MAYDAY

Batch Number:180437

MEMBERS INVOLVED:

1. Gurmehar Singh Soni	[18BCD7095]
2. Saswata Haldar	[18BCD7096]
3. Rahul Ganesh Ragella	[18BEC7045]
4. SK Saddam Hussain	[18BCE7022]
5. Sharaj Raja Chandran	[18BCD7044]
6. Viswanadha Vedvyas	[18BCE7190]

UNDER THE GUIDANCE OF:

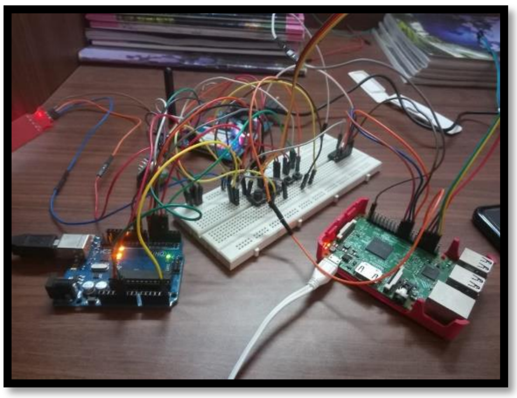
Prof. Amit Kumar

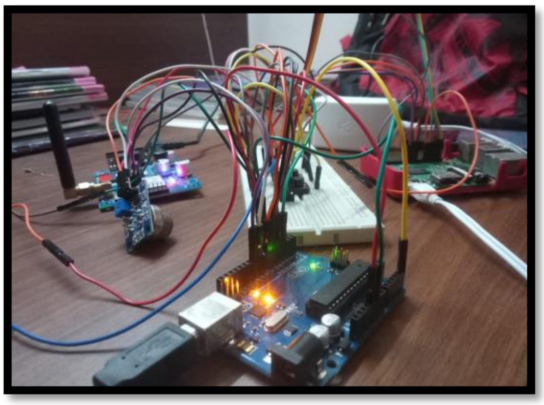
Department of Mechanical Engineering.

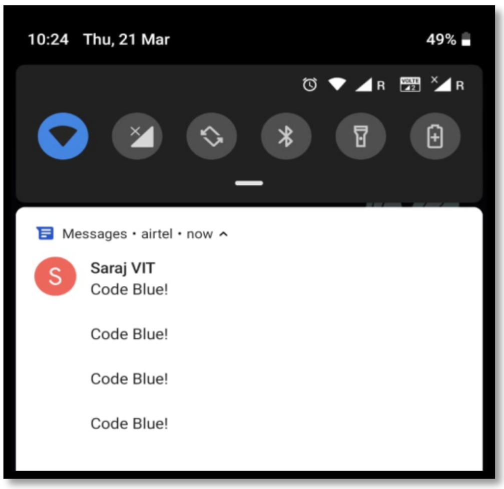
ABSTRACT

Emergency response system using Raspberry pi is a smart alternative to existing inefficient systems. The main goal of the project is to reduce the time to react and send a distress signal to the concerned authorities. As a complimentary result we also achieve better co-operation between concerned authorities. Any authorized person can operate the module whenever the need arises. As the system is Raspberry Pi based; after the prototype will go into mass production it will not only be very cost effective but also available easily, making it a go-to emergency response system. As Raspberry Pi is a very simple platform the maintenance cost also decreases exponentially with very less human effort required. As the system is a stand-alone system, it not only avoids the hassle of multiple servers and systems but also makes it extremely safe. The stand-alone feature also makes the device very reliable and gives it a very high success rate.

All in all the system will provide a great advantage over the existing emergency response models in a very consumer friendly and cost effective way.







10:24 Thu, 21 Mar 49%

Messages • airtel • now

Saraj VIT

Code Blue!

Code Blue!

Code Blue!

Code Blue!

CONTENTS

LIST OF FIGURES

INTRODUCTION

We live in a society where the need for an emergency response system is important. And when it comes to dealing with personal or institutional emergencies or distress calls, there's no need for one to have second thoughts on whether an emergency response system is required or not. For a long time, it has played a major role in the major part of our life. Through the years, many businesses and institutions all over the world have used emergency response systems to recover their stolen assets and to save lives. As a result, all these organisations make sure that they deploy the best emergency response system; required when someone breaks through their security systems.

Some of the identified core elements of the emergency response field are:

Recovery of stolen assets or belongings (personal or institutional), crisis and disaster management, saving a life among many others. Emergency response has become very important in today's world, as a result more methods need to be adopted to confront the attempts to bypass security and get away with it. In the past humans themselves had to reach the concerned authority which is far from ideal. Currently, emergency response works through phone lines on which time is wasted as the details are to be communicated clearly between the victim and the operator. If the victim is in shock or the communication lines are down this method fails. But with the advent of technology we have a come a long way from that. With the present technology we have more advanced machinery and algorithms that helps to make sure that quick and precise emergency response is ensured to the employer.

