CSD 404: INTERNET OF THINGS

VEHICLE COLLISION DETECTION AND AVOIDANCE PROJECT REPORT

ARAVINDH SATHYAMOORTHY 1310110073

BIKKI HARSHVARDHAN MANOJ 1310110104

TOSHAN MAJUMDAR 1310110343

**PROJECT GROUP MEMBERS:**

* **ARAVINDH SATHYAMOORTHY**
* **BIKKI HARSHVARDHAN MANOJ**
* **TOSHAN MAJUMDAR**

**Contents**

|  |  |  |
| --- | --- | --- |
| S NO | NAME OF TOPIC | PAGE NO |
| 1 | Introduction | 2-3 |
| (a) | Requirements of safe driving | 2 |
| (b) | Need for smart vehicles and smart cities | 2-3 |
| (c) | Advantages of making smart vehicles and smart cities | 3 |
| 2 | Our Project’s objectives | 3 |
| 3 | Sensors and other materials used | 4 |
| 4 | Sensor information | 4-5 |
| (a) | PIR sensor specifications | 4 |
| (b) | Ultrasonic sensor specifications | 4-5 |
| (c) | Vibration sensor specifications | 5 |
| 5 | Project Architecture | 6 |
| 6 | Complete Source Code | 8-23 |
| 7 | Snapshots of the output | 24-26 |
| 8 | Performance Evaluation | 26 |
| 9 | Limitations and Boundary Conditions | 27 |
| 10 | Conclusion | 27 |
| 11 | Bibiliography | 27 |
| 12 | Acknowledgements | 27 |

**Introduction**

**Requirements of safe driving**

Safe driving is very important in today's world. With the population of vehicles exploding, careless driving puts many lives at risk. With the introduction of smart cities, vehicles capable of preventing collisions becomes necessary. In the future, the data generated by the sensors attached in the vehicles will generate data which could later be used to visualize traffic pattern and help in optimizing them.

There are many ideas and laws that are created to ensure a safe and pleasurable driving experience. Some are avoiding aggressive drivers, alertness while driving, and vehicle following distance, vehicle speed, and special driving situations.

One of many factors in remaining safe on the roads is the ability to control your emotions and your stress. Some drivers do not know how to deal with stress. This emotion can hinder the ability to drive safely and ensure safety on the roads. If you encounter a vehicle that is driving recklessly, and constantly following too closely the best thing to do is to do to avoid a confrontation is to reduce speed and allow him to over take your vehicle. Do not provoke the driver just let them pass.

Accident are avoidable, all it takes is to be aware of your surroundings. Pay attention do what you are doing. Cell phones radios and newspapers are common distractions while in a motor vehicle. If you must use a mobile device ensure that you have a hands free accessory for it. If not pull over and make your call. Never try to understand or navigate a map while operating a vehicle. This can cause you to commit a traffic violation, or even injure a pedestrian.

There is a rule of thumb that can help keep a safe following distance while traveling the roads. It is called the “Two Second Rule”. Following a vehicle too closely is called "tailgating". Using the two-second rule to determine a safe following distance is beneficial. Select a fixed object on the road ahead such as a sign, tree or overpass. When the vehicle ahead of you passes the object, count "one-thousand-one, one-thousand-two". You should not reach the object before you count to one-thousand-two. If you do, you are following too closely. Most rear end collisions are caused by the vehicle in back following too closely.

**Need for smart vehicles and smart cities**

City populations are growing rapidly and our transport systems are feeling the pressure. Roads are increasingly gridlocked, buses are busier, and subway carriages are crammed to capacity during peak hours. With so many people tugging on the same resources, the need for smarter solutions has never been greater. Enabling people to get around efficiently isn’t as simple as just adding more roads or increasing public transport links. Technology has a big role to play, especially the [Internet of Things](https://inmotionventures.com/jargon-busting-future-transport/). Already we’re seeing cars get smarter with lane assist systems, blind spot monitoring, automatic parking and GPS that automatically reroutes to avoid congestion. These are all ‘building blocks’ which, when combined with vehicle-to-vehicle communication (V2V), will help improve and modernise cities. Autonomous cars and vehicles will be a vital source of information in the future. Interestingly, not much will have to change to accommodate these vehicles, as they will be intelligent enough to assimilate into smart city life. While they will certainly benefit from modernised transport networks, they are ultimately being designed to make informed decisions about any kind of environment or climate they find themselves in. In other words, like humans, except, it’s fair to say, far better. One of the key questions surrounding future connectivity is whether current mobile internet speeds and Wi-Fi are quick enough to handle so much data. But there’s another option. Li-Fi, which harnesses light for wireless communication, has ample potential. Cars will help to drive the smart city revolution forward and soon we could see the vehicles we drive – and those that drive us – seamlessly communicating with the world outside.

**Advantages of making smart vehicles and smart cities:**

It is a lucrative market since there are many vehicles today on the road and with the concept of "smart" city gaining popularity, vehicles have to be made smart as well. This technology costs very less since the sensors used are not expensive while also providing security and support to the users. It is beneficial for both manufacturers and consumers alike. There are numerous important research challenges for the smart city IoT applications:

* Overcoming traditional silo based organization of the cities, with each utility responsible for their own closed world.Although not technological this is one of the main barriers.
* Creating algorithms and schemes to describe information created by sensors in different applications to enable useful exchange of information between different city services
* Mechanisms for cost efficient deployment and even more important maintenance of such installations, including energy scavenging
* Ensuring reliable readings from a plethora of sensors and efficient calibration of a large number of sensors deployed everywhere from lampposts to waste bins
* Low energy protocols and algorithms
* Algorithms for analysis and processing of data acquired in the city and making “sense” out of it.
* IoT large scale deployment and integration.

**Our Project’s objectives**

The Vehicular Accident Detection System primarily detects various obstacles in the surroundings of the vehicle, processes it using raspberry pi, and displays meaningful information to the user. It particularly does the following:

* The ultrasonic sensor attached in the front part of the vehicle calculates the relative velocity of the incoming obstacle/vehicle at regular time intervals and alerts the user if the obstacle is too close.
* The PIR sensor attached on the rear side of the vehicle alerts the user about the obstacles while the vehicle is in reverse gear.
* The IR sensor detects the consciousness of the driver and buzzer goes off if he is unconscious. If the driver remains unconscious for some amount of time, the motor is automatically shut down.
* Additionally, the vibration sensor detects the condition of the road and helps the user to make smart decisions on which route to take while driving. All the data from the sensor are analyzed using Android application which alerts the user for any accidents.

