Report On

IoT-Based Smart Ambulance

Prepared
By
Group 5 - ECE
Adarsh Nayak – 211010205 - adarsh21101@iiitnr.edu.in
Anshul Gupta – 211010216 - anshul21101@iiitnr.edu.in
Aaryansh Verma – 211010218 – aaryansh21101@iiitnr.edu.in

Submitted

To
Dr. Debanjan Das
Assistant Professor, IIIT-NR

Course Name: Introduction to IoT

Dr. Shyama Prasad Mukherjee
International Institute of Information Technology, Naya Raipur
(A Joint Initiative of Govt. of Chhattisgarh and NTPC)
Email: iiitnr@iiitnr.edu.in Tel: (0771)2474040 Web: www.iiitnr.ac.in

IoT-Based Smart Ambulance

B. tech 1st year (2021-2025), Electronics and Communication Engineering,

IIIT Naya Raipur, Chhattisgarh, India

Adarsh Nayak
ECE
211010205
adarsh21101@iiitnr.edu.in

Anshul Gupta
ECE
211010216
anshul21101@iiitnr.edu.in

Aaryansh Verma
ECE
211010218
aaryansh21101@iiitnr.edu.in

Abstract- A smart Ambulance is developed using Internet of Things (IoT) technology which is capable of monitoring a person's temperature, heartbeat, and blood pressure. The Model is helpful for rural areas or villages where nearby clinics can be in touch with city hospitals about the patient's health conditions. If any changes occur in a patient's health condition based on typical values, this Model will give an alert message to the doctor. This allows doctors to collect real-time data effortlessly. Provision of interactive real-time multimedia communication, real-time location tracking, etc., have also been integrated into the proposed system to monitor the exact condition on a real-time basis. The system prototype has been designed with the Raspberry Pi 3, Arduino Uno, GPS, and Thingspeak. It predicts the patient's criticality using actual sensor data. When the sensed parameters exceed the pre-set threshold in the rule-base, it initiates data transfer to the fog or cloud server. If fog or the cloud is unreachable, it performs onboard predictions. Thus, the framework ensures essential service delivery to the user.

Keywords: IoT, Smart Ambulance, Sensor interfacing, Cloud platform, Raspberry Pi 3, real-time location tracking, mobile patient monitoring

Introduction

We are living in the Internet era and rapidly moving towards an intelligent planet where every device will be connected. Internet of Things (IoT) is the technology supporting us to achieve the goal of creating a smart world. IoT and Cyber-Physical systems can change the vision of our way of living. There is an essential need for advancement in the health sector, like merging the new technologies in every step involved in the health sector. These would help provide good treatment to the patient and reduce one's life.

A. Motivation- With the associated enabling technologies, it is now possible to monitor and provide life-saving suggestions for

critical patients in transit or a sick person away from a medical facility.

In addition, the paradigm shift in computing to meet the low latency demands and lower bandwidth usage has led to the introduction of edge and fog computing.

The proposed framework is generic and is readily applicable for service delivery in diverse IoT systems. The noticeable traits of our proposed framework are as follows:

1.Internet Connection And On-board Prediction: - It stores pre-trained models on the edge device that are only triggered when the Internet is unavailable.

2.Prediction Accuracy Enhancement: - The proposed framework aims to provide a highly accurate prediction.

B. Related Work And Research Motivation-

Researchers have carried out work for monitoring the health parameters of patients through various sensors. Work on IoT and ML-based detection of diseases was done like mobile dental care system where the mobile application captured images of patients' teeth, and the pictures were fed to the DL models to categorize the images into seven classes. A CNN-based algorithm for the detection of "Myocardial Infraction" using ECG signals was developed. The algorithm was tested in "MATLAB" and "ARM Cortex-A9" based systems in real-time and achieved 96.00% accuracy. There has also been worked on the real-time location tracker of a patient going through an ambulance. Further, a Cloud, Fog, and Edge Based system was proposed to detect episodes of epilepsy. The computationally intensive tasks were offloaded to Fog or Cloud servers to reduce energy consumption by the battery-powered wearable. The researchers aimed at making the devices "Self-Aware" that considered the complexity of the tasks. Task distribution was done based on energy requirement and the delay incurred. however, they have not considered the low level or no network region, and they were not sending continuous real-time data of the patient to the consulting doctor and finding the nearest hospital in case of emergency

C. Contribution-

We propose a system that would provide rural residents with the necessary health monitoring and information about their location and ongoing health status that would aid doctors in making decisions in an emergency. The key contribution is: -

1. We proposed sensors like ECG Heart Monitoring, Temperature to get patient health.
2. We proposed a cloud to store real-time data, which will be available to the visiting doctor.
3. We proposed a GPS tracker to get the real-time location of the patient.
4. We proposed an ML model to predict the patient health parameter during no network area.
5. We proposed a model which would find a nearby hospital in case of any emergency.