Our device is one such device that employs and makes use of the modern technology to ensure the same. It's kind of a stand-alone device that's powered by a Raspberry Pi, a GSM module and a few more other components that help in the smooth working of the system. Even though the project's intention is to enhance the emergency response in medical care institutions during any kind of medical emergency, the same project can be implemented in various other scenarios. As stated, our project on a click of a button sends a distress signal (message) to the concerned authorities depending on the situation. The authorities then without having the need to communicate or cross checking can proceed to successfully tackle the task in hand.

Though at present our project has integrated everything that can be done in a given short span of time, we believe it still has the potential to incorporate more details and facilities, which will take our project to the next level with several other add-ons making the procedure of emergency response way faster and easier to implement.

BACKGROUND

The reason behind the success of this project resides with the six-member team that toiled days behind this project, trying to resolve the unforeseen issues that aroused from the same. Even though the project might seem to be simple in concept, the implementation requires a lot of coding for each component to be integrated.

As stated in the introductory page, when the time came for the team to choose a topic for the project, the team picked this one because after all we believed that lack of an effective emergency response system is something that can affect life adversely. And if there’s something that we could do to ensure better emergency response, it would definitely help to improve life in one way or another.

So, then the whole team put their hearts and will to it, started from scratch, improvising solutions to all the unforeseen problems and finally integrated all the components so as to deliver our promise.

We are also extremely grateful to and indebted to our project guide and all other friends who gave us a hand during the difficult times.

The detailed facts and technical details regarding the project will be discussed in the following pages of the report.

PROBLEM DEFINITION

“Better safe than sorry” - Emergencies never come announced and as such, one must always be prepared for them in any way they can. Some emergencies require rapid response, but the modern man covered in things to do may fall short in responding or may not see the emergency coming at all. Apart from his endless commitments in daily life, man has something which can turn the tides, technology. If only there was something that could not only help detect a mishap but also support in finding solutions and is made for the common man. With MAYDAY this all becomes possible. It is equipped with the ability to sense water coagulations and gas leakages which can go on to cause fatal situations. It also has the correct and immediate answer for distress calls.

Heart attacks, burglary, fire or even bomb threats can be reported with the press of a simple button. The distress message will reach the concerned authority, saving one the reflex and the decisiveness required to act in these situations. MAYDAY’s greatest strength lies in its portability and mutability. With a little bit of tweaking it can respond to any situation that is not already prebuilt in it. This furthers its usability as not all people expect the same emergencies or same help. Whatever the situation may be, MAYDAY will adapt an will constantly be on the lookout to keep its owner safe.

OBJECTIVE

Our objective is to make a stand alone emergency response system using Raspberry Pi which can send distress signals on the click of a button with the help of a GSM module. It can be used in big industries, institutions, public places, homes etc. There it can be used by any authorized individual to send a distress signal to the concerned authority. This will reduce the human dependency and time delay factor. It will also lead the industry towards automation and will be a cost effective solution.

PROCEDURE/METHODOLOGY

1. CIRCUIT BUILDING

- In the beginning gather all the components i.e. Arduino Uno, GSM Module SIM900, Breadboard or PCB, Connecting jumper wire, Power Supply, SIM Card, Raspberry pi3, Buttons, Smoke Sensor, Water Level Sensor, ADC, resistors

- Connect the buttons to D4,D5,D6,D7 on the Arduino Uno and GPIO 6,13,19,26 on the Raspberry Pi

- Connect the Smoke Sensor to GPIO 5 on the raspberry pi 3 and D3 on the Arduino Uno.

- Connect the water level sensor to ADC which in turn is connected to GPIO 11,10,9,8 on raspberry pi and A5 on Arduino Uno.

- After this the gsm module is connected . Connect Rx,Tx pin on the module to the D10,D9 pin on the board respectively. Vcc pin is connected to the 5V supply and GND pin is connected to ground on the Arduino Uno. Supply the gsm module with 12V supply. Insert a sim in the slot provide.

2. CODING

- Open Arduino IDE and write the code.
- Compile the code.
- Upload the code to the board by clicking on the upload button and connecting the board to one of the usb ports on the Raspberry Pi. Ensure that you have selected the correct port.
- Make sure that the gsm module is disconnected from the board while the code is

- getting uploaded.
- Once the code is uploaded, connect the gsm module to the board and the power
- supply.
- Allow the module to stabilize.

- Now open Python IDLE in raspberry pi.
- Write the code.
- Run the code.
- The connected modules should run in the required fashion.

