MECHATRONICS SYSTEMS DESIGN - MEE1045

HAPTAP – A WEARABLE HAPTICS WITH IOT FOR DISABLED

FINAL

PROJECT REPORT

Mothish Raj – 15BME1019

VIT UNIVERSITY CHENNAI

ABSTRACT

Sensory and physical impairments decrease the quality of life. In this project our goal is to create a wearable mechatronic system that enables the user to communicate through taps or gesture and get a feedback; and be monitored continuously remotely and alarm others remote when danger is detected. This can be achieved by using various sensors and haptic feed back systems connected to IOT.

The main concerned area of people are the ones with cerebral palsy and multiple impairments. With computationally powerful electronics it can be achieved.

INTRODUCTION

- **Haptic Wearables**

Haptic systems recreate a sense of tactile as a feedback from the computer in form of defined vibration using a micro vibrator. [Human – Computer Interface]

- **Sensory Augmentation**

Sensory impairment is the major issue when it comes to disabled people such as cerebral palsy and visual come ear impairment. Through haptics as feedback and accelerometer as their input communication can be achieved.

  - By tapping patterns on Device

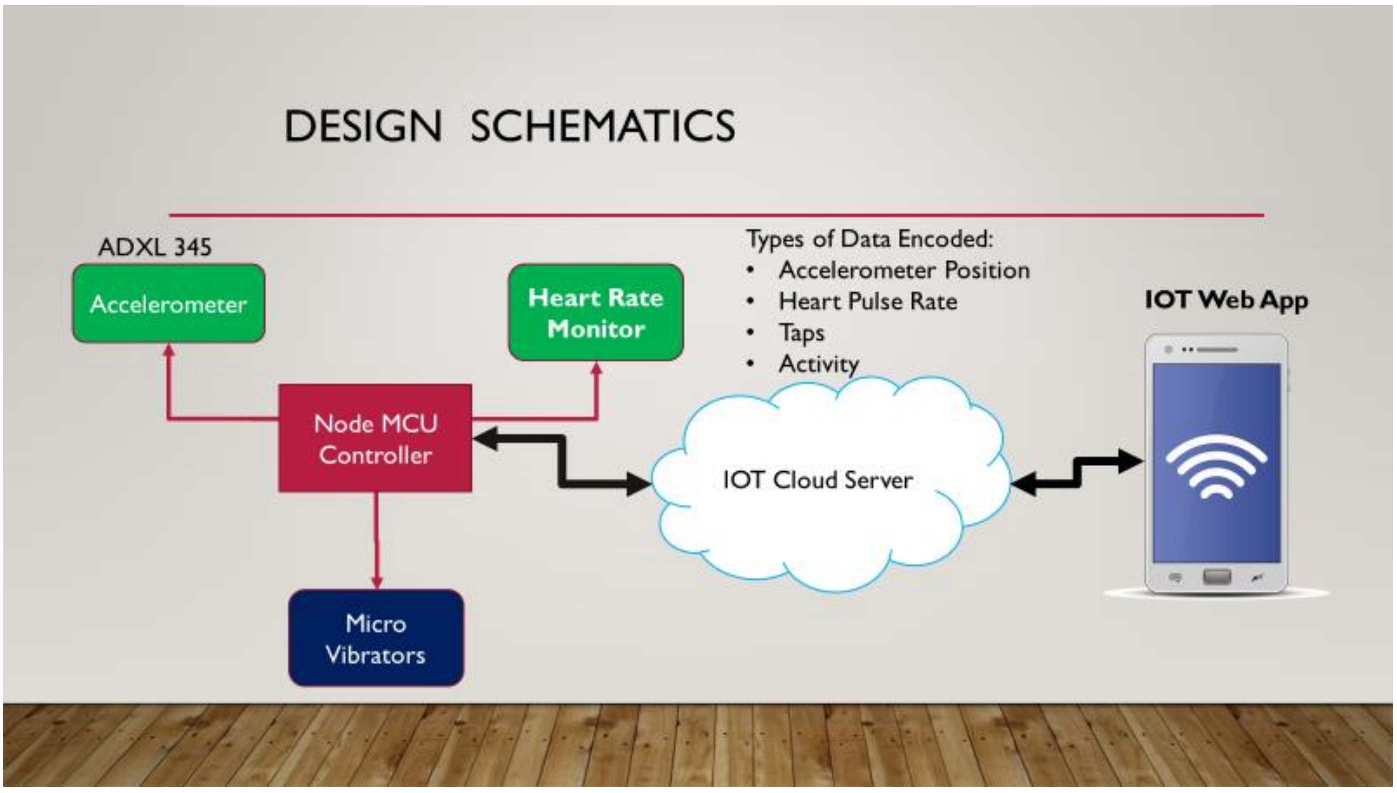
- **Better Monitoring System with IOT**

The disabled person's state can always be monitored by the helper or by themselves as an independent using IOT technology. It automatically detects dangers and alarms others.