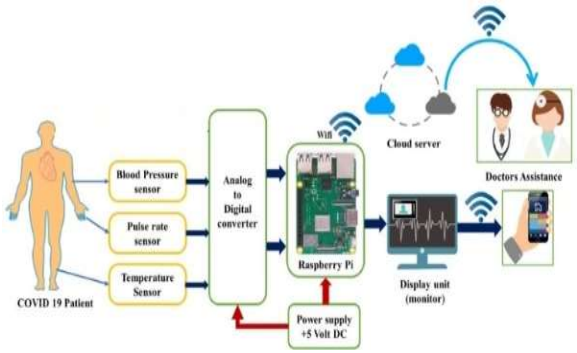
OBJECTIVE

1. To measure the temperature, heartbeat, and blood pressure of the patient.
2. Send real-time data to the consulting doctor about the patient health parameter.
3. To track the actual time and location of the patient and find the nearest hospital in case of emergency.

4. To store the patient Health data in no network region, and whenever the network is achieved, the health data and the predicted data are sent to the doctor.

Proposed Methodology

Conceptual Diagram



System Description

1. AD8232 ECG Sensor- The AD8232 ECG sensor is used to calculate the electrical activity of the human heart. This action can be a chart as an Electrocardiogram and provides an analog reading. Electrocardiograms are noisy, so the AD8232 chip can be used to reduce noise. The ECG sensor operating principle is similar to the operational amplifier to help get a clear signal from time to time quickly.

2.Pulse Sensor- This sensor collects the heartbeat. This sensor clips on a fingertip and is connected to an Arduino board through jumper wires.

3.DS18B20 Temperature sensor- The DS18B20 is a 1-wire temperature sensor. It measures the temperature of objects in Tough environments.

4.Ardunio Uno- We used Arduino Uno to collect sensor data and used it as an Analog-digital converter.

5.Raspberry Pi 3- We have used raspberry pi to collect data from the sensor from Arduino Uno and push it to the cloud.

7.Phone GPS-

It collects the **latitude, longitude, and speed** of the ambulance and plots the graph for the above parameters.

Result

The effective functioning of any proposed system can be measured by its accuracy and efficacy. To ensure the proposed system is functional, it has been tested rigorously under various conditions. While we cannot use the prototype on real patient as it can cause any issue to his/her health, we have put the prototype through a series of tests that can ensure its working and feasibility in a real-world scenario.

