



**PennState**  
Abington

# BRAKING SYSTEM

IST440W – INFORMATION SCIENCES AND  
TECHNOLOGY INTERGRATION APPLICATION



## TEAM 1

FALL 2016 | PROFESSOR JOE OAKES |

THE PENNSYLVANIA STATE UNIVERSITY - ABINGTON

## Table of Contents

SYSTEM DOCUMENTATION.....	3
Introduction.....	3
Purpose.....	3
System Overview .....	3
Design Constraints .....	4
Roles and Responsibilities .....	4
System Architecture.....	5
Database Design.....	8
System Security and Integrity Controls .....	8
DEVELOPMENT DOCUMENTATION .....	9
Requirements: .....	9
Source Code: .....	9
Android Development.....	16
Libraries .....	23
Development Methodologies .....	24
Test Analysis.....	25
USER DOCUMENTATION .....	25
Purpose of the Document.....	25
Button used in the user interface.....	25
Brief Description.....	26
Connecting to the application .....	26
Functionalities.....	27
MAINTENANCE DOCUMENTATION .....	32
Raspberry Pi Ports.....	32
How to set up a Raspberry Pi.....	33
Installation of the OS .....	33
Assembly .....	34
Troubleshooting .....	34
Video.....	34
Operation System.....	35
Networking .....	35

# SYSTEM DOCUMENTATION

## Introduction

This System was created to outline the proposed system design for IST 440W class. The Project was to simulate real braking system in Raspberry Pi with servo control. This system intended to be some new features on braking system of a vehicle. By designing, testing, and deploying the system, the next IST 440W class will improve its functionality, capabilities, stability, and many other aspects such as communicating between devices. This document and technical specification listed herein comply with all industrial standards and infrastructure.

## Purpose

The purpose of this System Documentation is to provide a description for how the new Braking System will be constructed. The System Documentation was created to ensure that the System design meets the requirements specified in the Project requirements documentation. The System Documentation provides a description of the system architecture, software, hardware, and security.

## System Overview

There are thousands of accidents every years and a lot of them were driver did not pay attention on the road. This Mini Braking System will be utilize existing technology to develop a new features on Braking System in real vehicle because sensors had been existed for many years a sensor to trigger braking is possible in theory.

The Braking System is designed into two Operating System but mainly in Linux for operation. Linux System created connection to a servo, which control the brakes of all a demo car. The Servo will have different pattern for different kind of brakes such as normal brake, parking brake, anti-blocking braking system, and most important sensor braking, which sensor will be reading the distance and brake if there is an object in front of the vehicle. Another Operating System will be Android which will be display status of the braking system actually the servo. Also there are Android Buttons that will control the servos as well such as braking, abs braking, e-brake (Parking Brake) and so forth.

Android in this case is Nexus 7 will control the Linux System, in this case is raspberry Pi, using SSH (secure shell) to execute python command within the raspberry Pi. The raspberry Pi will be perform different on the servo depends on what python script is running.

The Braking System will provide the following capabilities:

- Sensor will detect any object in the front of the vehicle and send PWD signal to the Linux to start the servo which is braking.

- Android will have different android buttons to start braking.
- E-brake will trigger the servo to turn 90 degrees and hold it there.
- Abs braking will have be trigger for 4 second.

## Design Constraints

Our Team (Braking System) identified serval constraints which will impact and limit the function or design of the system. These constraints are beyond the scope of the Braking System Project but must be carefully factored into the system design. The following constraints have been identified:

1. Only the driver will be able to access the Braking System, no one else should have access to Braking System. Security is one of the important factor. Even Android is using SSH connection to raspberry Pi, java code will be hardcoded username and password. Solution was to create another login screen for user to type in and it will read from the login information.
2. Braking System must be able to communicate with other devices. In this case, we are using Xbees to communicate with other raspberry PIs. However, there is another way to communicate which is to create a socket for both devices using IP address to communicate with each other. Make sure all devices using the same method to talk.
3. The System must be able to send data from Linux to Android Device. This is the next level development which the system will be able to talk to an android and show data on Nexus 7 screen.

## Roles and Responsibilities

The following table defines the Braking System Design roles and responsibilities. This matrix also serves as the list of point of contact for issues and concerns relating to the Braking System Design.

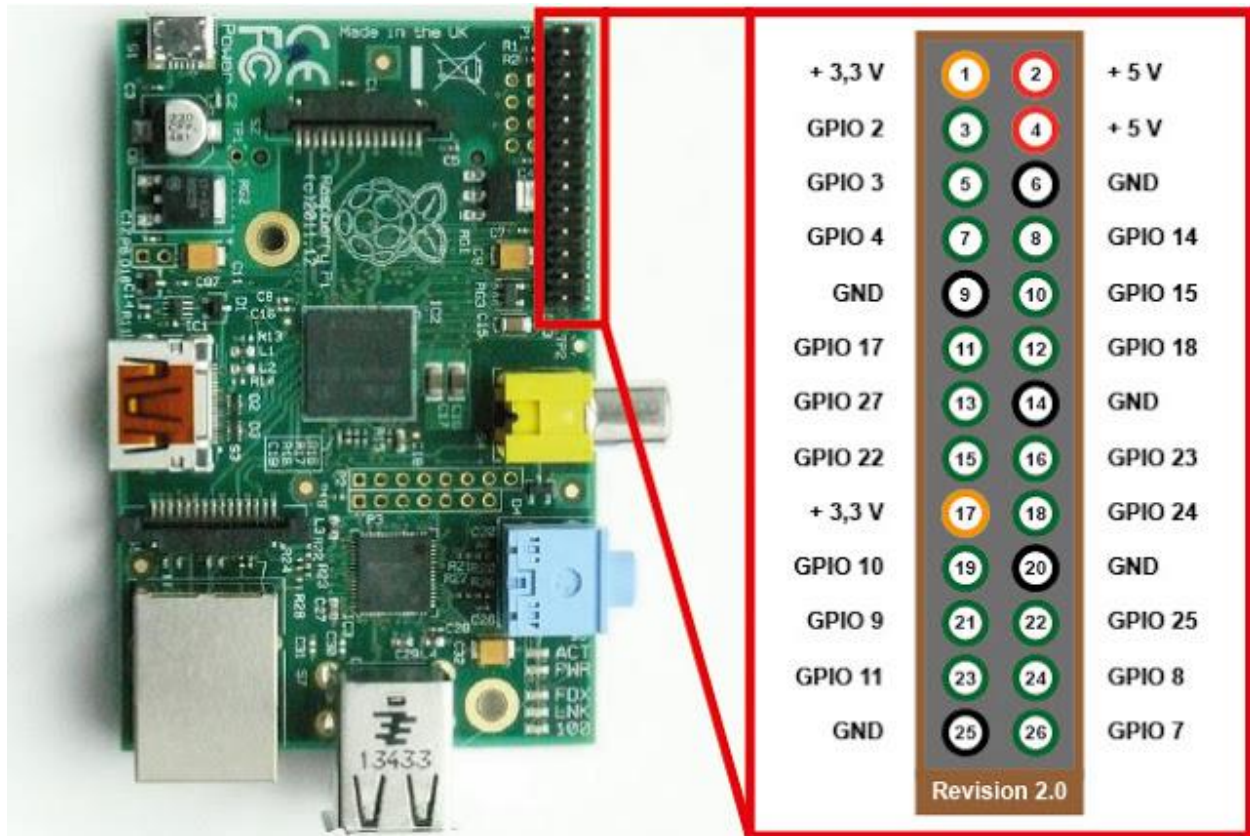
Name	Role	Phone	Email
Abu Sakif	Project Manager	2674750904	ass5260@psu.edu
Qili Jian	Android Developer	2672371026	qqj5004@psu.edu
David Austin	Python Developer	4848607354	daa5188@psu.edu
Chakman Fung	UI Designer	2675752585	cmf5538@psu.edu
Abu Chowdhury	Slacker1.0	2673704538	asc5292@psu.edu
Gary Martorana	Slacker2.0	4843472957	gjm5155@psu.edu