**Sensors and other materials used**

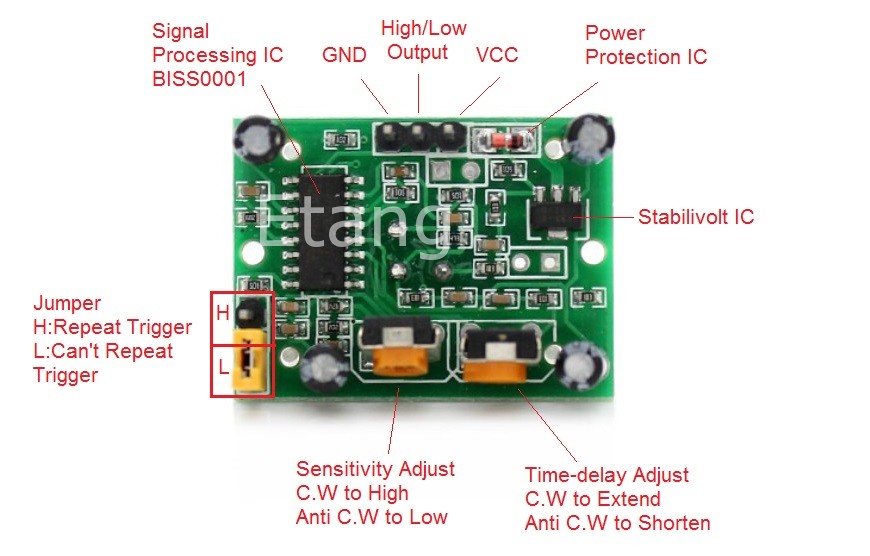
The following materials were required to complete the project:

* Raspberry Pi 3 model B+
* Ultrasonic sensor (HC-SR04)
* IR sensor (FC-31)
* PIR sensor (HC-SR501)
* Vibration sensor (SW-420)
* Breadboard and jumper cables

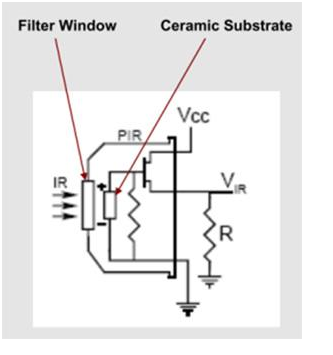
**Sensor Information**

The refresh rate (time interval at which sensors sense data) for different sensors used in the project are as follows:

IR sensor: 5000 ms

Ultrasonic sensor: 1000 ms

PIR sensor: 5000 ms

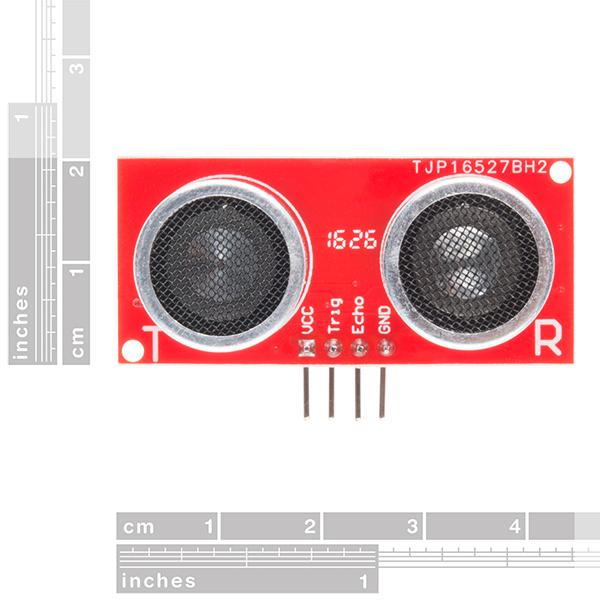


PIR specifications:

* PIR sensitivity range: 3-7 meters,
* PIR time delay range: 5 secs - 300 sec,
* Dimensions: 28 X 38 mm,
* Supply current: DC5V - 20V,
* Operation temperature: -15-degree C - 70-degree C