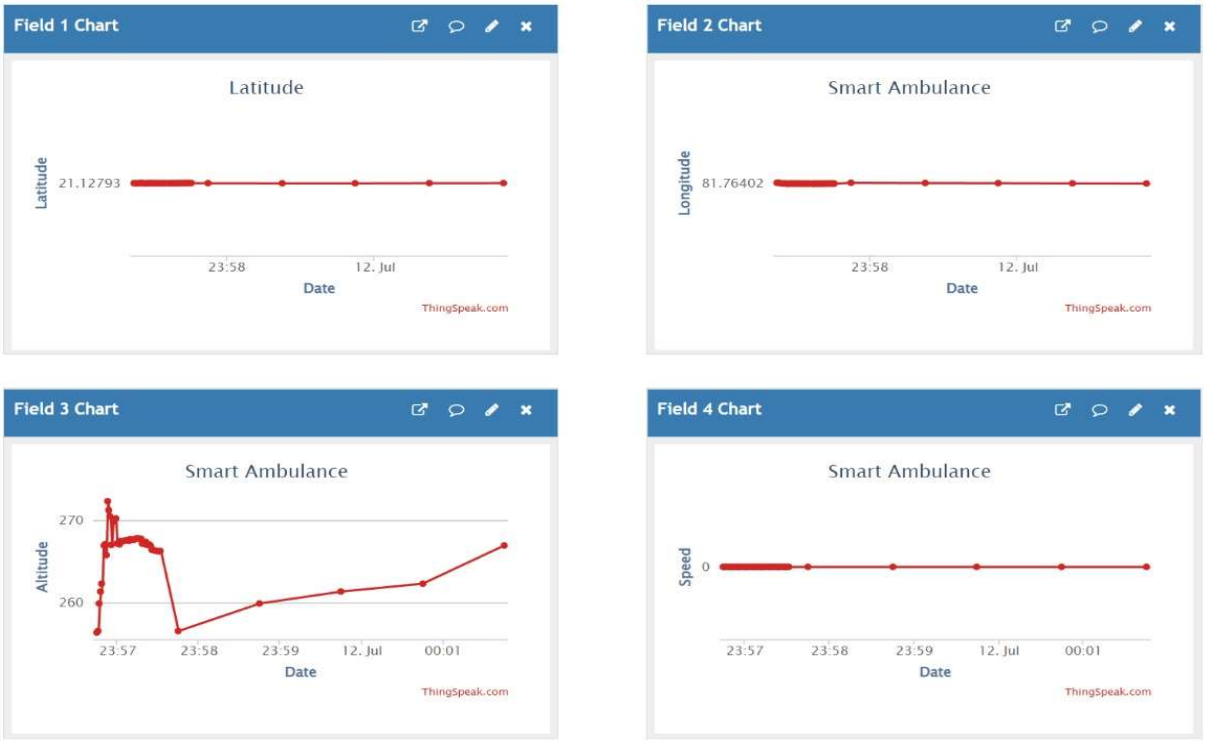


Fig. Location data of ambulance

Fig. health data of patient

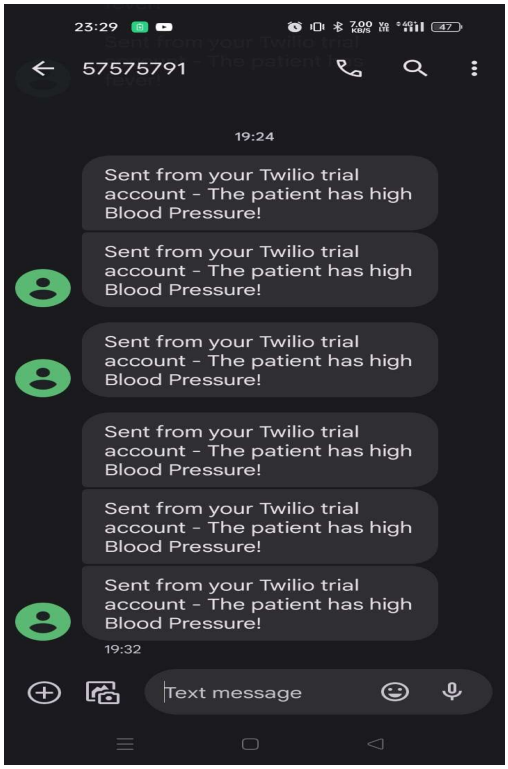


Fig. Message alert