## System Architecture

### Hardware:

The Braking System design is based on existing hardware, technology architecture already deployed across the development world. The hardware consists of the following component:

- Raspberry Pi 3 model B
  - a. SoC: Broadcom BCM2837 (roughly 50% faster than the Pi 2)
  - b. CPU: 1.2 GHz quad-core ARM Cortex A53
  - c. GPU: Broadcom VideoCore IV @ 400MHz
  - d. Memory: 1 GB LPDDR2-900 SDRAM
  - e. USB ports: 4
  - f. Network: 10/100 MBPS Ethernet, 802.11n Wireless LAN, Bluetooth 4.0
  - g. SD Card



- Android Table Nexus 7



- 5 Voltage Servo



- Jumper Cables



### **Software:**

The Braking System designed user interface based on android development, users will enter the information to log in into the raspberry pi. The software architecture is designed to control Linux devices to perform specific operation and achieve the specific goals. The components which comprise the software architecture include:



- Raspbian, Linux OS for mini Computer, Raspberry Pi
- GPIO library for Raspbian.
- System Updates.
- Android Studio (software for android apps development)
- Android 5.01 OS in Nexus 7
- JSch library (Java Secure Channel)
  - JSch allows users to connect to an SSHD Server and use port for forwarding data, X11 forwarding and files transfer, etc.
- Java Development Kit from Oracle (utilized in Android Studio)
- GitHub (Online repository)
- Jira (Project tracking, monitoring, progress)

## Database Design

There will be a database designed by other class. The braking system will be sending JSON message to the database and database will send that message to specific devices to do some operation. However, the database was not created successfully. Therefore, our team figured out another way to communicate with other devices.

## System Security and Integrity Controls

The braking system was originally hardcoded the username and password for the raspberry pi because there is credential to log in for raspberry pi. Our team changed to different way to make it works. We designed a login screen, an xml file and background code. Therefore, users will need to put in the pi's username, password and IP address to access the brakes control. This method which added another layer of security for raspberry Pi, there will be users who know the credential will able to access the braking system.



# DEVELOPMENT DOCUMENTATION

## Requirements:

### Brake

- (Yes) Servo (Quantity: 2) Mechanical
- (No) Relay (Quantity: 4) Electrical (Motor controller using relay switches on four wheels)

### Raspberry Pi

- (Yes) The vehicle should stop within a reasonable distance.
- (Yes) Permit the operator to retain control of the vehicle must not skid when brakes are applied
- (Yes) Raspberry Pi will interact with electronic sensors, motors, and other parts.
- (Yes) Raspberry Pi will be connected to the Wi-Fi.

### GrovePi

- (Yes) It will be connected to the Raspberry Pi
- (Yes) Ultrasonic Sensor will be connected to the GrovePi
- (No) Buzzer will be provided to provide a sound alert if close to an object in amount of distance.
- (Yes) Vehicle will automatically stop when it is close to an object in amount of distance.
- (Yes) It will read distance from ultrasonic sensor.

## Source Code:

We used Raspberry Pi as main operation, which we will be using Python as our main programming language. We have several different .py programs, which are **braking.py**, **eBrakeOn.py**, **eBrakeOff.py**, **sensor\_stop.py**, and **absbraking.py**.

See braking.py code following:

#Penn State Abington

#IST 440W

#Fall 2016

#Team Pump Your Brakes

#Members: Abu Sakif, David Austin, Qili Jian, Abu Chowdhury, Gary Martorana, Chakman Fung

```

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(11,GPIO.OUT)#

p = GPIO.PWM(11,50)#PWM'Post-width Modulation' puts pin 11 to 50Hz
p.start(9)

loopStart = time.time()#time when program starts
timePassed = 0.0
try:
    while timePassed < 2:
        now = time.time()#current time
        timePassed = now - loopStart#current time minus time when program started
        timePassed = int(timePassed)#makes timePassed into integer
        p.ChangeDutyCycle(13.5)#engage brake
    if timePassed >= 2:
        p.ChangeDutyCycle(9)#disengage brakes after 2 seconds
        p.stop()#stops power to servo
        GPIO.cleanup()

except KeyboardInterrupt:
    GPIO.cleanup()
    p.stop()

```

Imported GPIO which is one of the library from raspberry. And the explanation for the code is right next to the scripts with # sign.

eBrakeOn.py

#Penn State Abington

#IST 440W

#Fall 2016

#Team Pump Your Brakes

#Members: Abu Sakif, David Austin, Qili Jian, Abu Chowdhury, Gary Martorana, Chakman Fung

```
import RPi.GPIO as GPIO
```

```
import time
```

```
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setwarnings(False)
```

```
GPIO.setup(11,GPIO.OUT)
```

```
p = GPIO.PWM(11,50)#PWM'Post-width Modulation' puts pin 11 to 50Hz
```

```
p.start(9)
```

```
try:
```

```
    p.ChangeDutyCycle(13.5)#engage emergency brake
```

```
except KeyboardInterrupt:
```

```
    GPIO.cleanup()
```

```
    p.stop()
```

We used GPIO most of our python because the servo connected the pins.

p.ChangeDutyCycle(13.5) is the how much the servo turn, but another method is to use ChangeFrequency(). To turn how many degree, you will need to adjust the values of range () with suitable frequency value. Have fun experimenting.

eBrakeOff.py

#Penn State Abington

#IST 440W

#Fall 2016

#Team Pump Your Brakes

#Members: Abu Sakif, David Austin, Qili Jian, Abu Chowdhury, Gary Martorana, Chakman Fung

```
import RPi.GPIO as GPIO
```

```
import time
```

```
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setwarnings(False)
```

```
GPIO.setup(11,GPIO.OUT)
```

```
p = GPIO.PWM(11,50)#PWM'Post-width Modulation' puts pin 11 to 50Hz
```

```
p.start(13.5)
```

```
try:
```

```
    p.ChangeDutyCycle(9)#disengage emergency brake
```

```
except KeyboardInterrupt:
```

```
    GPIO.cleanup()
```

```
    p.stop()
```

p.ChangeDutyCycle() is the how much the servo turn, but another method is to use ChangeFrequency(). Here is to reverse the servo to simulate release the e-brake, and to turn how many degree, you will need to adjust the values of range () with suitable frequency value. Have fun experimenting.