PIR SENSOR MODULE

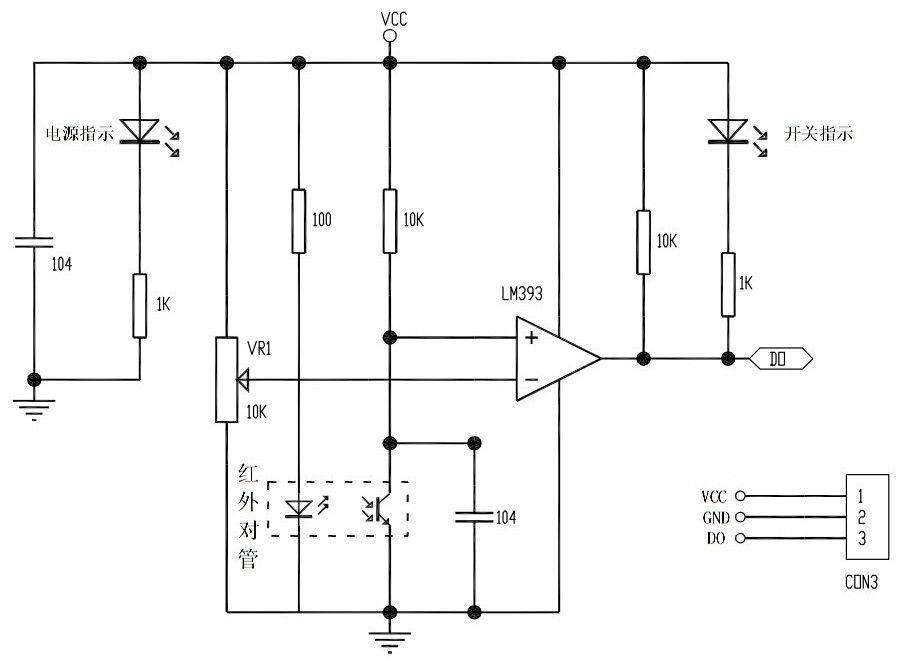
PIR SENSOR INTERNAL CIRCUIT



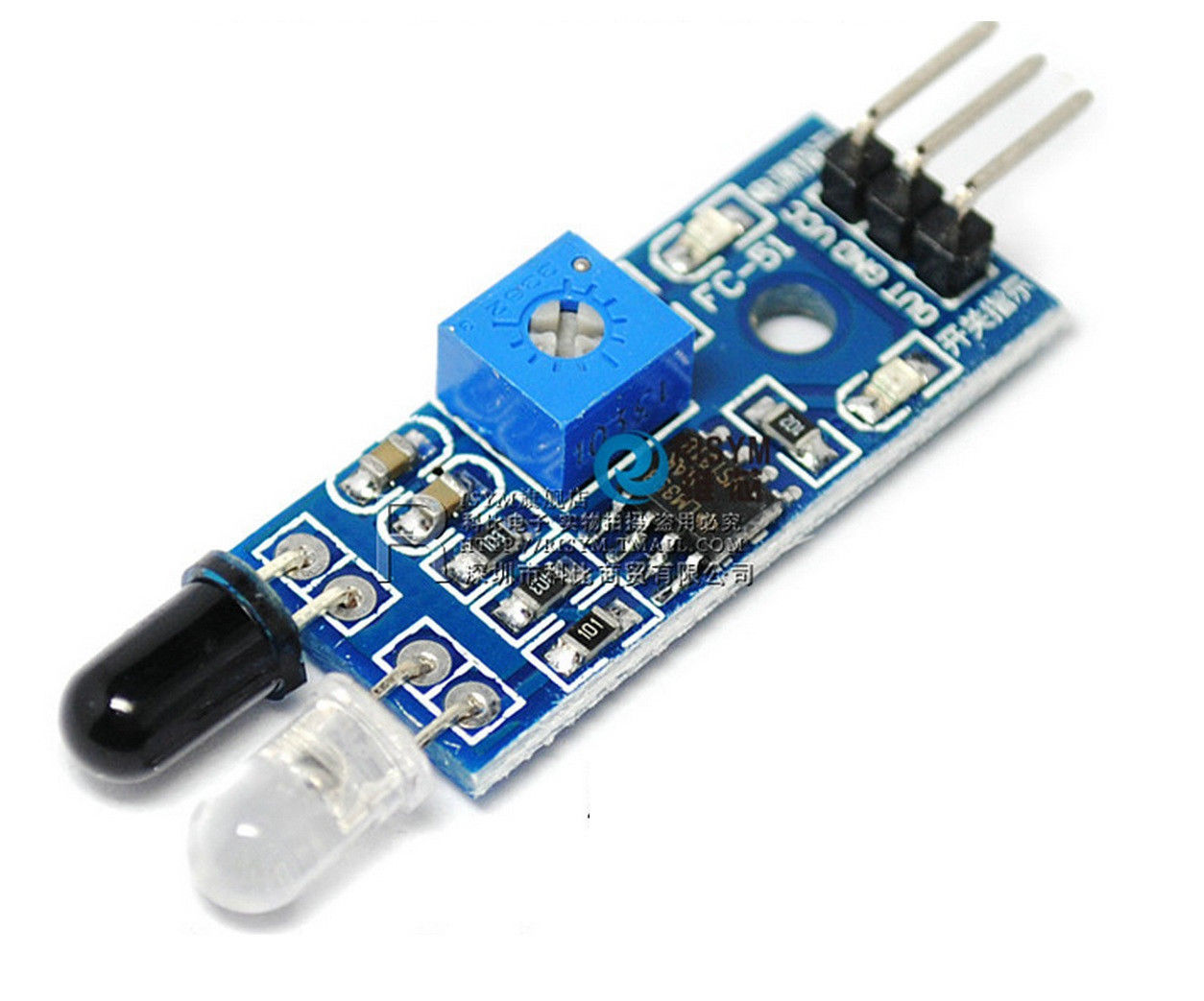
Ultrasonic distance sensor specifications:

* Power supply: 5V Dc
* quiescent current: <2mA
* effectual angle: <15 degree
* ranging distance: 2cm-500cm
* resolution: 0.3 cm

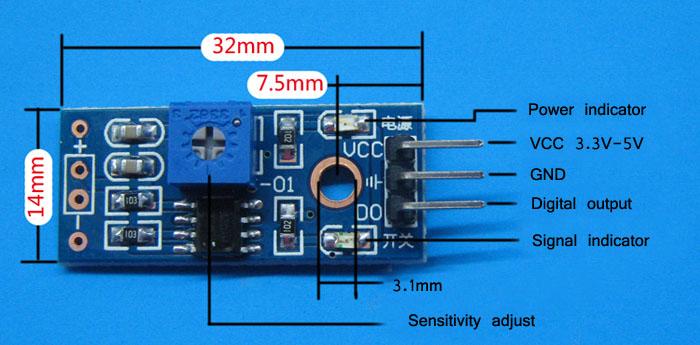
ULTRASONIC DISTANCE SENSOR MODULE

Infrared sensor specifications:

* Detection angle: 35 degrees,
* Operating Voltage: 3 V - 6.0 V,
* Detection range: 2 cm - 30 cm,
* Overall Dimension: 4.5 cm - 5.5 cm



Infrared sensor internal circuit diagram

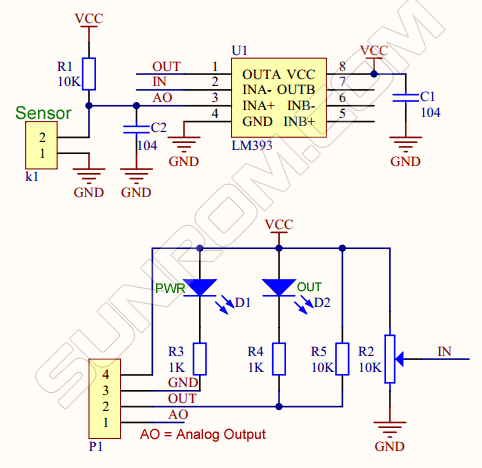


IR SENSOR MODULE

Vibration sensor specifications:

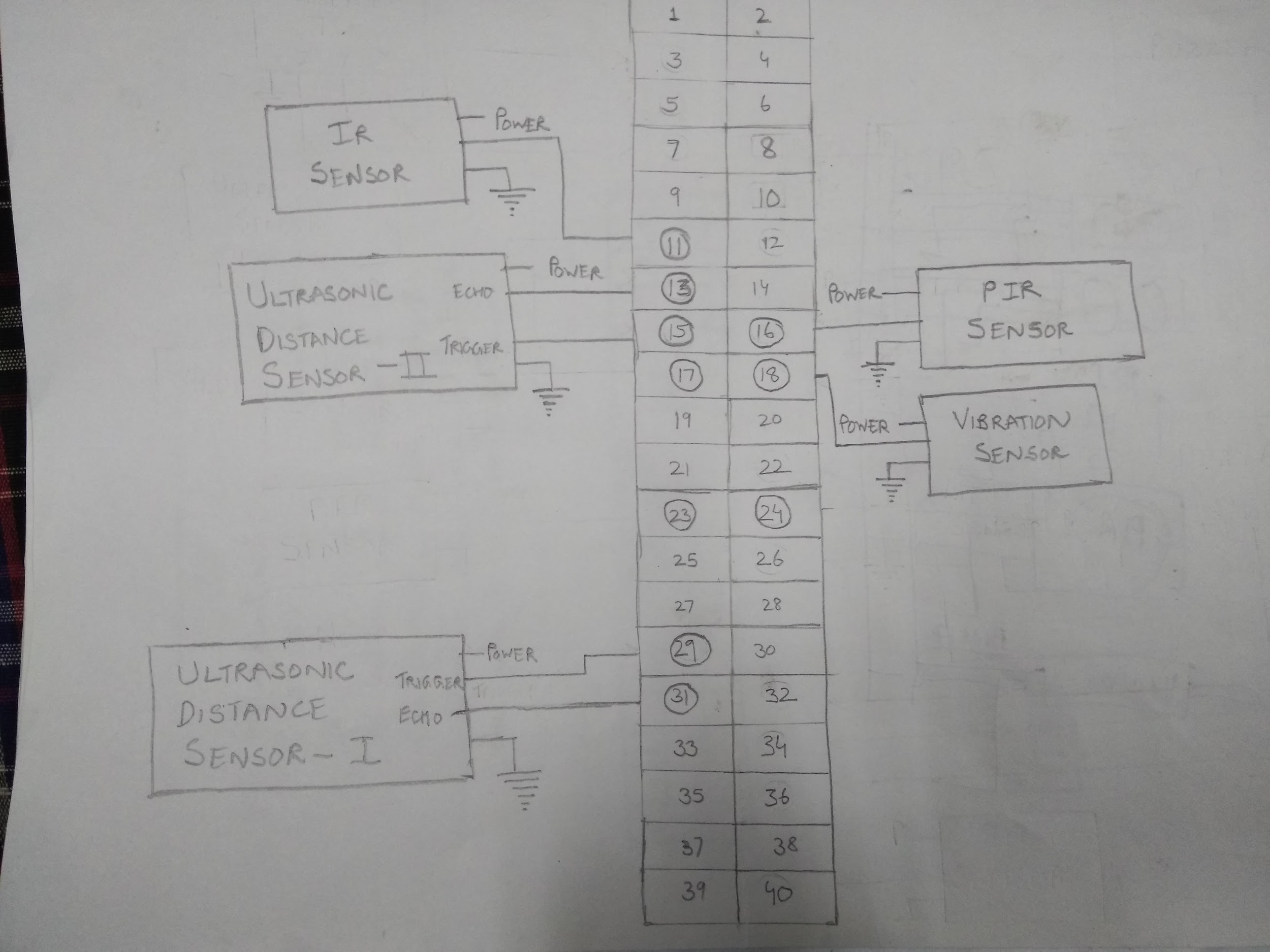
* Operating voltage: 3.3 V - 5 V
* Dimensions: 3.2 X 1.4