3. RUNNING THE SYSTEM

- Press any one of the buttons i.e "Code Red", "Code Blue", "Code Yellow", "Code Orange"
  The Gsm module will send the messages to the required authorities and the website hosted on the Raspberry Pi will get updated accordingly i.e Police for code red, Hospital for code blue, Fire station for code orange, Family Members for code yellow.

- If the Smoke Sensor detects any Smoke, the message is sent automatically to the fire station and the website is updated as "Fire Station"

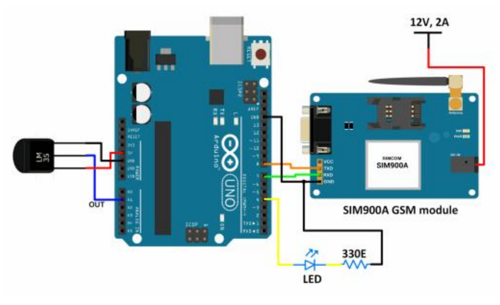
- If the Water Level Sensor detects water logging, the message is sent automatically to the fire station and the website is updated as "Fire Station".

4. COMPONENTS EXPLAINED

1. GSM Module

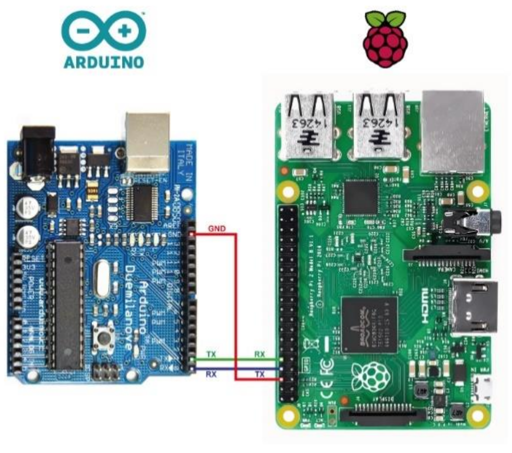
The GSM module we used for our project is SIM900A. The 900 in the name signifies its working frequency that is 900 Hz or 1800 Hz. The power requirement for this module is 12V and 1A.The module has three led which help us in finding if the module is working correctly. The band rate can be varied between 9600 – 115200.
The first led is power which glows red when the dc power is given to the module. The second led is the status led which will glow blue if the power provided is sufficient for the working of the module. The third pin is the network led will blink green with the blink rate of 1 blink per second which means the module is searching for network. When the module is connected the network, we see the blink rate drops to 1 blink per 3 sec.
The antenna used with the module is a SMA antenna which stand for standard male antenna. The module has 10 pins but for this project we only require 4 pins, namely

Tx, Rx, GND, VCC. The Tx pin is connected to the D2 pin of the Arduino and the Rx pin to D3 while GND to ground and VCC to 5V supply of the Arduino.
The work of the module in our project is to call and send SMS to the guard.
The circuit diagram of only GSM module connected to Arduino is



2. Arduino

The Arduino uses ATmega328 microcontroller which requires a working voltage of 5V. But the input voltage can vary between 7V to 20V.There are 14 digital pins in Arduino and 6 Analog pins. There are two power ports, one of 5V other of 3.3V. It has 3 GND pins.
The Arduino is capable of collecting data and do some calculations using that data and give output has per the result of the function and conditions written in the code.
The programming for Arduino is done in Arduino IDE which is a free software.
In our project the Arduino is used to control the GSM module, buttons and the sensors.



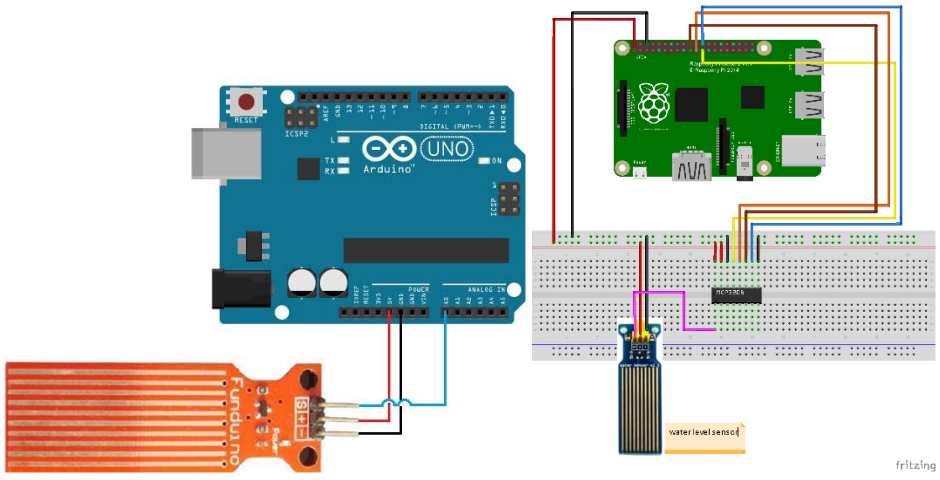
3. Raspberry Pi 3

Raspberry pi 3 is small single board computer having a 1.4 ghz Broadcom processor, wifi, Bluetooth, ethernet, usb port. It has 40 GPIO pins and runs on linux(Raspbian) os.