ABSbraking.py

#Penn State Abington

#IST 440W

#Fall 2016

#Team Pump Your Brakes

#Members: Abu Sakif, David Austin, Qili Jian, Abu Chowdhury, Gary Martorana, Chakman Fung

```
import RPi.GPIO as GPIO
```

```
import time
```

```
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setwarnings(False)#removes warning about pin being in use
```

```
GPIO.setup(11,GPIO.OUT)
```

```
p = GPIO.PWM(11,50)#puts pin 11 at 50Hz
```

```
p.start(9)
```

```
loopStart = time.time()#time when program starts
```

```
timePassed = 0.0
```

```
try:
```

```
    while timePassed < 4:
```

```
        now = time.time()#current time
```

```
        timePassed = now - loopStart#current time minus time program started
```

```
        timePassed = int(timePassed)#changes timePassed into Integer
```

```
        p.ChangeDutyCycle(13.5)#engages brake
```

```
        time.sleep(0.125)#time between brake pumps
```

```
        p.ChangeDutyCycle(9)#disengages brake
```

```
        time.sleep(0.125)
```

```

        if timePassed >= 4:

            p.stop()#stops power to servo after 4 seconds

            GPIO.cleanup()

except KeyboardInterrupt:

    GPIO.cleanup()

    p.stop()

```

This is ABS braking, we set a time of 4 second of abs, after 4 second the brake will be stopped. We used time.sleep to turn the servo back and forth to simulate ABS braking. ChangeDutyCycle() is the method to control the servo.

See sensor\_stop.py at the following:

```

#Penn State Abington
#IST 440W
#Fall 2016
#Team Pump Your Brakes
#Members: Abu Sakif, David Austin, Qili Jian, Abu Chowdhury, Gary Martorana, Chakman Fung

```

```

from grovepi import *
import RPi.GPIO as GPIO
import time

ultrasonic_ranger = 4 # Insert ultrasonic_ranger to D4 on the grovepi
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(11,GPIO.OUT)

```

```

p = GPIO.PWM(11,50)#PWM'Post-width Modulation' puts pin 11 to 50Hz
p.start(9)
try:
    while True:
# Read distance value from Ultrasonic
        distant = ultrasonicRead(ultrasonic_ranger)
        if distant <= 50:
            p.ChangeDutyCycle(13.5)#engage brake
        else:
            p.ChangeDutyCycle(9)#engage brake
except KeyboardInterrupt:
    GPIO.cleanup()
    p.stop()

```

In the sensor\_stop.py, we used GrovePi to connect one of the ultrasonic sensor to read the distance. 50cm is the maximum that we put it. If there is an object in 50cm, the servo will apply 90 degree to pull the brake.

All the python scripts above are what we have to do four different types of braking. Also that's the end of the Pi Development. Next Part is the Java Development on Android Device which will show status of the Pi also to control the Pi.



## Android Development

This is the First activity class called **Login.java**

```
/**
 * Author: Chakman Fung, Qili Jian, Abu Sakif, David Austin.
 * IST 440W
 * Penn State Abington
 * TEAM 1, Pump Your Brake
 */

package ist440.brakescontrol;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class Login extends AppCompatActivity {
    //define variables
    Button logIn;
    EditText username, password, hostname;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        logIn = (Button) findViewById(R.id.main_Enter);
    }

    // initialize variables for connecting to pi
    public void Enter(View view) {
        username = (EditText) findViewById(R.id.main_username);
        password = (EditText) findViewById(R.id.main_password);
        hostname = (EditText) findViewById(R.id.main_hostname);

        //prompt for user to enter the info variables

        Intent myIntent = new Intent(Login.this, BrakesControl.class);
        startActivity(new Intent(Login.this, BrakesControl.class));
        myIntent.putExtra("username", username.getText().toString());
        myIntent.putExtra("password", password.getText().toString());
        myIntent.putExtra("hostname", hostname.getText().toString());
        startActivity(myIntent);
    }
}
```

```
//define variables
Button login;
EditText username, password, hostname;
```

EditText username, password and hostname value created which refer to the root element of a view from the layout page.

```
// initialize variables for connecting to pi
public void Enter(View view) {
    username = (EditText) findViewById(R.id.main_username);
    password = (EditText) findViewById(R.id.main_password);
    hostname = (EditText) findViewById(R.id.main_hostname);
```

The code below actually built an intent which provide runtime binding between separate components from other activities connected. The putExtra() method adds the EditText's value to the intent. Then, to retrieve the string value in the BrakesControl.java. The startActivity() method starts an instance of the DisplayMessageActivity specified by the Intent.

```
//prompt for user to enter the info variables

Intent myIntent = new Intent(Login.this, BrakesControl.class);
startActivity(new Intent(Login.this, BrakesControl.class));
myIntent.putExtra("username", username.getText().toString());
myIntent.putExtra("password", password.getText().toString());
myIntent.putExtra("hostname", hostname.getText().toString());
startActivity(myIntent);
```

The code below was the second activity called **BrakesControl.java** that we created. It received EditText value from the Login.java activity through intent method.

```
public class BrakesControl extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_brakes_control);

        Intent myIntent = getIntent();

        // initialize variables for connecting to pi from the Login java class
        final String username = myIntent.getExtras().getString("username");
        final String password = myIntent.getExtras().getString("password");
        final String hostname = myIntent.getExtras().getString("hostname");
        //final String hostname = "192.168.1.105";
```

In the **AndroidManifest.xml** file which every application must have in the root directory. The manifest file gives important information about the app to the android system, which the system must have before it can run any of the app's code.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ist440.brakescontrol">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".Login">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".BrakesControl"
            android:screenOrientation="portrait">

        </activity>
    </application>
</manifest>
```

We setup the application's starting activity as the login class which the user need to type in the IP address, Pi' username and password for connecting SSH from the android apps to the raspberry pi.

```
<activity android:name=".Login">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

The second activity is the BrakeControl and we set the orientation of BrakeControl activity to portrait.

```
<activity android:name=".BrakesControl"
    android:screenOrientation="portrait">

</activity>
```