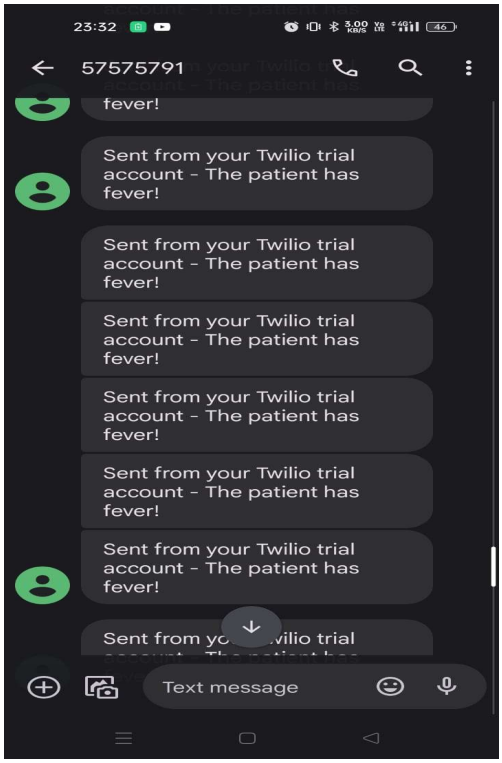
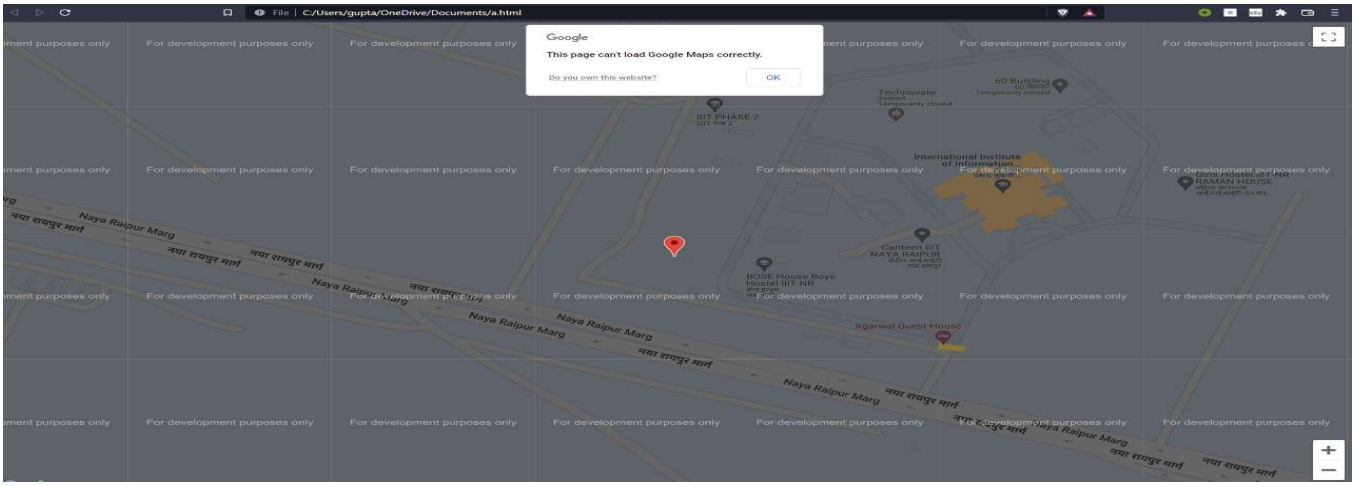
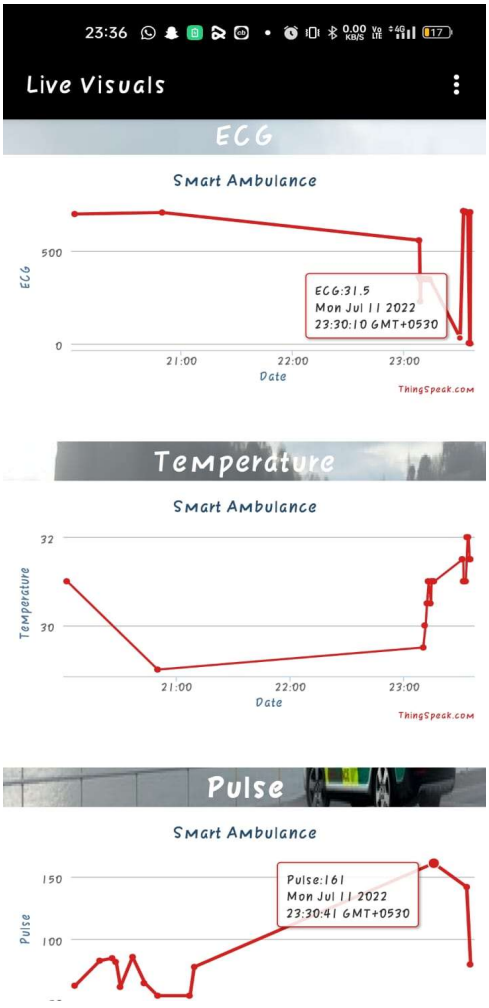
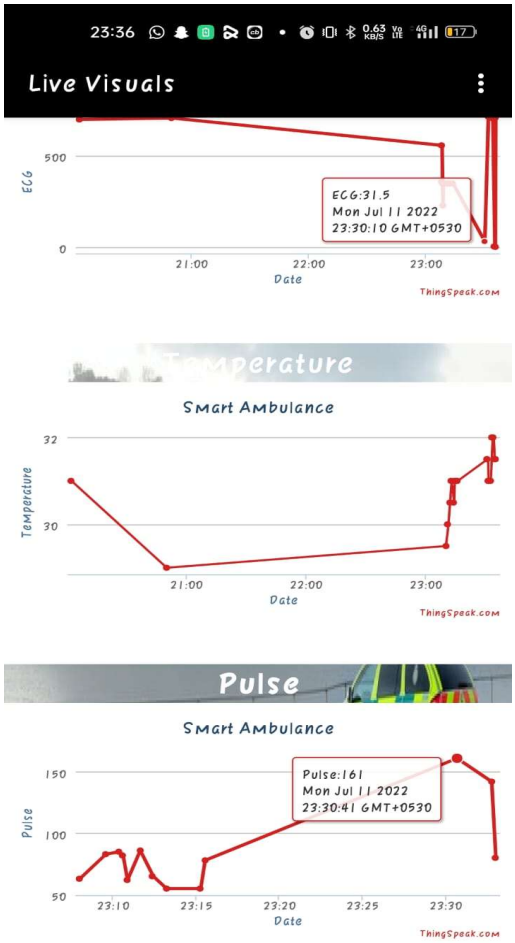