In our project the the website is hosted on raspberry pi and the buttons and the sensors are connected to it.

4. Water level sensor

water level sensor is an easy-to-use, cost-effective high level/drop recognition sensor, which is obtained by having a series of parallel wires exposed traces measured droplets/water volume in order to determine the water level. Easy to complete water to analog signal conversion and output analogue values can be directly read Arduino development board to achieve the level alarm effect.



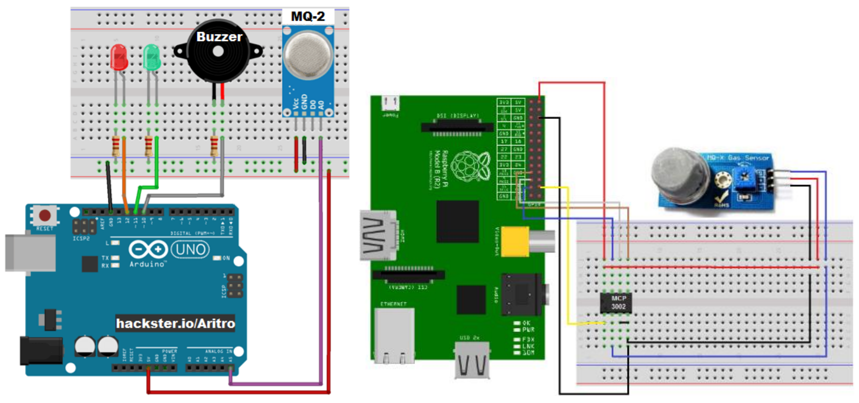
5. Smoke Sensor

The MQ-2 smoke sensor is sensitive to smoke and to the following flammable gases:

LPG Butane Propane Methane Alcohol Hydrogen

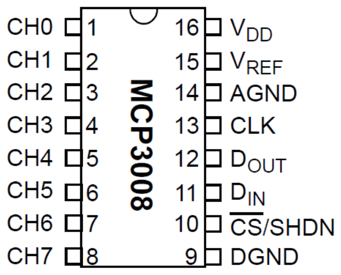
The resistance of the sensor is different depending on the type of the gas.

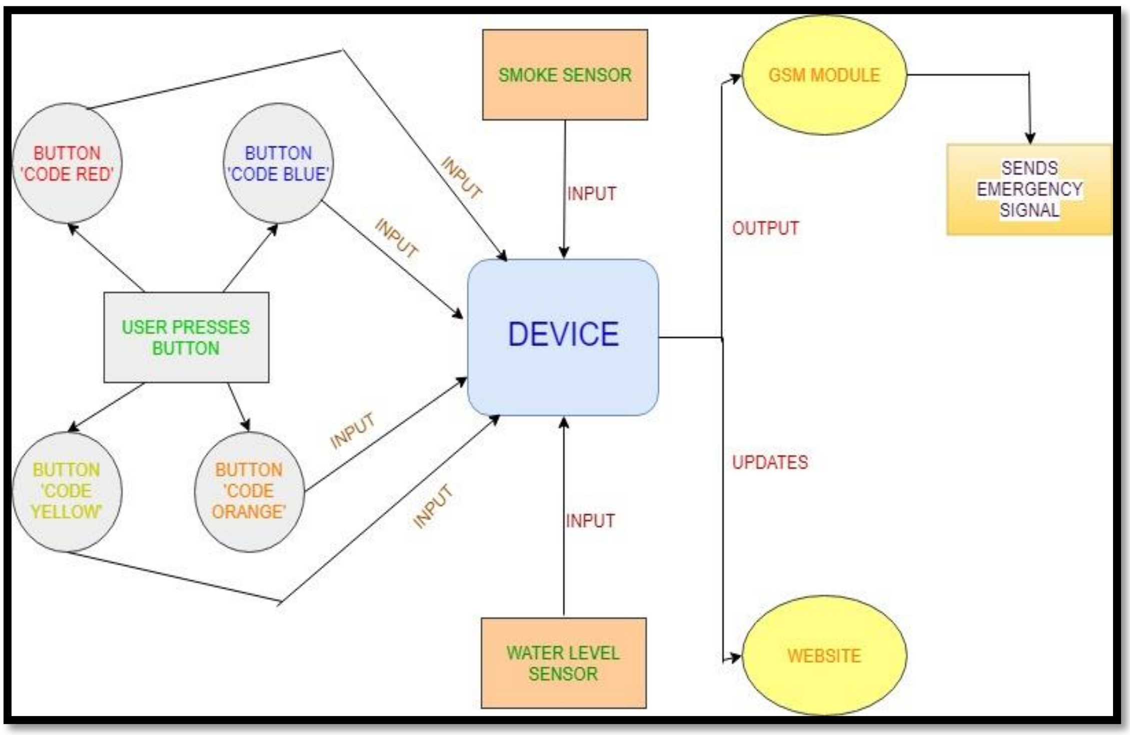
The smoke sensor has a built-in potentiometer that allows you to adjust the sensor sensitivity according to how accurate you want to detect gas.