Main activity Java class called BrakeControl.java:

```
/**
 * Author Qili Jian
 * IST 440W
 * Penn State Abington
 * TEAM 1, Pump Your Brake
 */
package ist440.brakescontrol;

import android.content.Intent;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.CompoundButton;
import android.widget.ImageView;
import android.widget.ToggleButton;

import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;

import java.util.Properties;

public class BrakesControl extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_brakes_control);

        Intent myIntent = getIntent();

        // initialize variables for connecting to pi from the Login java class
        final String username = myIntent.getExtras().getString("username");
        final String password = myIntent.getExtras().getString("password");
        final String hostname = myIntent.getExtras().getString("hostname");
        //final String hostname = "192.168.1.105";

        final String scriptDir = "python /home/pi/PSUABFA16IST440/BrakingSystem";
        //final String scriptDir2 = "python /home/pi/PSUABFA16IST440/BrakingSystem/E-
Brake Python";
        final int port = 22;

        //Creating toggle button for all brakes
        final ToggleButton absBrake = (ToggleButton) findViewById(R.id.absBrake);
        final ToggleButton sensorBrake = (ToggleButton)
findViewById(R.id.sensorBrake);
        final ToggleButton eBrake = (ToggleButton) findViewById(R.id.eBrake);
        final ToggleButton brake = (ToggleButton) findViewById(R.id.brake);

        //Creating the icon for each brake
        final ImageView fRight = (ImageView) findViewById(R.id.fRight);
        final ImageView rLeft = (ImageView) findViewById(R.id.rLeft);
        final ImageView rRight = (ImageView) findViewById(R.id.rRight);
        final ImageView fLeft = (ImageView) findViewById(R.id.fLeft);
    }
}
```

```

//set all icon to invisible
fRight.setVisibility(View.INVISIBLE);
rLeft.setVisibility(View.INVISIBLE);
rRight.setVisibility(View.INVISIBLE);
fLeft.setVisibility(View.INVISIBLE);

brake.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
{
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
    {
        if (isChecked){
            boolean success = true;
            new AsyncTask<Integer, Void, Void>() {
                String command = scriptDir + "/braking.py";

                protected Void doInBackground(Integer... params) {
                    try {
                        executeRemoteCommand(username, password, hostname,
command, port);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return null;
                }
            }.execute(1);
            if (success) {
                fRight.setVisibility(View.VISIBLE);
                rLeft.setVisibility(View.VISIBLE);
                rRight.setVisibility(View.VISIBLE);
                fLeft.setVisibility(View.VISIBLE);
            }
        } else {
            boolean success = true;
            new AsyncTask<Integer, Void, Void>() {
                String command = scriptDir + "/sensor_stop.py";

                protected Void doInBackground(Integer... params) {
                    try {
                        executeRemoteCommand(username, password, hostname,
command, port);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return null;
                }
            }.execute(1);
            if (success) {
                fRight.setVisibility(View.INVISIBLE);
                rLeft.setVisibility(View.INVISIBLE);
                rRight.setVisibility(View.INVISIBLE);
                fLeft.setVisibility(View.INVISIBLE);
            }
        }
    }
}

```

```
    }  
    });
```

To initialize all button and find the matching ids on xml file

```
//Creating toggle button for all brakes  
final ToggleButton absBrake = (ToggleButton) findViewById(R.id.absBrake);  
final ToggleButton sensorBrake = (ToggleButton) findViewById(R.id.sensorBrake);  
final ToggleButton eBrake = (ToggleButton) findViewById(R.id.eBrake);  
final ToggleButton brake = (ToggleButton) findViewById(R.id.brake);
```

Initialize all image view and find their specific ids in xml file

```
//Creating the icon for each brake  
final ImageView fRight = (ImageView) findViewById(R.id.fRight);  
final ImageView rLeft = (ImageView) findViewById(R.id.rLeft);  
final ImageView rRight = (ImageView) findViewById(R.id.rRight);  
final ImageView fLeft = (ImageView) findViewById(R.id.fLeft);
```

Action for the Buttons, we used AsyncTask <> because it allowed the app to perform background operation and publish on the UI thread without having to manipulate threads or the handlers.

```
brake.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  
  
    @Override  
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
        if (isChecked){  
            boolean success = true;  
            new AsyncTask<Integer, Void, Void>() {  
                String command = scriptDir + "/braking.py";  
  
                protected Void doInBackground(Integer... params) {  
                    try {  
                        executeRemoteCommand(username, password, hostname, command,  
port);  
                    } catch (Exception e) {  
                        e.printStackTrace();  
                    }  
                    return null;  
                }  
            }.execute(1);  
            if (success) {  
                fRight.setVisibility(View.VISIBLE);  
                rLeft.setVisibility(View.VISIBLE);  
                rRight.setVisibility(View.VISIBLE);  
                fLeft.setVisibility(View.VISIBLE);  
            }  
        }  
    }  
});
```

```

        } else {
            boolean success = true;
            new AsyncTask<Integer, Void, Void>() {
                String command = scriptDir + "/sensor_stop.py";

                protected Void doInBackground(Integer... params) {
                    try {
                        executeRemoteCommand(username, password, hostname, command,
port);

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return null;
                }
            }.execute(1);
            if(success){
                fRight.setVisibility(View.INVISIBLE);
                rLeft.setVisibility(View.INVISIBLE);
                rRight.setVisibility(View.INVISIBLE);
                fLeft.setVisibility(View.INVISIBLE);
            }

        }

    }
});

```

Next is the method that we used to SSH into raspberry Pi, We used JSch library because JSch is pure java implementation of SSH2, it allowed you to connect to an assigned ssh server and use port forwarding files transfer and data transfer. We used JSch library to execute our python scripts our raspberry Pi.

```

// This method is connects to the pi, and runs the command given
public void executeRemoteCommand (String username,
                                   String password,
                                   String hostname,
                                   String command,
                                   int port)

    throws JSchException {
    // New Jsch object for connecting
    JSch jsch = new JSch();

    // Try to create a session using username, hostname, and port
    Session session = null;
    try {
        session = jsch.getSession(username, hostname, port);
    } catch (JSchException e) {
        e.printStackTrace();
    }

    // Set the password for the session
    assert session != null;
    session.setPassword(password);

    // Avoid asking for key confirmation

```



```

Properties prop = new Properties();
prop.put("StrictHostKeyChecking", "no");
session.setConfig(prop);

// Attempt to connect
try {
    session.connect();
} catch (JSchException e) {
    e.printStackTrace();
}

// SSH Channel
ChannelExec channelssh = null;
try {
    channelssh = (ChannelExec)
        session.openChannel("exec");
} catch (JSchException e) {
    e.printStackTrace();
}

// Use this if there is any need for output to return to android
// ByteArrayOutputStream baos = new ByteArrayOutputStream();
//
// channelssh.setOutputStream(baos);

// Execute command
assert channelssh != null;
channelssh.setCommand(command);
try {
    channelssh.connect();
} catch (JSchException e) {
    e.printStackTrace();
}
channelssh.disconnect();
}
}

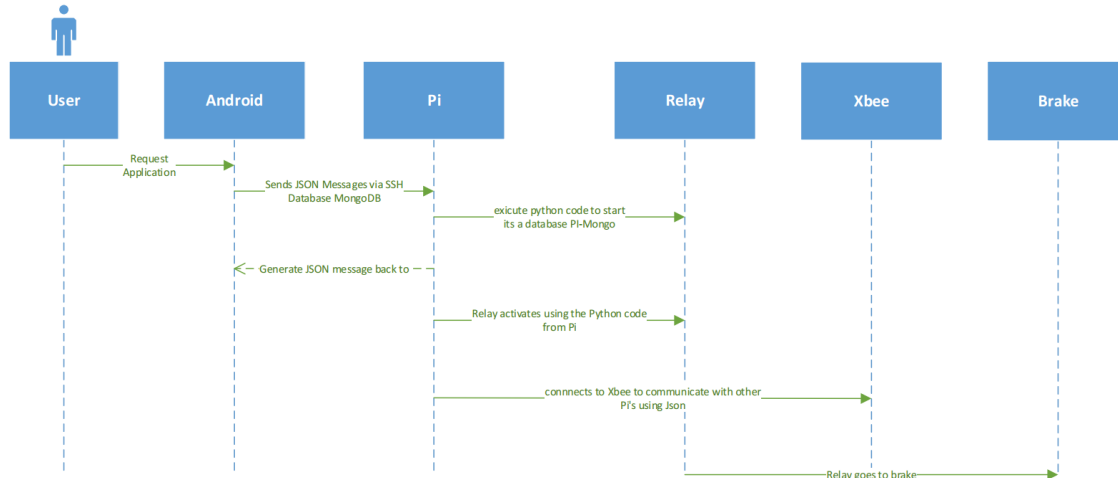
```