Fig Message alert

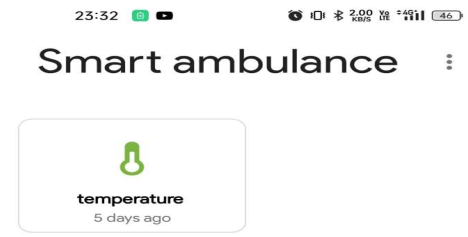


Above Fig. Location interface





Above Figure. Mobile interface for patient and ambulance data



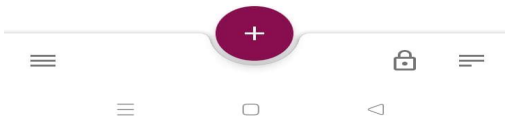


Fig Alternate Mobile interface

Comparison with existing solutions

The existing solution does not provide the proper solution to the problem. The Existing solution used a few health parameters through sensors and is there a way to provide immediate medical attention in

case of mishaps. A better solution has been proposed in an International Conference on Communication, Management and Information Technology (ICCMIT) paper titled "Ambulatory Health Monitoring System Using Wireless Sensors Node

". Our solution differs from the one provided in the research paper mentioned above paper

1 Our prototype has three sensors constantly checking a patient's health parameters: temperature, heartbeat, and blood pressure.

2 Our prototype has provided the actual-time location of the patient

3. Our prototype pushes the data to the cloud and helps the doctor continuously get the patient's real-time data.

4. Our prototype provides an alert message in case of any emergency and finds any nearby hospital

5. Our prototype store the data in no network region and publishes it whenever the network is achieved

Conclusion

Health is the most critical thing in-person life. People in rural areas lack to find the right or good facilities for their health check-ups. They often cannot find an ambulance or a nearby hospital in an emergency. This model will help people to get the correct health parameter check-up and find the nearest hospitals in case of any emergency. It also helps the consulting doctor to collect real-time patient data.

Learning Outcome

Making this project pushed us to expand our knowledge. Some of our new learnings are given
below.

- Learned to work with Raspberry Pi 3 and Arduino UNO.
- Learned to use Arduino IDE to upload code and Vnc Viewer, putty to run Raspberry pi 3

- Learned to make and test prototypes.
- Learned the basics about ECG Sensor, pulse sensor, DS18B20 Temperature sensor
- Learned to push and receive data to and from the cloud, i.e., ThingSpeak using Raspberry Pi 3
- Learned to send data from Arduino to Raspberry Pi 3
- Learned about the working conditions and safety practices on construction sites.
- Learned to track and plot the real-time location using phone GPS.

Future Scope

While our prototype successfully performs the tasks it aims to accomplish, there is a scope for further improvement.

1. We can use more sensors to get other health parameters of the patient
2. We can increase the accuracy of ML prediction of patient health parameters
3. We can also integrate a model which will tell the future health parameter of 1 or 2 hours to draw better inferences

References

1. DS18B20 (Digital Temperature Sensor) and Arduino, https://create.arduino.cc/projecthub/TheGadgetBoy/ds18b20-digital-temperature-sensor-and-arduino-9cc806
2. Arduino Based ECG & Heartbeat Monitoring Healthcare System, https://www.instructables.com/ECG-Monitoring-System-by-Using-Arduino-or-AD8232/
3. Pulse Rate (BPM) Monitor using Arduino & Pulse Sensor, https://create.arduino.cc/projecthub/abiodun1991/developing-a-device-for-heart-rate-using-pulse-sensor-22746c
4. Sending Arduino uno data to raspberry pi 3 Model, https://youtu.be/OJtpA_qTNL0
5. Sending raspberry pi 3 data to cloud, https://youtu.be/-IAlB_CgUyE
6. IoT Analytics - ThingSpeak Internet of Things https://thingspeak.com/
7. International Conference on Communication, Management and Information Technology (ICCMIT) "Ambulatory Health Monitoring System Using Wireless Sensors Node"
8. https://www.google.com/search?q=integrate+mobile+gps+with+raspberry+pi&oq=integrate+mobile+gps+with+r&aqs=chrome.1.69i57j33i160l5j33i22i29i30l4.19230j0j7&sourceid=chrome&ie=UTF-8#kpvalbx=_CTOsYoS3I6CP4-EPmfCn4AY33
9. https://www.researchgate.net/publication/317616307_An_IoT_Enabled_Real-Time_Communication_and_Location_Tracking_System_for_Vehicular_Emergency
10. https://pubmed.ncbi.nlm.nih.gov/35079705/#:~:text=A%20smart%20health%20monitoring%20system,and%20temperature%20of%20a%20person
11. https://www.raspberrypi.com/software/
12. https://www.arduino.cc/en/software

https://create.arduino.cc/projecthub/TheGadgetBoy/ds18b20-digital-temperature-sensor-and-arduino-9cc806