VIBRATION SENSOR INTERNAL CIRCUIT DIAGRAM

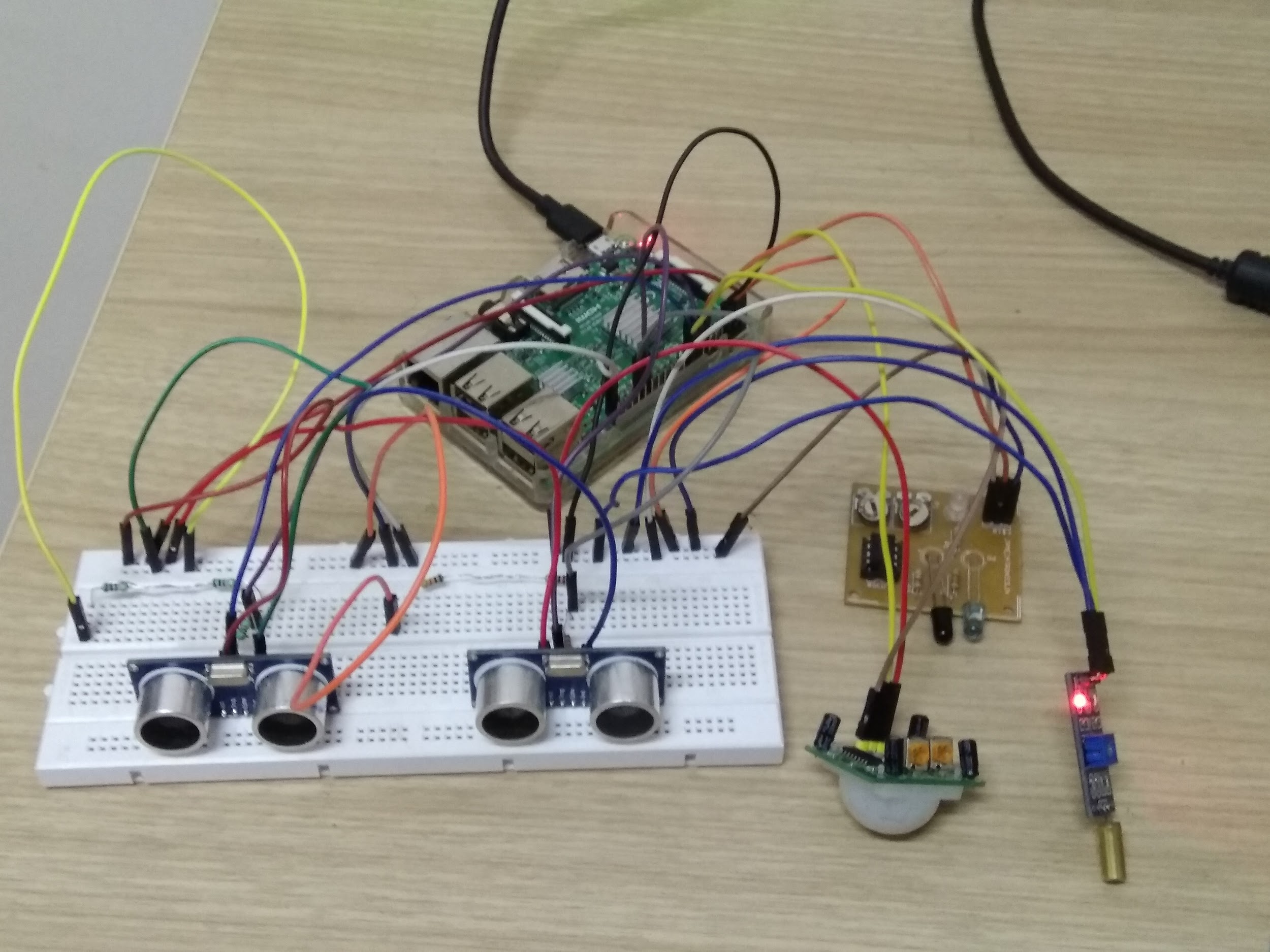


VIBRATION SENSOR MODULE

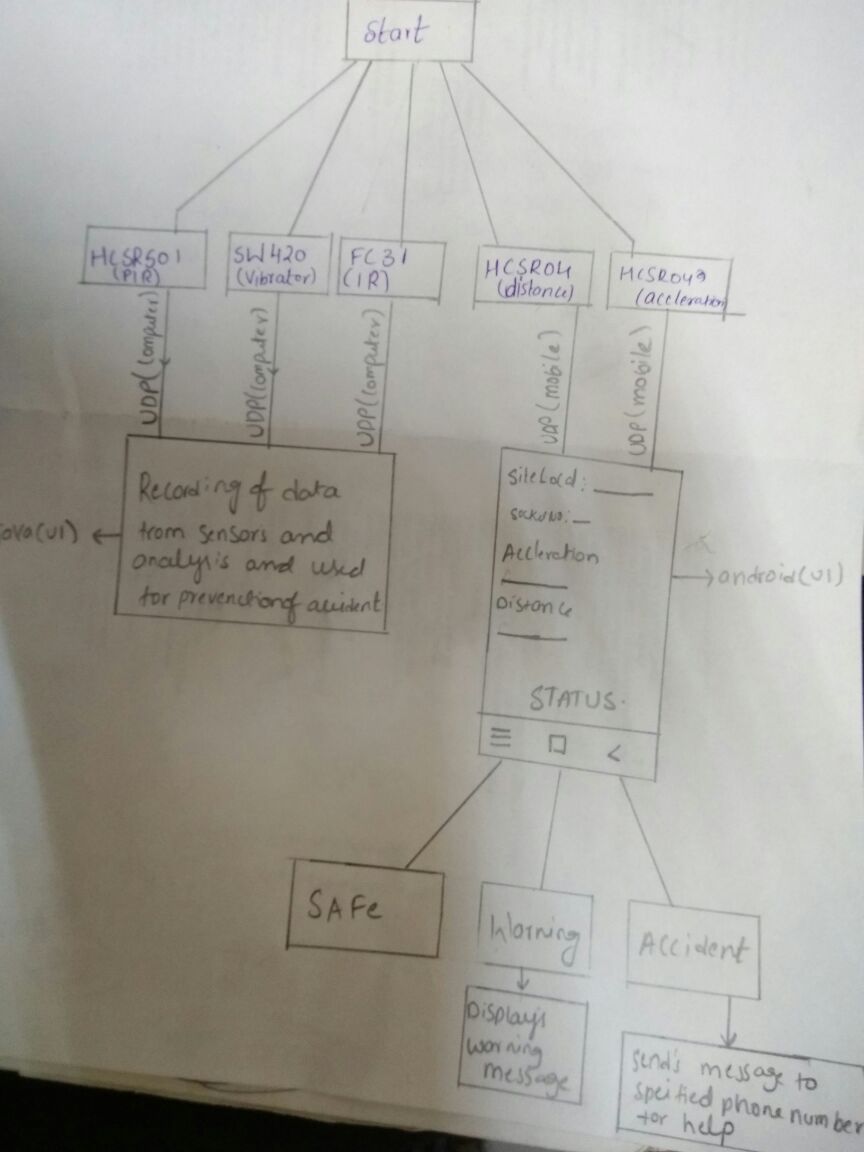
**Project Architecture and circuit diagram**



WIRING CONNECTIONS OF RASPBERRY PI AND VARIOUS SENSORS



VEHICLE COLLOSION DETECTION SYSTEM

****

SYSTEM ARCHITECTURE AND SCHEMATIC FLOW CHART

**Complete Source Code**

**MAIN DRIVER CLASS:**

**package pirsensor;**

**import java.io.IOException;**

**import java.util.logging.Level;**

**import java.util.logging.Logger;**

**import javax.microedition.io.Connector;**

**import javax.microedition.io.Datagram;**

**import javax.microedition.io.DatagramConnection;**

**import javax.microedition.midlet.MIDlet;**

**import javax.microedition.midlet.MIDletStateChangeException;**

**import jdk.dio.gpio.PinEvent;**

**import jdk.dio.gpio.PinListener;**

**public class GPIOMotion extends MIDlet implements PinListener**

**{**

**HCSR501Device pir;**

**HCSR502Device vib;**

**ir\_code ir;**

**ultra ultra\_1;**

**accler ultra\_2;**

**private static final int PIR\_PIN = 23,VIB\_PIN = 24,IR\_PIN= 17,DIST\_TRIG = 22,DIST\_ECHO = 27,ACC\_TRIG = 5,ACC\_ECHO = 6;**

**private double pir\_value=0,vib\_value=0,ir\_value=0;**

**private volatile boolean shouldRun = true,Run = true;**

**private ULTRA\_1\_SENSOR sensorsTask;**

**private ULTRA\_2\_SENSOR run;**

**@Override**

**public void startApp()throws MIDletStateChangeException**

**{**

**pir= new HCSR501Device(PIR\_PIN);**

**pir.setListener(new PIR\_SENSOR() );**

**vib = new HCSR502Device(VIB\_PIN);**

**vib.setListener(new VIB\_SENSOR() );**

**ir = new ir\_code(IR\_PIN);**

**ir.setListener(new ir\_SENSOR() );**

**ultra\_1 = new ultra(DIST\_TRIG,DIST\_ECHO);**

**sensorsTask=new ULTRA\_1\_SENSOR();**

**sensorsTask.start();**

**ultra\_2 = new accler(ACC\_TRIG,ACC\_ECHO);**

**run=new ULTRA\_2\_SENSOR();**

**run.start();**

**}**

**@Override**

**public void destroyApp(boolean unconditional)**

**{**

**try**

**{**

**pir.close();**

**vib.close();**

**ir.close();**

**ultra\_1.close();**

**ultra\_2.close();**

**Run=false;**

**shouldRun=false;**

**}**

**catch (Exception ex) {Logger.getLogger(GPIOMotion.class.getName()).log(Level.SEVERE, null, ex);}**

**}**

**@Override**

**public void valueChanged(PinEvent event)**

**{throw new UnsupportedOperationException("Not supported yet.");}**

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