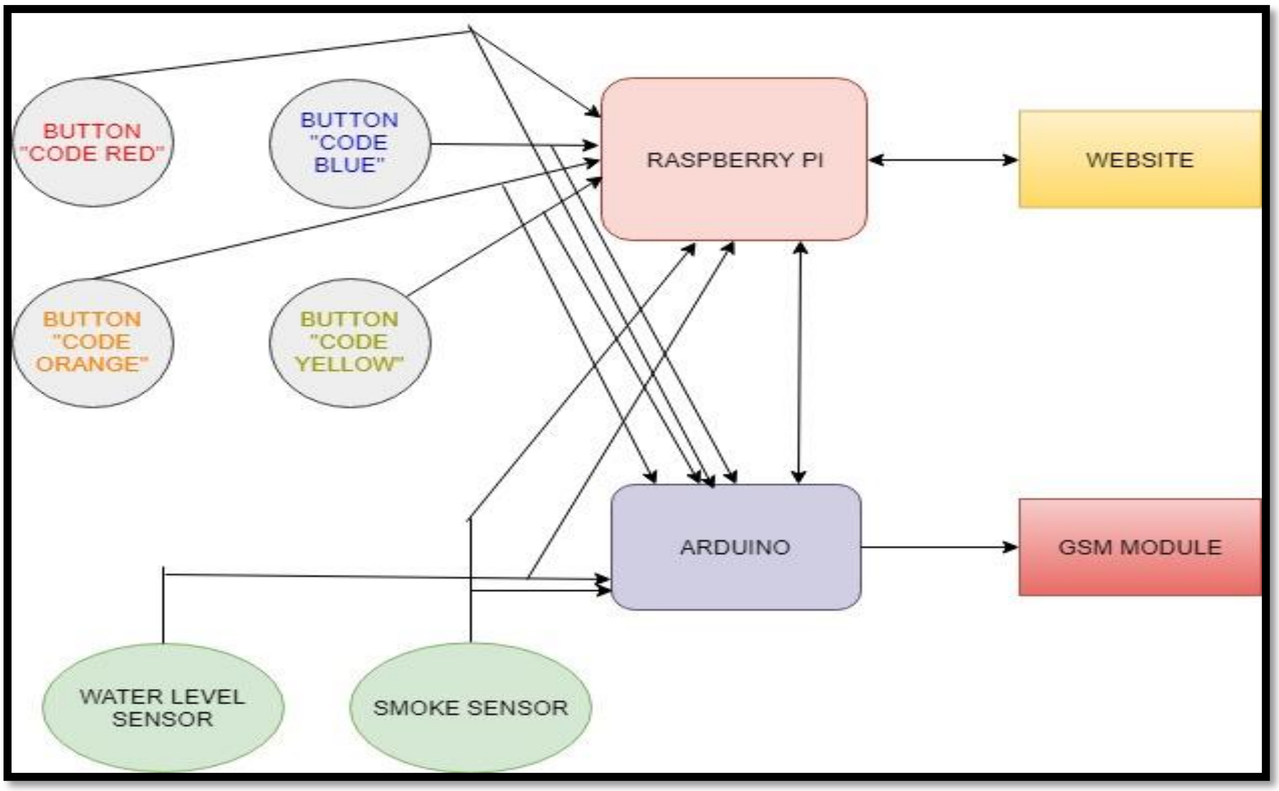
6. Analogue to Digital converter

In electronics, an analog-to-digital converter is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number representing the magnitude of the voltage or current.