Source Code

1.Sensor to Arduino and to Raspberry pi

```
// Include the libraries we need
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged into port 2 on the Arduino
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

// arrays to hold device address
DeviceAddress insideThermometer;

/*
 * Setup function. Here we do the basics
 */
 //code copied from arduino.cc
int pulsePin = A0;                 // Pulse Sensor purple wire connected to analog pin A0
int blinkPin = 13;                // pin to blink led at each beat

// Volatile Variables, used in the interrupt service routine!
```

```
volatile int BPM;                   // int that holds raw Analog in 0. updated every 2mS
volatile int Signal;                // holds the incoming raw data
volatile int IBI = 600;             // int that holds the time interval between beats! Must be seeded!
volatile boolean Pulse = false;     // "True" when User's live heartbeat is detected. "False" when not a "live beat".
volatile boolean QS = false;        // becomes true when Arduoino finds a beat.

static boolean serialVisual = true;   // Set to 'false' by Default.  Re-set to 'true' to see Arduino Serial Monitor ASCII Visual Pulse

volatile int rate[10];                    // array to hold last ten IBI values
volatile unsigned long sampleCounter = 0;          // used to determine pulse timing
volatile unsigned long lastBeatTime = 0;           // used to find IBI
volatile int P = 512;                     // used to find peak in pulse wave, seeded
volatile int T = 512;                     // used to find trough in pulse wave, seeded
volatile int thresh = 525;                // used to find instant moment of heart beat, seeded
volatile int amp = 100;                   // used to hold amplitude of pulse waveform, seeded
volatile boolean firstBeat = true;        // used to seed rate array so we startup with reasonable BPM
volatile boolean secondBeat = false;      // used to seed rate array so we startup with reasonable BPM

void setup(void)
{  pinMode(10,INPUT);
  pinMode(11,INPUT);

  // start serial port

  Serial.println("Dallas Temperature IC Control Library Demo");
```

```
  // locate devices on the bus
  Serial.print("Locating devices...");
  sensors.begin();
  Serial.print("Found ");
  Serial.print(sensors.getDeviceCount(), DEC);
  Serial.println(" devices.");

  // report parasite power requirements
  Serial.print("Parasite power is: ");
  if (sensors.isParasitePowerMode()) Serial.println("ON");
  else Serial.println("OFF");

  // Assign address manually. The addresses below will beed to be changed
  // to valid device addresses on your bus. Device address can be retrieved
  // by using either oneWire.search(deviceAddress) or individually via
  // sensors.getAddress(deviceAddress, index)
  // Note that you will need to use your specific address here
  //insideThermometer = { 0x28, 0x1D, 0x39, 0x31, 0x2, 0x0, 0x0, 0xF0 };

  // Method 1:
  // Search for devices on the bus and assign based on an index. Ideally,
  // you would do this to initially discover addresses on the bus and then
  // use those addresses and manually assign them (see above) once you know
  // the devices on your bus (and assuming they don't change).
  if (!sensors.getAddress(insideThermometer, 0)) Serial.println("Unable to find address for
Device 0");

  // method 2: search()
  // search() looks for the next device. Returns 1 if a new address has been
```

```
  // returned. A zero might mean that the bus is shorted, there are no devices,
  // or you have already retrieved all of them. It might be a good idea to
  // check the CRC to make sure you didn't get garbage. The order is
  // deterministic. You will always get the same devices in the same order
  //
  // Must be called before search()
  //oneWire.reset_search();
  // assigns the first address found to insideThermometer
  //if (!oneWire.search(insideThermometer)) Serial.println("Unable to find address for insideThermometer");

  // show the addresses we found on the bus
  Serial.print("Device 0 Address: ");
  printAddress(insideThermometer);
  Serial.println();

  // set the resolution to 9 bit (Each Dallas/Maxim device is capable of several different resolutions)
  sensors.setResolution(insideThermometer, 9);

  Serial.print("Device 0 Resolution: ");
  Serial.print(sensors.getResolution(insideThermometer), DEC);
  Serial.println();

  pinMode(blinkPin,OUTPUT);         // pin that will blink to your heartbeat!
  Serial.begin(115200);             // we agree to talk fast!
  interruptSetup();                 // sets up to read Pulse Sensor signal every 2mS
                                    // IF YOU ARE POWERING The Pulse Sensor AT VOLTAGE LESS THAN THE BOARD VOLTAGE,
```

```
                        // UN-COMMENT THE NEXT LINE AND APPLY THAT VOLTAGE TO THE A-REF PIN
                        //   analogReference(EXTERNAL);


}
float t;
int Bp,ECG;
// function to print the temperature for a device
void printTemperature(DeviceAddress deviceAddress)
{
  // method 1 - slower
  //Serial.print("Temp C: ");
  //Serial.print(sensors.getTempC(deviceAddress));
  //Serial.print(" Temp F: ");
  //Serial.print(sensors.getTempF(deviceAddress)); // Makes a second call to getTempC and then converts to Fahrenheit

  // method 2 - faster
  float tempC = sensors.getTempC(deviceAddress);
  t=tempC;
  if(tempC == DEVICE_DISCONNECTED_C)
  {
    Serial.println("Error: Could not read temperature data");
    return;
  }}
/*
 * Main function. It will request the tempC from the sensors and display on Serial.
 */
```

```
// function to print a device address
void printAddress(DeviceAddress deviceAddress)
{


}
void interruptSetup()
{
  // Initializes Timer2 to throw an interrupt every 2mS.
  TCCR2A = 0x02;     // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO INTO CTC MODE
  TCCR2B = 0x06;     // DON'T FORCE COMPARE, 256 PRESCALER
  OCR2A = 0X7C;      // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE RATE
  TIMSK2 = 0x02;     // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2 AND OCR2A
  sei();             // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED
}


void serialOutput()
{   // Decide How To Output Serial.