**//class for handling operations of the PIR sensor**

**class PIR\_SENSOR implements PinListener**

**{**

**public void valueChanged(PinEvent event)**

**{**

**if (event.getValue())**

**{**

**System.out.println("Motion detected");**

**pir\_value=1;**

**}**

**else**

**{pir\_value=0;}**

**String q = "PIR "+String.valueOf(pir\_value);**

**//sending data given by PIR senor to host computer**

**try**

**{**

**DatagramConnection dgc = (DatagramConnection)**

**Connector.open("datagram://10.6.13.20:9876");**

**try**

**{**

**byte[] payload = q.getBytes();**

**Datagram datagram = dgc.newDatagram(payload, payload.length);**

**dgc.send(datagram);**

**}**

**finally {dgc.close();}**

**}**

**catch (IOException x)**

**{x.printStackTrace();}**

**}**

**}**

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

**//class for handling the operations of ultrasonic distance sensor 1**

**class ULTRA\_1\_SENSOR extends Thread {**

**private double distance=0.0;**

**@Override**

**public void run(){**

**while(shouldRun){**

**distance = ultra\_1.pulse();**

**if(distance >0){**

**System.out.println("Object at " + distance + " cm.");**

**String a="D"+String.valueOf(distance);**

**//sending data given by ultrasonic distance sensor 1 to host computer**

**try{**

**DatagramConnection dgc = (DatagramConnection)**

**Connector.open("datagram://10.6.13.20:9876");**

**try {**

**byte[] payload = a.getBytes();**

**Datagram datagram = dgc.newDatagram(payload, payload.length);**

**dgc.send(datagram);**

**} finally {**

**dgc.close();**

**}**

**} catch (IOException x) {**

**x.printStackTrace();**

**}**

**try {**

**Thread.sleep(1000);**

**} catch (InterruptedException ex) {**

**Logger.getLogger(GPIOMotion.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**}**

**}**

**}**

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

**//class for handling the operations of ultrasonic distance sensor 2**

**class ULTRA\_2\_SENSOR extends Thread**

**{**

**private double distance1=0.0;**

**private double distance2=0.0;**

**private double acc=0.0;**

**@Override**

**public void run(){**

**while(Run){**

**try {**

**distance1 = ultra\_2.dis();**

**Thread.sleep(2000);**

**distance2 = ultra\_2.dis();**

**acc=distance2-distance1/2;**

**if(acc >0){**

**System.out.println("Object at " + acc + " cm/s.");**

**String w="A"+String.valueOf(acc);**

**//sending data given by ultrasonic distance sensor 2 to host computer**

**try{**

**DatagramConnection dgc = (DatagramConnection)**

**Connector.open("datagram://10.6.13.20:9876");**

**try {**

**byte[] payload =w.getBytes();**

**Datagram datagram = dgc.newDatagram(payload, payload.length);**

**dgc.send(datagram);**

**} finally {**

**dgc.close();**

**}**

**} catch (IOException x) {**

**x.printStackTrace();**

**}**

**try {**

**Thread.sleep(1000);**

**} catch (InterruptedException ex) {**

**Logger.getLogger(GPIOMotion.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**} } catch (InterruptedException ex) {**