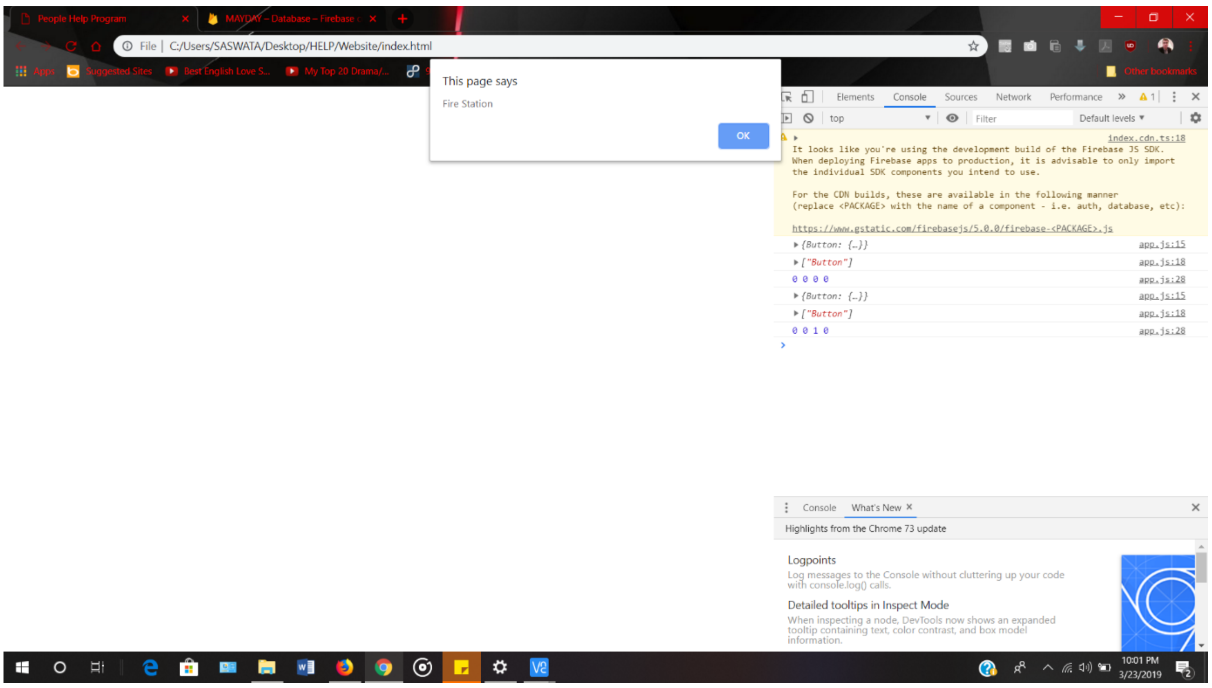




Architectural Diagram



Circuit Diagram



This page says

Fire Station

```
index.cdn.ts:18
It looks like you're using the development build of the Firebase JS SDK.
When deploying Firebase apps to production, it is advisable to only import
the individual SDK components you intend to use.

For the CDN builds, these are available in the following manner
(replace <PACKAGE> with the name of a component - i.e. auth, database, etc):

https://www.gstatic.com/firebasejs/5.0.0/firebase-<PACKAGE>.js
▸ {Button: {…}}                                              app.js:15
▸ ["Button"]                                                 app.js:18
0 0 0 0                                                      app.js:28
▸ {Button: {…}}                                              app.js:15
▸ ["Button"]                                                 app.js:18
0 0 1 0                                                      app.js:28
>
```

Highlights from the Chrome 73 update

Logpoints

Log messages to the Console without cluttering up your code with console.log() calls.

Detailed tooltips in Inspect Mode

When inspecting a node, DevTools now shows an expanded tooltip containing text, color contrast, and box model information.

Website

RESULT AND DISCUSSIONS:

The results of this project include the working of the GSM module, even though we faced a bit of difficulty at first, the water level and smoke sensors, the buzzer, buttons in unison with our own website using Google's firebase. We were able to achieve the messaging and website display functionality in our project.

Objectives Met:

1. Website display
2. Messaging Function
3. Raspberry Pi and Arduino running simultaneously

Objectives Not-Met:

1. The whole module to be activated and function by voice.

We had to re-configure the connections of the Arduino and Raspberry pi to so as to get a module working unison with the website.

We had to get a local GSM Subscriber Identification Module so as to remove the delay in messaging.

We connected the sensors to both Arduino and Raspberry Pi using an ADC so that both messaging and website display could be activated at once.

CONCLUSION AND FUTURE SCOPE

The present study has been made to suggest and develop some tools which will eventually be useful to hospitals, schools, MNC's, educational institutes, homes and any other space with human habitation to provide an emergency response system at a reasonable cost and of a specified quality. This project is targeted towards financing, designing, implementing and operating emergency response facilities and services that were traditionally expensive and complicated yet with many loopholes. This project aims to provide a cost effective emergency response system to everyone ranging from homes to large MNC's. As this project is simple to operate anyone with some basic knowledge of computers can operate this. Also because the project is not dependent on the internet the system will be always active independent of the internet and the fact that the system can run on any energy source (requires very low energy) largely reduces maintenance. As raspberry pi boards and components are the main building blocks of this system, this product is highly cost-effective and easy to mass produce.