 if (serialVisual == true)
  {
    arduinoSerialMonitorVisual('-', Signal);   // goes to function that makes Serial Monitor Visualizer
  }
 else
  {
     sendDataToSerial('S', Signal);     // goes to sendDataToSerial function
   }
}
```

```
void serialOutputWhenBeatHappens()
{
 if (serialVisual == true) //  Code to Make the Serial Monitor Visualizer Work
   {
      //ASCII Art Madness
    Bp=BPM;
     }
 else
   {
     sendDataToSerial('B',BPM);   // send heart rate with a 'B' prefix
     sendDataToSerial('Q',IBI);   // send time between beats with a 'Q' prefix
   }
}

void arduinoSerialMonitorVisual(char symbol, int data )
{
  const int sensorMin = 0;      // sensor minimum, discovered through experiment
  const int sensorMax = 1024;    // sensor maximum, discovered through experiment
  int sensorReading = data; // map the sensor range to a range of 12 options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 11);
  // do something different depending on the
  // range value:
}
void loop(void)
{
  // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
```

```
  sensors.requestTemperatures(); // Send the command to get temperatures


  // It responds almost immediately. Let's print out the data
  printTemperature(insideThermometer); // Use a simple function to print out the data
  serialOutput();

  if (QS == true) // A Heartbeat Was Found
    {
      // BPM and IBI have been Determined
      // Quantified Self "QS" true when arduino finds a heartbeat
      serialOutputWhenBeatHappens(); // A Beat Happened, Output that to serial.
      QS = false; // reset the Quantified Self flag for next time
    }

  if((digitalRead(10)==1)||(digitalRead(11)==1)){
  }
  else{ECG=analogRead(A1);
      }
  delay(100);

  delay(20); //  take a break
if(t>0&&Bp>0&&ECG>0)
{  delay(100);


  Serial.println(t);
```

```
Serial.println(Bp);
Serial.println(ECG);
  delay(20);}
}


void sendDataToSerial(char symbol, int data )
{ }

ISR(TIMER2_COMPA_vect) //triggered when Timer2 counts to 124
{
  cli();                                      // disable interrupts while we do this
  Signal = analogRead(pulsePin);              // read the Pulse Sensor
  sampleCounter += 2;                         // keep track of the time in mS with this variable
  int N = sampleCounter - lastBeatTime;       // monitor the time since the last beat to avoid noise
                                              //  find the peak and trough of the pulse wave
  if(Signal < thresh && N > (IBI/5)*3) // avoid dichrotic noise by waiting 3/5 of last IBI
    {
      if (Signal < T) // T is the trough
      {
        T = Signal; // keep track of lowest point in pulse wave
      }
    }

  if(Signal > thresh && Signal > P)
    {          // thresh condition helps avoid noise
      P = Signal;                             // P is the peak
    }                                         // keep track of highest point in pulse wave
```

```
//  NOW IT'S TIME TO LOOK FOR THE HEART BEAT
// signal surges up in value every time there is a pulse
if (N > 250)
{                                   // avoid high frequency noise
  if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) )
    {
      Pulse = true;                               // set the Pulse flag when we think there is a pulse
      digitalWrite(blinkPin,HIGH);                // turn on pin 13 LED
      IBI = sampleCounter - lastBeatTime;         // measure time between beats in mS
      lastBeatTime = sampleCounter;               // keep track of time for next pulse

      if(secondBeat)
      {                     // if this is the second beat, if secondBeat == TRUE
        secondBeat = false;                 // clear secondBeat flag
        for(int i=0; i<=9; i++) // seed the running total to get a realisitic BPM at startup
        {
          rate[i] = IBI;
        }
      }

      if(firstBeat) // if it's the first time we found a beat, if firstBeat == TRUE
      {
        firstBeat = false;                  // clear firstBeat flag
        secondBeat = true;                  // set the second beat flag
        sei();                              // enable interrupts again
        return;                             // IBI value is unreliable so discard it
      }
```

```
    // keep a running total of the last 10 IBI values
    word runningTotal = 0;                  // clear the runningTotal variable

    for(int i=0; i<=8; i++)
      {                // shift data in the rate array
        rate[i] = rate[i+1];                // and drop the oldest IBI value
        runningTotal += rate[i];            // add up the 9 oldest IBI values
      }

    rate[9] = IBI;                          // add the latest IBI to the rate array
    runningTotal += rate[9];                // add the latest IBI to runningTotal
    runningTotal /= 10;                     // average the last 10 IBI values
    BPM = 60000/runningTotal;               // how many beats can fit into a minute? that's BPM!
    QS = true;                              // set Quantified Self flag
    // QS FLAG IS NOT CLEARED INSIDE THIS ISR
  }
}

if (Signal < thresh && Pulse == true)
  {   // when the values are going down, the beat is over
    digitalWrite(blinkPin,LOW);            // turn off pin 13 LED
    Pulse = false;                         // reset the Pulse flag so we can do it again
    amp = P - T;                           // get amplitude of the pulse wave
    thresh = amp/2 + T;                    // set thresh at 50% of the amplitude
    P = thresh;                            // reset these for next time
    T = thresh;
  }
```

```
 if (N > 2500)
   {                          // if 2.5 seconds go by without a beat
     thresh = 512;                          // set thresh default
     P = 512;                               // set P default
     T = 512;                               // set T default
     lastBeatTime = sampleCounter;          // bring the lastBeatTime up to date
     firstBeat = true;                      // set these to avoid noise
     secondBeat = false;                    // when we get the heartbeat back
   }

 sei();                                     // enable interrupts when youre done!
}// end isr
```