## Libraries

- GPIO
- GrovePi
- RPI.GPIO
- JSch Library 0.1.54
- JZzlib Library 1.07

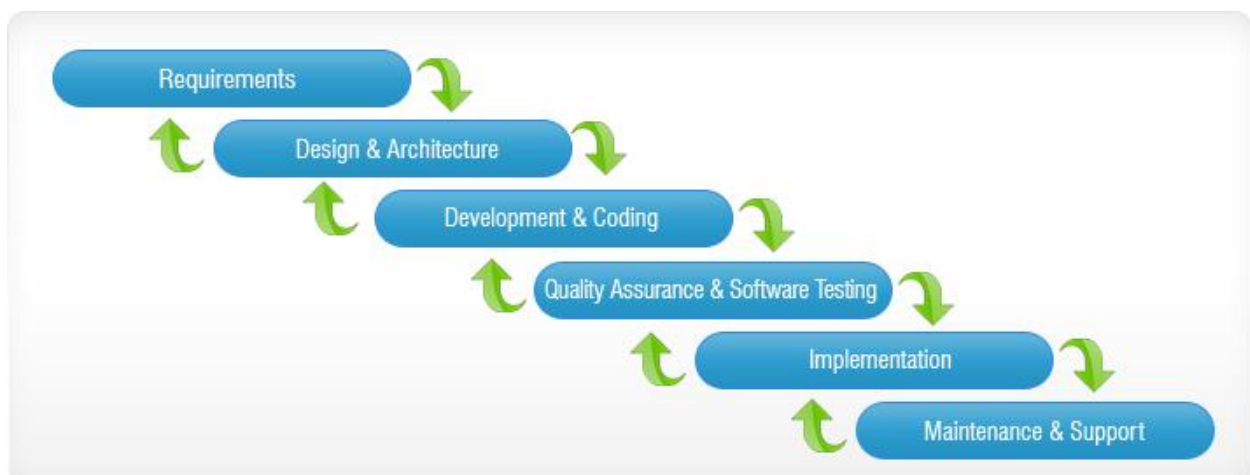
## Development Methodologies

We used Waterfall model Development Methodologies to develop our project this time.



In the beginning of the making requirement, our team wasted a lot of time on it because we could not find any mini parts for braking system. After a long time, we changed our objective to control a servo from relays, which will pull the brake in autonomous car. That we used different pattern for the different types of brakes.

Finally Java and python codes were done, and Android will be able to control raspberry pi using SSH.



## Test Analysis





### USER DOCUMENTATION

#### Purpose of the Document

The purpose of this document is to describe the android application supplied to the user

#### Button used in the user interface

Throughout this document, the chart below are used to underline points or important button

	Ultrasonic Sensor switch button
	Braking on and off switch button
	parking brake or emergency brake switch button
	Anti-lock braking system switch button



#### Caution

Because we only have one servo on the Car. So application can only run one button at a time. You must make sure the button is off before you trying to start to click another button to execute the .py files from the raspberry pi.

## Brief Description

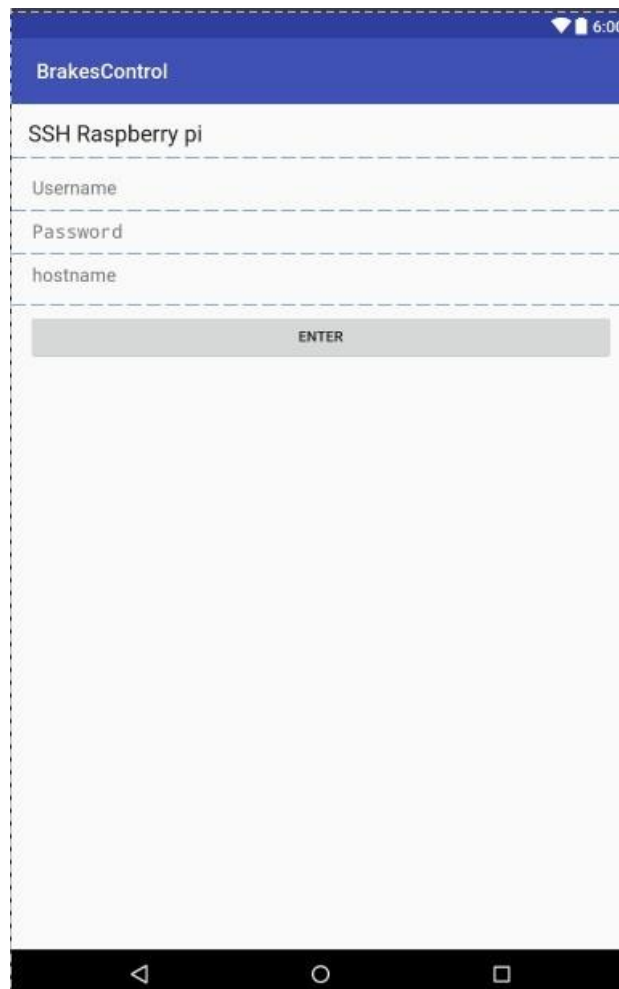
This android application is for Braking System based on raspberry pi, it will enabled the remote control SSH files for Android Apps. Android apps button will send command to the Raspberry pi to run one of the python application in pi and servo will start the braking, Anti-lock braking system, ultrasonic sensor and parking brake. There will be red tiles notification on which tire's brake was applied.

## Connecting to the application

### How to enter your Pi's IP Address number:

First, you will need to connect though the raspberry pi, finding your Pi's IP Address by type `ifconfig wlan0` or `sudo ifconfig` (based in Linux operation system). Next to the **wlan0** entry you will see ex: (inet addr: **192.168.1.10**) which is the IP address of the Raspberry Pi.