That being said the current prototype can be highly improved in the coming years with more R&D.

- Integration of voice recognition
- Enable user to customize the functioning of the product as per requirement
- Integrating tele communication
- Installing a display which shows present status
- Developing an application for the device
- Cooperating with all private security firms
- If integrated with a moving object, live location will also be sent.

REFERENCES

- https://en.wikipedia.org/wiki/Raspberry_Pi
- https://www.youtube.com/watch?v=wJgyszOSoQU
- https://www.rototron.info/raspberry-pi-analog-water-sensor-tutorial/
- http://www.learningaboutelectronics.com/Articles/MQ-2-smoke-sensor-circuit-with-raspberry-pi.php
- https://www.raspberrypi.org/forums/viewtopic.php?t=143301
- https://maker.pro/raspberry-pi/tutorial/how-to-connect-and-interface-raspberry-pi-with-arduino

APPENDIX

1).
```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(9,10);//RX,TX

const int buttonOne=7;
const int buttonTwo=6;
const int buttonThree=5;
const int buttonFour=4;


const int read = A0; //Sensor AO pin to Arduino pin A0
int value;        //Variable to store the incomming data


int buttonLastState1=0;
int buttonLastState2=0;
int buttonLastState3=0;
int buttonLastState4=0;
int buttonPushCounter1=0;
int buttonPushCounter2=0;
int buttonPushCounter3=0;
int buttonPushCounter4=0;
int buttonState1=0;
int buttonState2=0;
int buttonState3=0;
int buttonState4=0;


void setup() {
  mySerial.begin(9600);
  pinMode(buttonOne, INPUT);
  pinMode(buttonTwo, INPUT);
  pinMode(buttonThree, INPUT);
  pinMode(buttonFour, INPUT);
  Serial.begin(9600);
}

void loop() {
  buttonState1 = digitalRead(buttonOne);
  buttonState2 = digitalRead(buttonTwo);
  buttonState3 = digitalRead(buttonThree);
  buttonState4 = digitalRead(buttonFour);

  if(buttonState1 != buttonLastState1) {//WORKING ON BUTTON 2 FOR GSM CODE BLUE
    if(buttonState1 == HIGH) {
      buttonPushCounter1++;
```

```
     mySerial.println("AT+CMGS=\"+918013984974\"\r");
     delay(700);
     mySerial.println("Code Blue!");
     delay(100);
     mySerial.println((char)26);
     delay(700);
  } else {
    delay(10);
  }
  delay(20);
}
if(buttonState2 != buttonLastState2) {//WORKING ON BUTTON 2 FOR GSM CODE BLUE
  if(buttonState2 == HIGH) {
     buttonPushCounter2++;
     mySerial.println("AT+CMGS=\"+919641805409\"\r");
     delay(700);
     mySerial.println("Code Red!");
     delay(100);
     mySerial.println((char)26);
     delay(700);
  }
  else {
  delay(10);
  }
  delay(20);
}
if(buttonState3 != buttonLastState3) {//WORKING ON BUTTON 2 FOR GSM CODE BLUE
  if(buttonState3 == HIGH) {
    buttonPushCounter3++;
     mySerial.println("AT+CMGS=\"+91790106620\"\r");
     delay(700);
     mySerial.println("Code Yellow!");
     delay(100);
     mySerial.println((char)26);
     delay(700);
  } else {
    delay(10);
  }
  delay(20);
}
if(buttonState4 != buttonLastState4) {//WORKING ON BUTTON 2 FOR GSM CODE BLUE
  if(buttonState4 == HIGH) {
    buttonPushCounter4++;
     mySerial.println("AT+CMGS=\"+916296223496\"\r");
     delay(700);
     mySerial.println("Code Orange!");
     delay(100);
     mySerial.println((char)26);
     delay(700);
  } else {
    delay(10);
  }
  delay(20);
}

int sensorValue = analogRead(A0);
Serial.print(sensorValue);
if( sensorValue>700){
  mySerial.println("AT+CMGS=\"+918013984974\"\r");
  delay(700);
  mySerial.println("smoke detected");
  delay(100);
  mySerial.println((char)26);
  delay(5000);
}


value = analogRead(read); //Read data from analog pin and store it to value variable

if (value<=480){
  Serial.println("Water level: 0mm");
}
else if (value>480 && value<=530){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 0mm to 5mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
else if (value>530 && value<=615){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 5mm to 10mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
else if (value>615 && value<=660){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 10mm to 15mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
```

```
else if (value>660 && value<=680){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 15mm to 20mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
else if (value>680 && value<=690){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 20mm to 25mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
else if (value>690 && value<=700){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 25mm to 30mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
else if (value>700 && value<=705){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 30mm to 35mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}
else if (value>705){
  mySerial.println("AT+CMGS=\"+917901066620\"\r");
     delay(700);
     mySerial.println("Water level: 35mm to 40mm");
     delay(100);
     mySerial.println((char)26);
     delay(700);
}

delay(5000); // Check for new value every 5 sec


buttonLastState1=buttonState1;
buttonLastState2=buttonState2;
buttonLastState3=buttonState3;
```

```
  buttonLastState4=buttonState4;
}
```