2.Raspberry pi to Thingsspeak

```
import http.client, urllib.request
import time
import serial
import os
from twilio.rest import Client
from urllib.parse import urlparse

account_sid ="AC41cda92ea4fc2f76e5253b2f265dbe98" # Put your Twilio account SID here
auth_token ="650b7f37f490358a9c8cb74a51ffa98f" # Put your auth token here
client = Client(account_sid, auth_token)
ser = serial.Serial('/dev/ttyACM0',115200)
z=0
def loop():
```

```
        x = ser.readline()
        y = ser.readline()
        u = ser.readline()
        def a():
                params = urllib.parse.urlencode({'field5': x,'key':'OHB9RP2NDJI4BO2Q'})
                headers = {"Content-type": "application/x-www-form-
urlencoded","Accept":"text/plain"}
                conn = http.client.HTTPConnection("api.thingspeak.com:80")
                try:
                        conn.request("POST", "/update", params, headers)
                        response = conn.getresponse()
                        print (x)
                        data = response.read()
                        conn.close()
                except:
                        print ("connection failed")
        a();
        def b():
                params = urllib.parse.urlencode({'field6': y,'key':'OHB9RP2NDJI4BO2Q'})
                headers = {"Content-type": "application/x-www-form-
urlencoded","Accept":"text/plain"}
                conn = http.client.HTTPConnection("api.thingspeak.com:80")
                try:
                        conn.request("POST", "/update", params, headers)
                        response = conn.getresponse()
                        print👍
                        data = response.read()
                        conn.close()
                except:
```

```
                print ("connection failed")
    b();
    def c():
        params = urllib.parse.urlencode({'field7': u,'key':'OHB9RP2NDJI4BO2Q'})
        headers = {"Content-type": "application/x-www-form-
urlencoded","Accept":"text/plain"}
        conn = http.client.HTTPConnection("api.thingspeak.com:80")
        try:
            conn.request("POST", "/update", params, headers)
            response = conn.getresponse()
            print(u)
            data = response.read()
            conn.close()
        except:
            print ("connection failed")
    c();
if _name_ == "_main_":
    while True:
        loop()
        time.sleep(1)
```

3.Mobile to ThingsSpeak

```
close all;clear all% connector on
%%
m = mobiledev; % create object
t0 = datetime; % get the start time
m.Logging = 1; % stat logging data

%%
```

```
% Copyright 2017 The MathWorks, Inc
% upload the data to ThingSpeak
[lat, lon, t, speed, course, alt, horizacc] = poslog(m); % get sensor data
while lat
   thingSpeakWrite(1754483,[lat, lon, alt,
speed],'WriteKey','OHB9RP2NDJI4BO2Q','Timestamp' ,t0+t/24/60);
end
%
% discardlogs(m)
% clear m%%
```