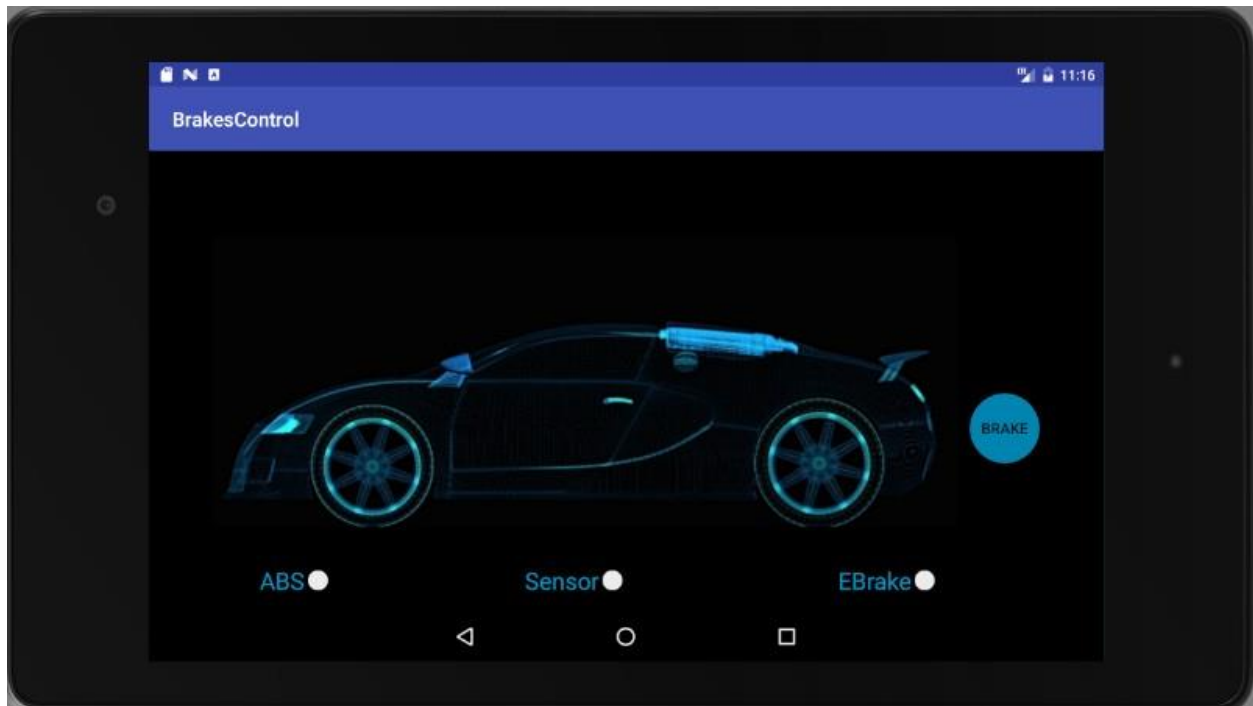
Hostname is your IP address, for the username and password, use your own username and pi' password that you setup before.



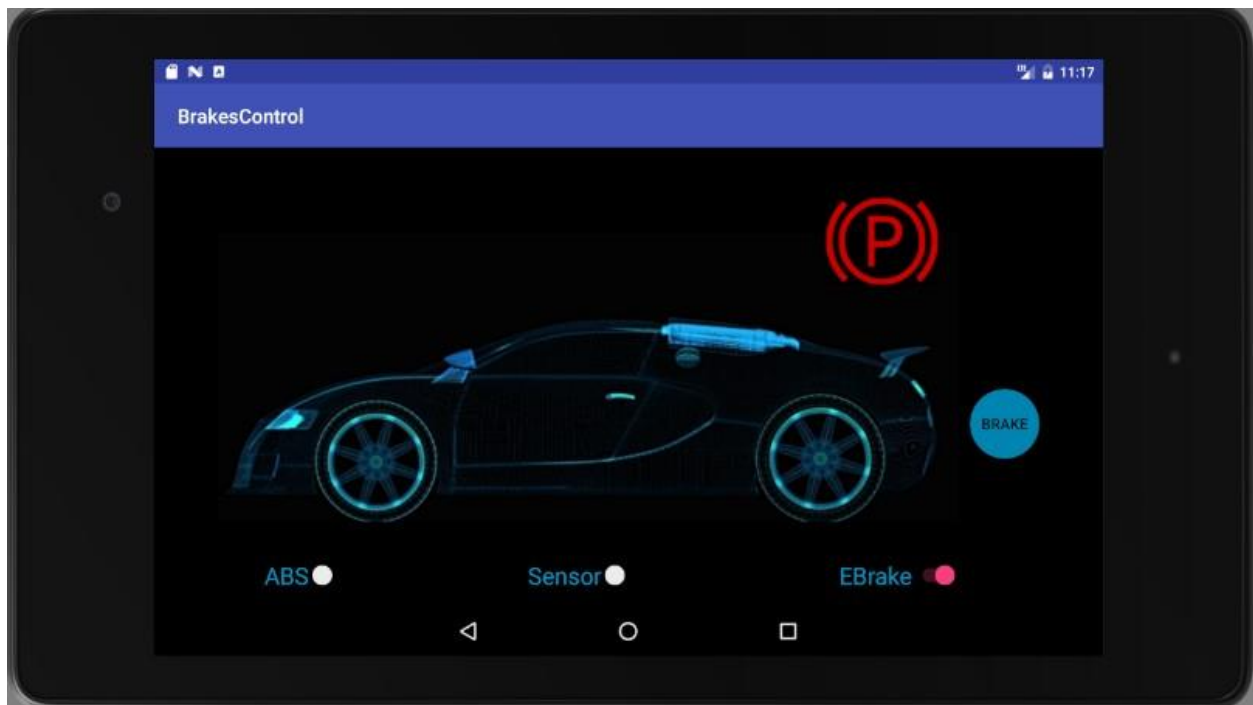
## Functionalities

The image below shows the how the User interface looks like.

The image below shows the UI, Brake button which will execute [braking.py](#)

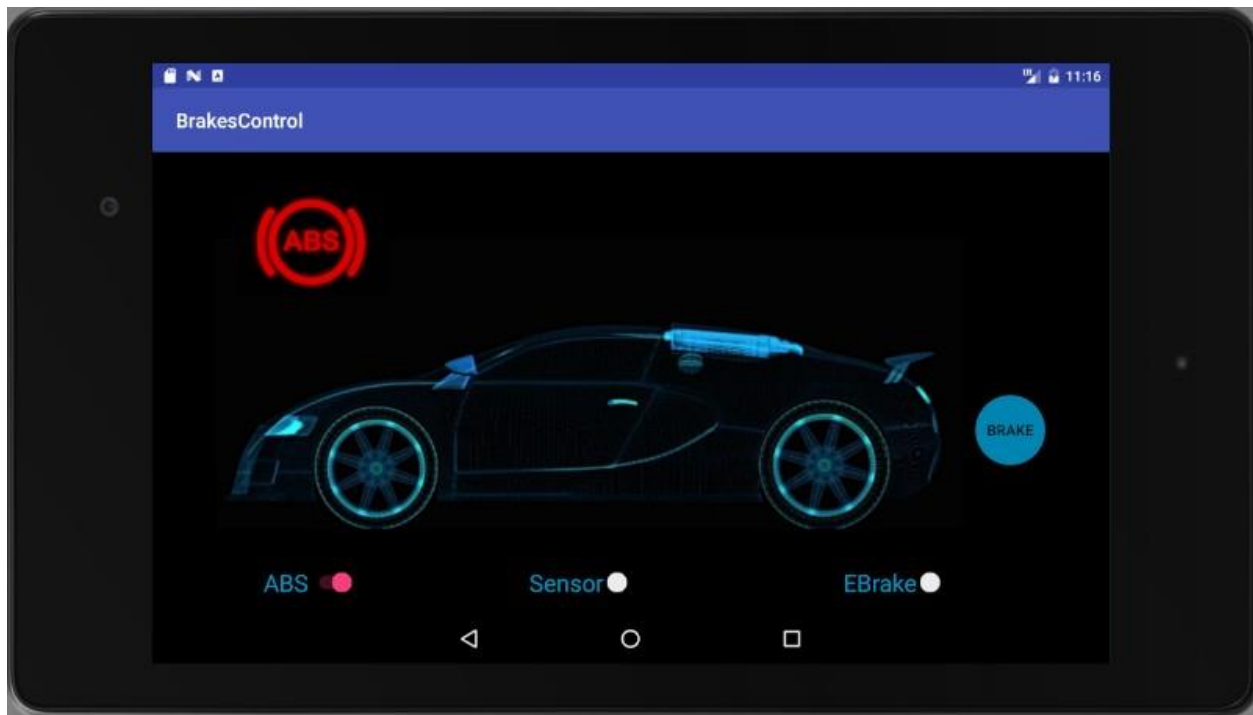


The image below shows the Users clicked the E-Brake button which will execute [engage\\_e\\_brake.py](#). If you click on more time, it will execute [disengage\\_e\\_brake.py](#)

