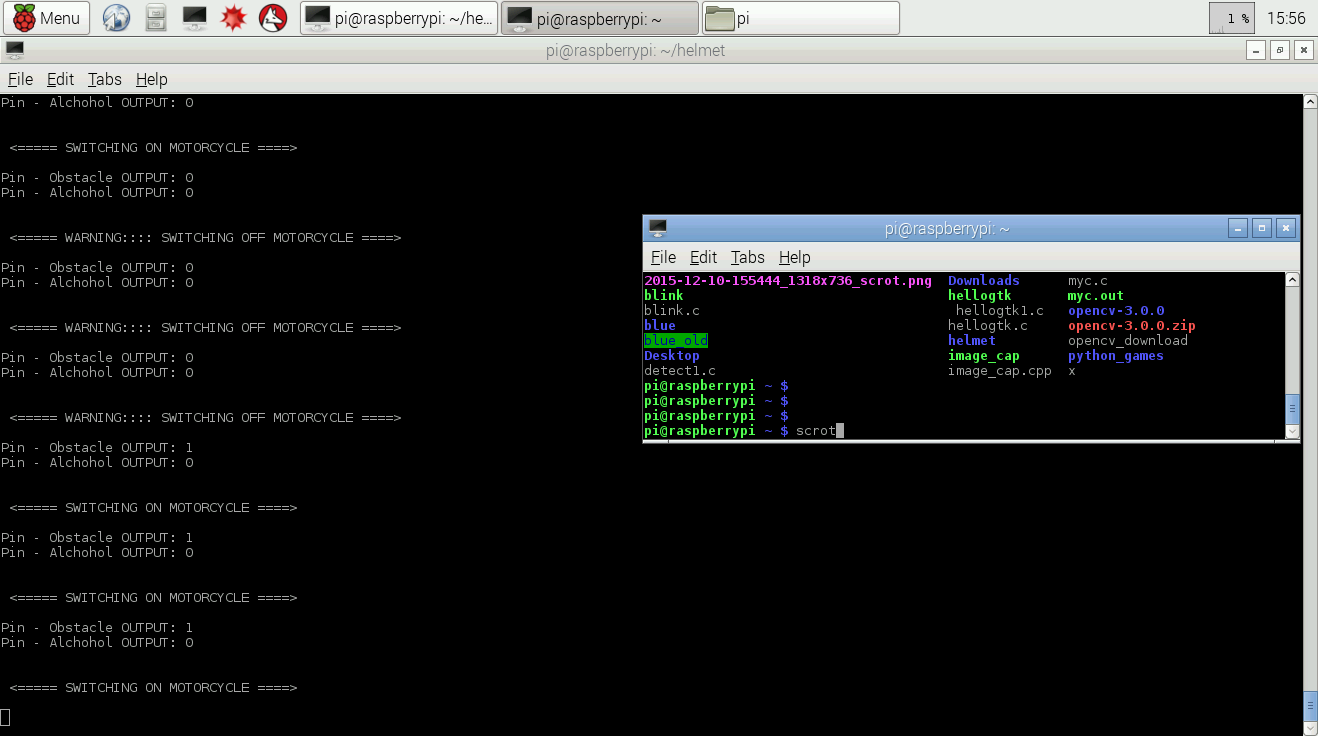
**Snapshots**

****Snap1

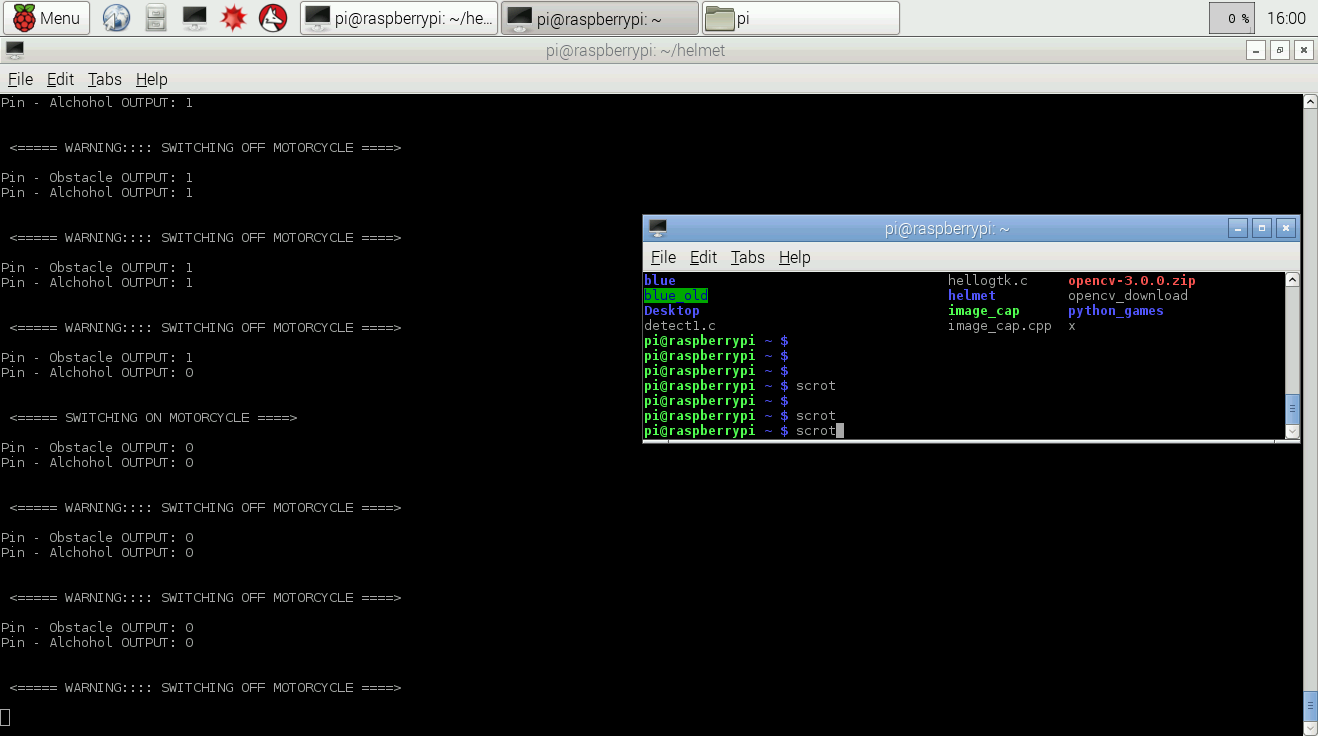
Snap2



Snap3



Snap4



**8 CONCLUSION & SCOPE FOR FUTURE WORK**

**Conclusion**

This system is very effective for the safety purpose of the user. User has to wear helmet to ride two wheeler vehicle and hence traffic rules will follow with this. This system is under pocket control ie. Ride two wheeler vehicle having safety in hand and in budget also. Easy functioning to operate this system. It provides a better security to the biker.

**Future Work**

In future if there is a large demand of this type of helmets we can manufacture the whole circuit in printed circuit board, so that circuit becomes smaller and can be easily fitted into helmet. The circuit can also be powered by solar energy so that it uses green energy and does no harm to environment .The flexible solar panels can fixed all along surface of helmet. This type of helmet technology can be implemented for the combat helmets used by the soldiers working under extreme temperatures.

**9 REFERENCES**

1. International Journal of Science and Research (IJSR) ISSN (Online): 2319‐7064 Volume 3 Issue 3, March 2014
2. International Journal Of Computer Science And Applications Vol. 6, No.2, Apr 2013 ISSN: 0974-1011 (Open Access)
3. Article from The Hindu [online] 2011 Feb. 10 Available from:URL:http://www.hindu.com/2011/02/10/stori es/2011021063740500.htm
4. International Journal of Scientific & Engineering Research Volume 2, Issue 12, December-2011 1 ISSN 2229-5518
5. Bishop, R (2002). The road ahead for intelligent vehicle sytem: what”s in store for riders? 8th Annual Minnesota Motorcycle safety conference
6. Sayeed and A. Perrig, “Secure Wireless Communications: Secret Keys through Multipath,” Proc. IEEEInt’l Conf. Acoustics, SpeechSignal Processing, pp. 3013-3016, Apr.2008
7. William R. Reagen, (1979) ―Auto theft detection system‖ US4177466 (US Patent) Computer”, May 2011
8. www.wikipeia.org/wiki/
9. www.electronicsforu.com
10. www.elecktor.com