2.) import RPi.GPIO as GPIO

```
import time



# change these as desired - they're the pins connected from the
# SPI port on the ADC to the Cobbler
SPICLK = 11
SPIMISO = 9
SPIMOSI = 10
SPICS = 8

# photoresistor connected to adc #0
photo_ch = 0

#port init
def init():
        GPIO.setwarnings(False)
        GPIO.cleanup()                       #clean up at the end of your script
        GPIO.setmode(GPIO.BCM)               #to specify whilch pin numbering system
        # set up the SPI interface pins
        GPIO.setup(SPIMOSI, GPIO.OUT)
        GPIO.setup(SPIMISO, GPIO.IN)
        GPIO.setup(SPICLK, GPIO.OUT)
```

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

#read SPI data from MCP3008(or MCP3204) chip,8 possible adc's (0 thru 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin, True)

        GPIO.output(clockpin, False)  # start clock low
        GPIO.output(cspin, False)     # bring CS low

        commandout = adcnum
        commandout |= 0x18  # start bit + single-ended bit
        commandout <<= 3    # we only need to send 5 bits here
        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True)
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True)
                GPIO.output(clockpin, False)

        adcout = 0
        # read in one empty bit, one null bit and 10 ADC bits
        for i in range(12):
                GPIO.output(clockpin, True)
```

```
            GPIO.output(clockpin, False)
            adcout <<= 1
            if (GPIO.input(misopin)):
                adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1       # first bit is 'null' so drop it
        return adcout

def main():
         init()
         time.sleep(1)
         print"will start detec water level\n"
         while True:
                  adc_value=readadc(photo_ch, SPICLK, SPIMOSI, SPIMISO, SPICS)
                  if adc_value == 0:
                           print"no water\n"
                  elif adc_value>0 and adc_value<20 :
                           print"it is raindrop\n"
                  elif adc_value>=25 and adc_value<250 :
                           print"it is water flow"
                           print"water level:"+str("%.1f"%(adc_value/200.*100))+"%\n"
                  #print "adc_value= " +str(adc_value)+"\n"
                  time.sleep(1)
```

```
if __name__ == '__main__':
        try:
                main()

        except KeyboardInterrupt:
                pass
GPIO.cleanup()
```

3). import RPi.GPIO as GPIO #import Raspberry Pi GPIO library

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
cred = credentials.Certificate(r"'/home/pi/Desktop/mayday-saddy-firebase-adminsdk-jjbng-bd4e2e4263.json"')
firebase_admin.initialize_app(cred, {
    'databaseURL' : 'https://mayday-saddy.firebaseio.com/'
})
ref = db.reference()
users_ref = ref.child('Button')

def button_callback1(channel):
    ref = db.reference()
    users_ref = ref.child('Button')
    users_ref.update({'Button1': 1})
    print("A")
    users_ref.update({'Button1': 0})
```

```
def button_callback2(channel):

    ref = db.reference()

    users_ref = ref.child('Button')

    users_ref.update({'Button2': 1})

    print("B")

    users_ref.update({'Button2': 0})



def button_callback3(channel):

    ref = db.reference()

    users_ref = ref.child('Button')

    users_ref.update({'Button3': 1})

    print("C")

    users_ref.update({'Button3': 0})


def button_callback4(channel):

    ref = db.reference()

    users_ref = ref.child('Button')

    users_ref.update({'Button4': 1})

    print("D")

    users_ref.update({'Button4': 0})



GPIO.setwarnings(False) # Ignore warning for now

GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(31, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Set pin 10 to be an input pin and set initial value to be pulled low (off)

GPIO.setup(33, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(35, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(37, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)



GPIO.add_event_detect(31,GPIO.RISING,callback=button_callback1)# Setup event on pin 10 rising edge

GPIO.add_event_detect(33,GPIO.RISING,callback=button_callback2)

GPIO.add_event_detect(35,GPIO.RISING,callback=button_callback3)

GPIO.add_event_detect(37,GPIO.RISING,callback=button_callback4)



message = input("Press enter to quit") # Run until someone presses enter


print ('SUCCESS')



GPIO.cleanup() # Clean up
```