**Logger.getLogger(GPIOMotion.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**}**

**}**

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

**//class for handling operations of the IR sensor**

**class ir\_SENSOR implements PinListener**

**{**

**public void valueChanged(PinEvent event) {**

**if (event.getValue())**

**{**

**System.out.println("ir detected");**

**ir\_value=1;**

**}**

**else if(event.getValue()==false)**

**{ir\_value=0;}**

**String s=" IR "+String.valueOf(ir\_value);**

**//sending data given by IR sensor to host computer**

**try**

**{**

**DatagramConnection dgc = (DatagramConnection)**

**Connector.open("datagram://10.6.13.20:9876");**

**try**

**{**

**byte[] payload = s.getBytes();**

**Datagram datagram = dgc.newDatagram(payload, payload.length);**

**dgc.send(datagram);**

**}**

**finally {dgc.close();}**

**}**

**catch (IOException x)**

**{x.printStackTrace();}**

**}**

**}**

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

**//class for handling operations of the vibration sensor**

**class VIB\_SENSOR implements PinListener{**

**public void valueChanged(PinEvent event) {**

**if (event.getValue())**

**{**

**System.out.println("Rough road detected");**

**vib\_value=1;**

**}**

**else**

**{vib\_value=0;}**

**String e="VIB "+String.valueOf(vib\_value);**

**//sending data given by vibration to host computer**

**try**

**{**

**DatagramConnection dgc = (DatagramConnection)**

**Connector.open("datagram://10.6.13.20:9876");**

**try**

**{**

**byte[] payload = e.getBytes();**

**Datagram datagram = dgc.newDatagram(payload, payload.length);**

**dgc.send(datagram);**

**}**

**finally {dgc.close();}**

**}**

**catch (IOException x)**

**{x.printStackTrace();}**

**}**

**}**

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

**}**

**PIR SENSOR OBJECT CLASS:**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package pirsensor;**

**import java.io.IOException;**

**import java.util.logging.Level;**

**import java.util.logging.Logger;**

**import jdk.dio.DeviceManager;**

**import jdk.dio.gpio.GPIOPin;**

**import jdk.dio.gpio.GPIOPinConfig;**

**import jdk.dio.gpio.PinListener;**

**public class HCSR501Device**

**{**

**private GPIOPin pin=null;**

**public HCSR501Device(int pinGPIO)**

**{**

**try {**

**pin = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, pinGPIO, GPIOPinConfig.DIR\_INPUT\_ONLY, GPIOPinConfig.MODE\_INPUT\_PULL\_DOWN,**

**GPIOPinConfig.TRIGGER\_RISING\_EDGE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(HCSR501Device.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public void setListener(PinListener pirListener)**

**{**

**if (pin!=null)**

**try {**

**pin.setInputListener(pirListener);**

**} catch (IOException ex) {**

**Logger.getLogger(HCSR501Device.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public void close() throws Exception {**

**if (pin!=null){**

**pin.setInputListener(null);**

**pin.close();**

**}**

**}**

**}**

**VIBRATION SENSOR OBJECT CLASS:**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package pirsensor;**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**import java.io.IOException;**

**import java.util.logging.Level;**

**import java.util.logging.Logger;**

**import jdk.dio.DeviceManager;**

**import jdk.dio.gpio.GPIOPin;**

**import jdk.dio.gpio.GPIOPinConfig;**

**import jdk.dio.gpio.PinListener;**

**public class HCSR502Device**

**{**

**private GPIOPin pin=null;**

**public HCSR502Device(int pinGPIO)**

**{**

**try {**

**pin = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, pinGPIO, GPIOPinConfig.DIR\_INPUT\_ONLY, GPIOPinConfig.MODE\_INPUT\_PULL\_DOWN,**

**GPIOPinConfig.TRIGGER\_RISING\_EDGE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(HCSR501Device.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public void setListener(PinListener pirListener)**

**{**

**if (pin!=null)**

**try {**

**pin.setInputListener(pirListener);**

**} catch (IOException ex) {**

**Logger.getLogger(HCSR501Device.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public void close() throws Exception {**

**if (pin!=null){**

**pin.setInputListener(null);**

**pin.close();**

**}**

**}**

**}**

**ULTRASONIC SENSOR 2 OBJECT CLASS:**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package pirsensor;**

**/\*\***

**\***

**\* @author Toshan**

**\*/**

**import java.io.IOException;**

**import java.util.logging.Level;**

**import java.util.logging.Logger;**

**import jdk.dio.DeviceManager;**

**import jdk.dio.gpio.GPIOPin;**

**import jdk.dio.gpio.GPIOPinConfig;**

**public class accler**

**{**

**private final int PULSE = 10000; // #10 Âµs pulse = 10,000 ns**

**private final int SPEEDOFSOUND = 34029; // Speed of sound = 34029 cm/s**

**private GPIOPin trigger = null;**

**private GPIOPin echo = null;**

**public accler(int \_trigger, int \_echo)**

**{**

**try {**

**trigger = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, \_trigger, GPIOPinConfig.DIR\_OUTPUT\_ONLY, GPIOPinConfig.MODE\_OUTPUT\_PUSH\_PULL,GPIOPinConfig.TRIGGER\_NONE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(accler.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**try {**

**echo = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, \_echo, GPIOPinConfig.DIR\_INPUT\_ONLY, GPIOPinConfig.MODE\_INPUT\_PULL\_UP,GPIOPinConfig.TRIGGER\_NONE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(accler.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**try {**