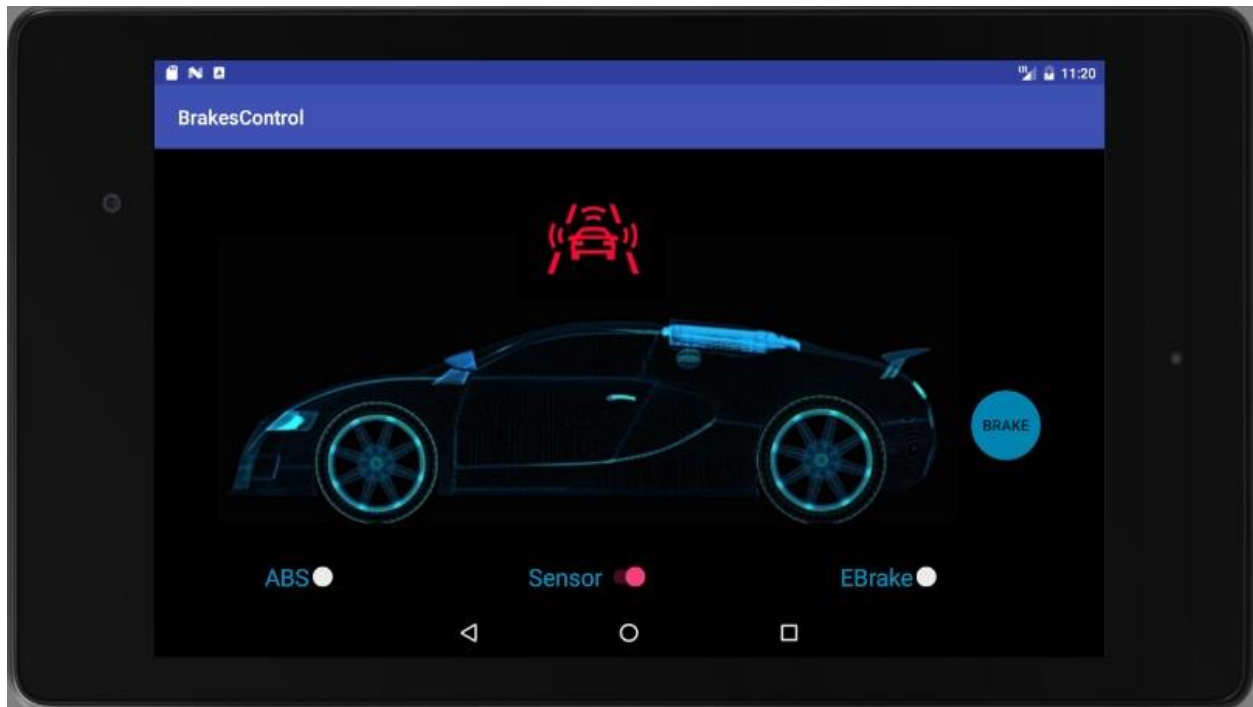


The image below shows the Users clicked the ABS button which will execute [absbraking.py](#)

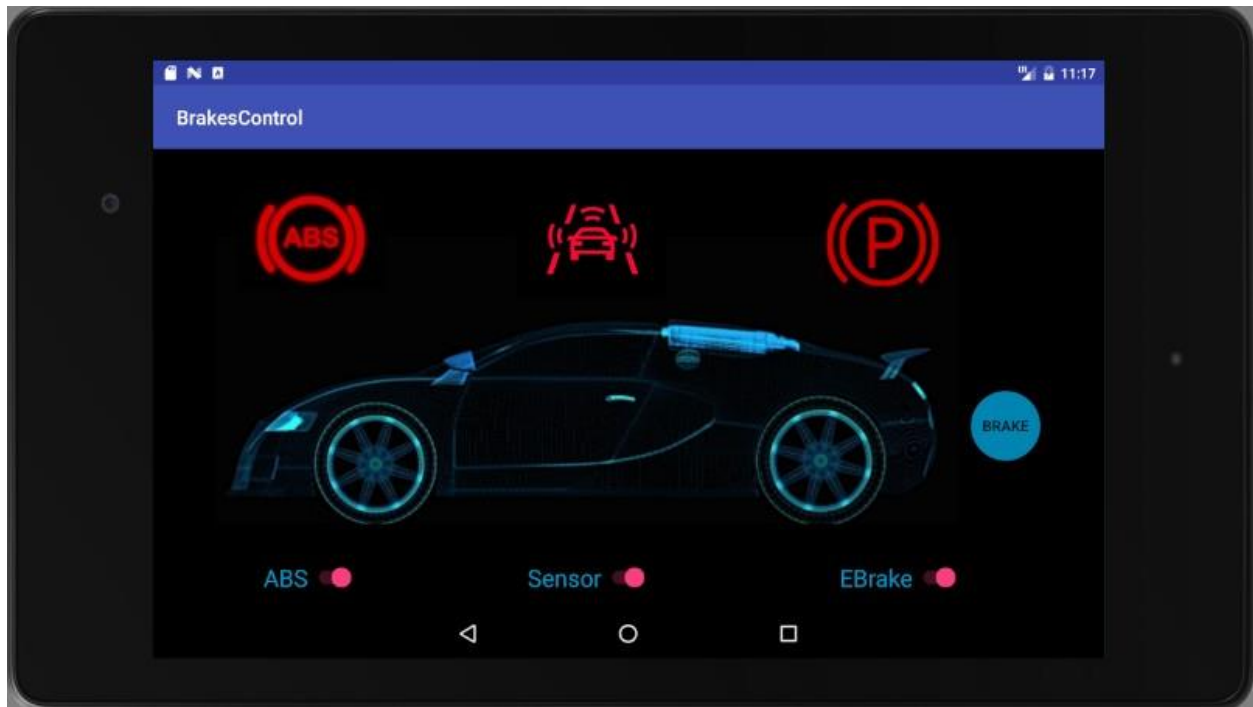




The image below shows the Users clicked the Sensor button which will execute [sensor\\_stop.py](#)