**Thread.sleep(1000);**

**} catch (InterruptedException ex) {**

**Logger.getLogger(accler.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public double dis() {**

**long distance = 0;**

**try {**

**trigger.setValue(true); //Send a pulse trigger; must be 1 and 0 with a 10 Âµs wait**

**try {**

**Thread.sleep(0, PULSE);// wait 10 Âµs**

**} catch (InterruptedException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**trigger.setValue(false);**

**long starttime = System.nanoTime(); //ns**

**long stop = starttime;**

**long start = starttime;**

**//echo will go 0 to 1 and need to save time for that. 2 seconds difference**

**while ((!echo.getValue()) && (start < starttime + 1000000000L \* 2)) {**

**start = System.nanoTime();**

**}**

**while ((echo.getValue()) && (stop < starttime + 1000000000L \* 2)) {**

**stop = System.nanoTime();**

**}**

**long delta = (stop - start);**

**distance = delta \* SPEEDOFSOUND; // echo from 0 to 1 depending on object distance**

**} catch (IOException ex) {**

**Logger.getGlobal().log(Level.WARNING,ex.getMessage());**

**}**

**return distance / 2.0 / (1000000000L); // cm/s**

**}**

**public void close() {**

**if ((trigger!=null) && (echo!=null)){**

**try {**

**trigger.close();**

**} catch (IOException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**try {**

**echo.close();**

**} catch (IOException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**;**

**}**

**}**

**}**

**IR SENSOR OBJECT CLASS:**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package pirsensor;**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**import java.io.IOException;**

**import java.util.logging.Level;**

**import java.util.logging.Logger;**

**import jdk.dio.DeviceManager;**

**import jdk.dio.gpio.GPIOPin;**

**import jdk.dio.gpio.GPIOPinConfig;**

**import jdk.dio.gpio.PinListener;**

**public class ir\_code**

**{**

**private GPIOPin pin=null;**

**public ir\_code(int pinGPIO)**

**{**

**try {**

**pin = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, pinGPIO, GPIOPinConfig.DIR\_INPUT\_ONLY, GPIOPinConfig.MODE\_INPUT\_PULL\_DOWN,**

**GPIOPinConfig.TRIGGER\_RISING\_EDGE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(HCSR501Device.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public void setListener(PinListener pirListener)**

**{**

**if (pin!=null)**

**try {**

**pin.setInputListener(pirListener);**

**} catch (IOException ex) {**

**Logger.getLogger(HCSR501Device.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public void close() throws Exception {**

**if (pin!=null){**

**pin.setInputListener(null);**

**pin.close();**

**}**

**}**

**}**

**ULTRASONIC SENSOR 1 OBJECT CLASS:**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package pirsensor;**

**/\*\***

**\***

**\* @author Toshan**

**\*/**

**import java.io.IOException;**

**import java.util.logging.Level;**

**import java.util.logging.Logger;**

**import jdk.dio.DeviceManager;**

**import jdk.dio.gpio.GPIOPin;**

**import jdk.dio.gpio.GPIOPinConfig;**

**public class ultra**

**{**

**private final int PULSE = 10000; // #10 Âµs pulse = 10,000 ns**

**private final int SPEEDOFSOUND = 34029; // Speed of sound = 34029 cm/s**

**private GPIOPin trigger = null;**

**private GPIOPin echo = null;**

**public ultra(int \_trigger, int \_echo)**

**{**

**try {**

**trigger = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, \_trigger, GPIOPinConfig.DIR\_OUTPUT\_ONLY, GPIOPinConfig.MODE\_OUTPUT\_PUSH\_PULL,GPIOPinConfig.TRIGGER\_NONE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**try {**

**echo = (GPIOPin) DeviceManager.open(new GPIOPinConfig(0, \_echo, GPIOPinConfig.DIR\_INPUT\_ONLY, GPIOPinConfig.MODE\_INPUT\_PULL\_UP,GPIOPinConfig.TRIGGER\_NONE, false));**

**} catch (IOException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**try {**

**Thread.sleep(1000);**

**} catch (InterruptedException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**}**

**public double pulse() {**

**long distance = 0;**

**try {**

**trigger.setValue(true); //Send a pulse trigger; must be 1 and 0 with a 10 Âµs wait**

**try {**

**Thread.sleep(0, PULSE);// wait 10 Âµs**

**} catch (InterruptedException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**trigger.setValue(false);**

**long starttime = System.nanoTime(); //ns**

**long stop = starttime;**

**long start = starttime;**

**//echo will go 0 to 1 and need to save time for that. 2 seconds difference**

**while ((!echo.getValue()) && (start < starttime + 1000000000L \* 2)) {**

**start = System.nanoTime();**

**}**

**while ((echo.getValue()) && (stop < starttime + 1000000000L \* 2)) {**

**stop = System.nanoTime();**

**}**

**long delta = (stop - start);**

**distance = delta \* SPEEDOFSOUND; // echo from 0 to 1 depending on object distance**

**} catch (IOException ex) {**

**Logger.getGlobal().log(Level.WARNING,ex.getMessage());**

**}**

**return distance / 2.0 / (1000000000L); // cm/s**

**}**

**public void close() {**

**if ((trigger!=null) && (echo!=null)){**

**try {**

**trigger.close();**

**} catch (IOException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**try {**

**echo.close();**

**} catch (IOException ex) {**

**Logger.getLogger(ultra.class.getName()).log(Level.SEVERE, null, ex);**

**}**

**;**

**}**

**}**

**}**

**UDP DATAGRAM CODE RECEIVER CODE:**

/\*

 \* To change this license header, choose License Headers in Project Properties.

 \* To change this template file, choose Tools | Templates

 \* and open the template in the editor.

 \*/

package udp;

/\*\*

 \*

 \* @author Manoj

 \*/

import java.io.\*;

import java.net.\*;

class UDp

{

   public static void main(String args[]) throws Exception

      {

         DatagramSocket serverSocket = new DatagramSocket(9876);

            byte[] receiveData = new byte[13];

            while(true)

               {

                  DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

                  serverSocket.receive(receivePacket);

                  String sentence = new String( receivePacket.getData());

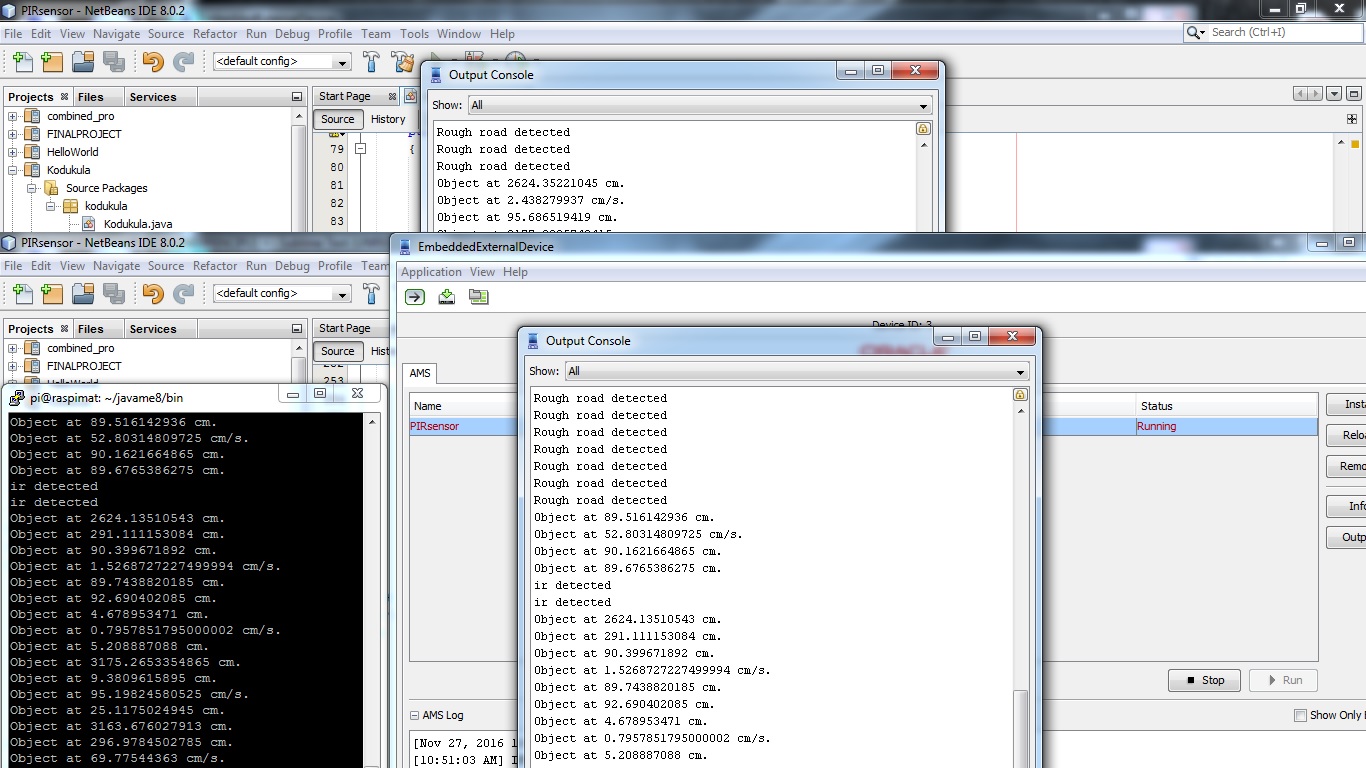
                  System.out.println(sentence);

               }

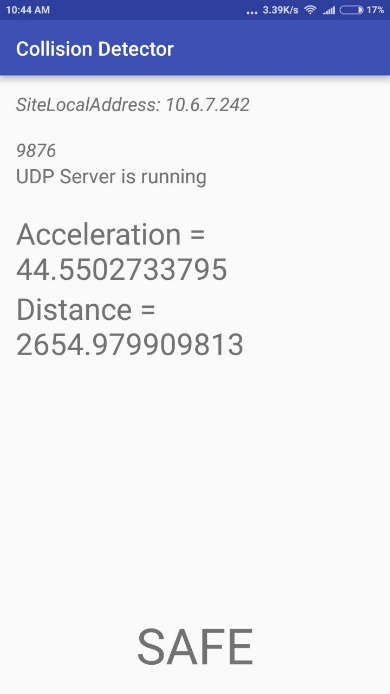
      }

}

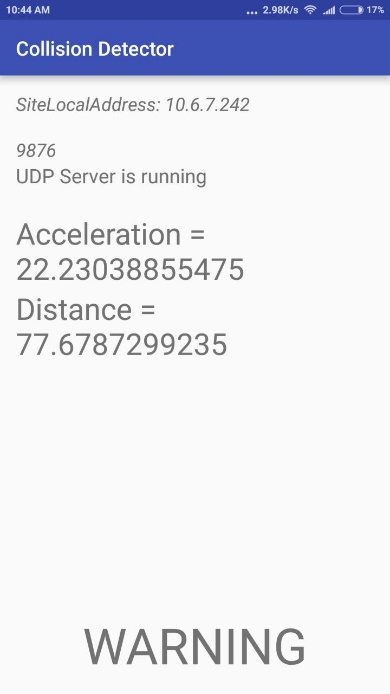
**Snapshots of the output**



HOST COMPUTER READINGS SNAPSHOT



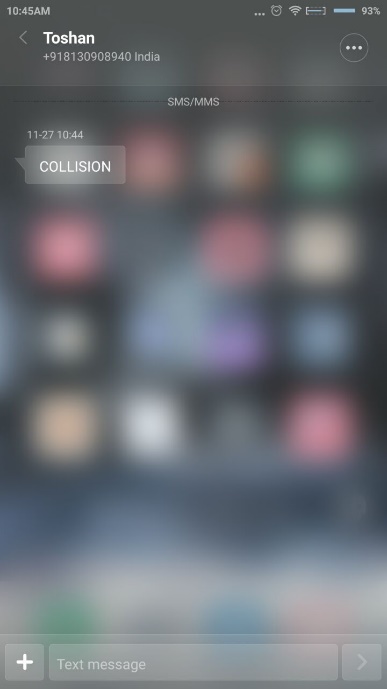
USER MOBILE UPDATES (SAFE SITUATION)



USER MOBILE UPDATE:(UNSAFE SITUATION)



SMS MESSAGE WHEN ACCIDENT COLLISION DETECTED



SMS MESSAGE RECEIVED BY EMERGENCY RESPONSE UNIT

**PERFORMANCE EVALUATION:**

**We intended to design an efficient vehicle collision detection and avoidance system which would alert the user of incoming obstacles on time and promptly alert emergency response services if collision has occurred. We had to keep in mind the sensor sensitivities as well as time delay in sending packets of information through the network as well as constant connectivity issues.**

**We were able to achieve an end to end system, however with basic UI implementation.**

**The final user android application is not remarkable.**

**LIMITATIONS AND BOUNDARY CONDIDTIONS**

Most of the inaccuracies in the project stems from the limitations of sensitivity of various sensors used. Some examples:

* The PIR sensor will raise alarm even if the vehicle is stuck in traffic due to the prolonged presence of vehicles near it.
* The project also does not take into consideration the fact that the accidents can happen from any direction since there is only one ultrasonic sensor used.
* Also, the project assumes that internet connectivity is always present since the lack of it can disrupt calculations.
* The indications of rough road conditions are not enough to prevent accidents; they are simply given to help the user choose other roads with good conditions.
* Due to less development on android application, the app closes unexpectedly for few cases.

**Conclusion**

The system which we have designed is capable of being deployed in current cars after improving the basic implementation by using accurate sensor alternatives. The future scope of the project is very high as due to the low cost of implementation of sensor designs as well as the requirement of security and safety systems in car vehicles nowadays.

**Bibiliography**

SETTING UP RASPBERRY PI AND JAVA ME ENVIRONMENT:

<http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/RaspberryPi_GPIO/RaspberryPi_GPIO.html>

EXAMPLE JAVA ME CODES:

<http://www.oracle.com/technetwork/articles/java/cruz-gpio-2295970.html>

INFORMATION ON SENSORS:

[www.alldatasheet.com](http://www.alldatasheet.com)

**Acknowledgements**

We would like to thank Dr. Debopam Acharya and Dr. Divya Lohani for giving us the opportunity to work on this project as a part of CSD404: Internet of Things course.