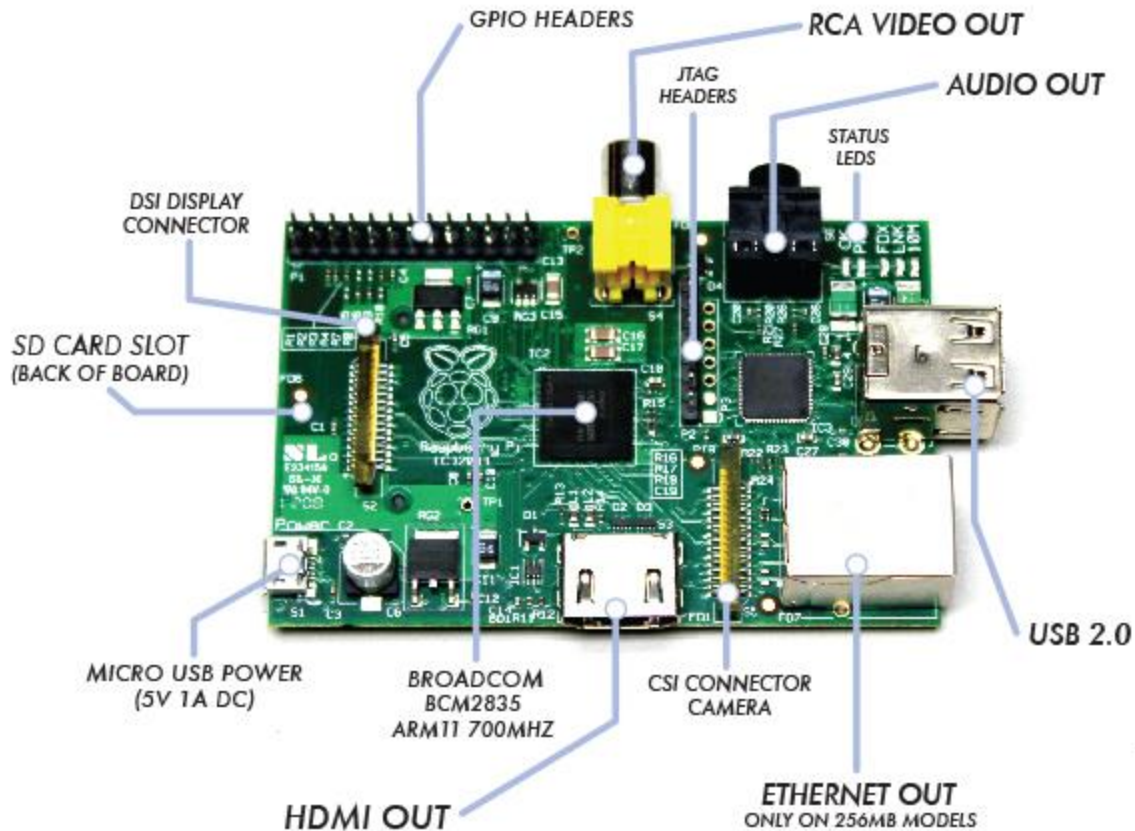


The image below shows the Users clicked and turned on all the buttons



## MAINTENANCE DOCUMENTATION

### Raspberry Pi Ports



## How to set up a Raspberry Pi

To set up a raspberry, you will need the following:

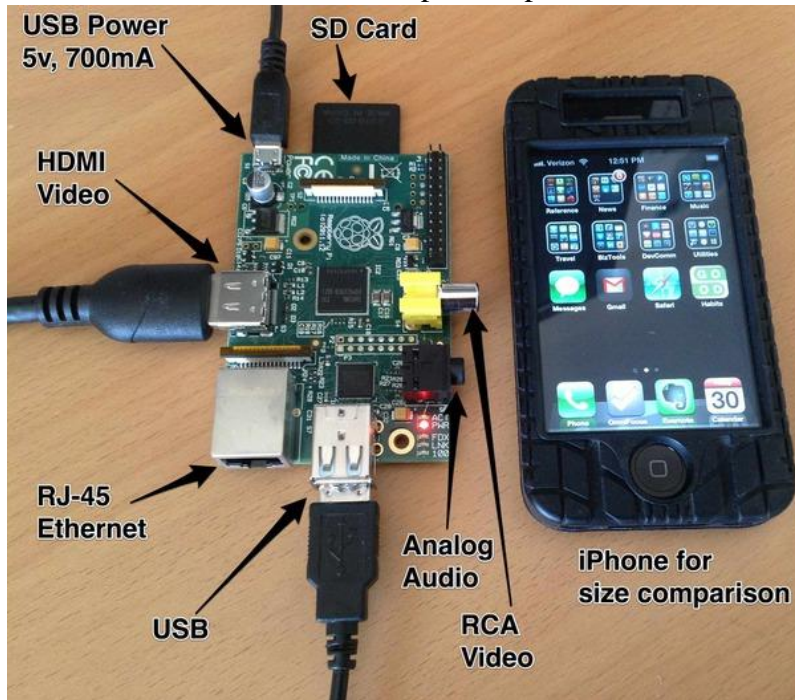
- HDMI Cable and DVI-d adapter
- Micro USB Cable for power supply
- A monitor ( HDMI or DVI )
- Keyboard and mouse
- Micro SD card for OS

## Installation of the OS

- Download an image from raspberry pi website.
- Use Win32DiskImager to overwrite the SD Card.
- Put SD card into Pi after completion.
- With PIXEL no login required, but default user and password are: pi and raspberry

## Assembly

- Plug in HDMI, keyboard, mouse.
- Put the Grovepi on the raspberry Pi if needed.
- Final insert micro USB cable to power up.



## Troubleshooting

### General Guild

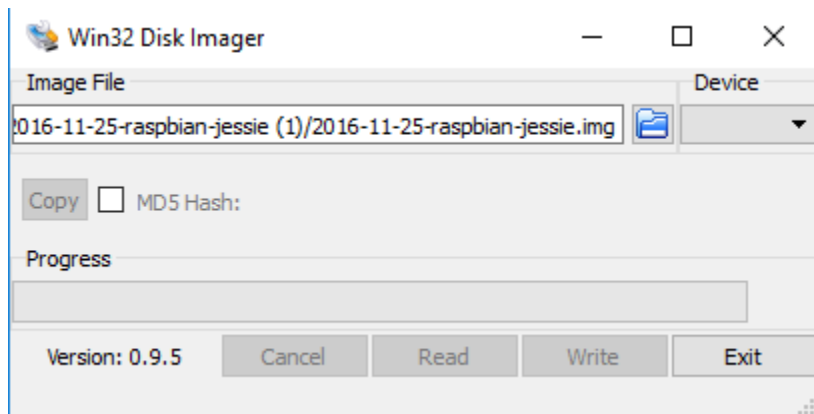
- Make sure power supply USB cable is good, not defected.
- Make sure the HDMI Cable is all the way plugged in.
- Put your raspberry pi on the flat surface because the power cable and HDMI are easily affected by any movement.
- Always keep the firmware update
- Make sure do a proper shut down on Pi.
- Do not unplug the SD card while the device is on, it will crash the I/O data.

## Video

- Change the monitor to correct source (HDMI or DVI)
- If the Monitor shows nothing, try another ports.

## Operation System

- First thing to update and upgrade the system use script following:
  - Sudo apt-get update
  - Sudo apt-get upgrade -y
  - Sudo apt-get dist-upgrade
  - Sudo git clone <https://github.com/DexterInd/GrovePi> if needed
- If files corrupted, you can always do a hard reset by overwrite the OS again



## Networking

There is another to get into Raspberry Pi is using SSH.

- Make sure host name or IP address is correct.
- To find out IP, type ifconfig in terminal to find out IP.
- Port is 22 is most of the case except other specific servers.