- **Safety**

  - fall detection
  - Active / Inactive
  - Heart Rate limits

DESIGN SCHEMATIC

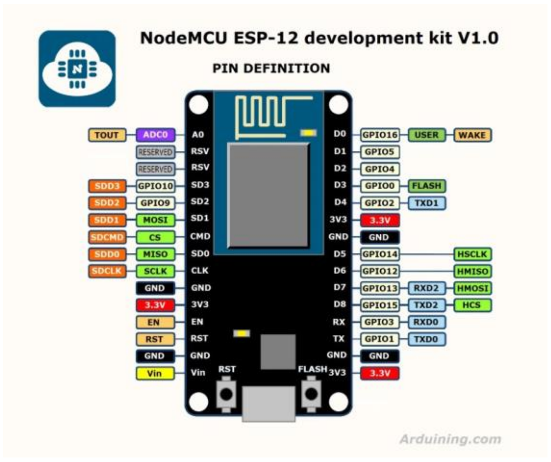


COMPONENTS USED

- **<u>Node MCU</u>**
- **<u>Accelerometer Sensor</u>**
- **<u>Heart beat Sensor</u>**
- **<u>Micro Vibrator</u>**
- **<u>Battery 3.7 Volt LIPO 250 MAH</u>**
- **<u>Node Red IOT Server</u>**
- **<u>Lithium battery charging module</u>**

Node MCU

- It is a Wi-Fi enabled Microcontroller.
- It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.
- It can be programmed through Arduino IDE.
- This will transfer data from the system to IOT cloud and vice versa.

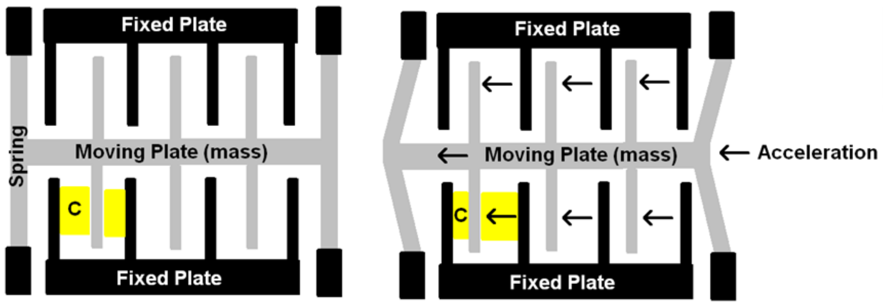


ACCELEROMETER SENSOR

- An accelerometer is an electromechanical device used to measure acceleration forces.

PURPOSE :

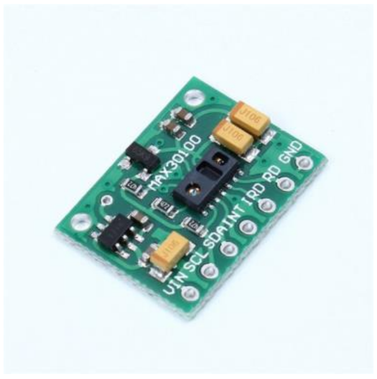
- Used to monitor the user's body positions and movements because of their connection to particular diseases (For example : Restless legs syndrome).
- It helps to detect fainting or falling of the operator while working.



Accelerometer Sensor MEM Mechanism

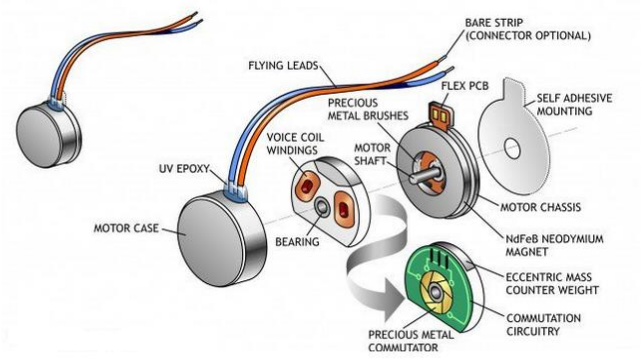
HEART BEAT SENSOR

- To detect the pulse of the elderly
- Plug and play module
- Range of pulse: 60-120 bpm
- Optical rate sensor with amplification and noise cancellation



MICRO VIBRATOR [Precision Haptics]

- Eccentric mass type micro vibrator.
- DC electric motors that have got offset weight in them and so they vibrate when they spin.
- Button-sized vibrating motors.
- They are great for haptics.

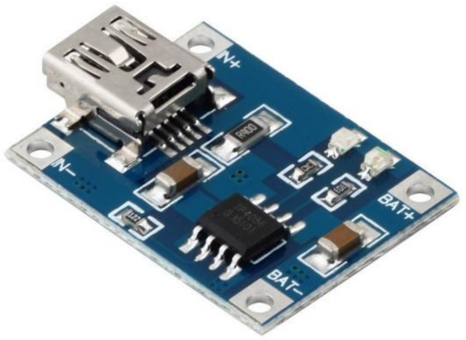


Battery 3.7 Volt LIPO 250 MAH

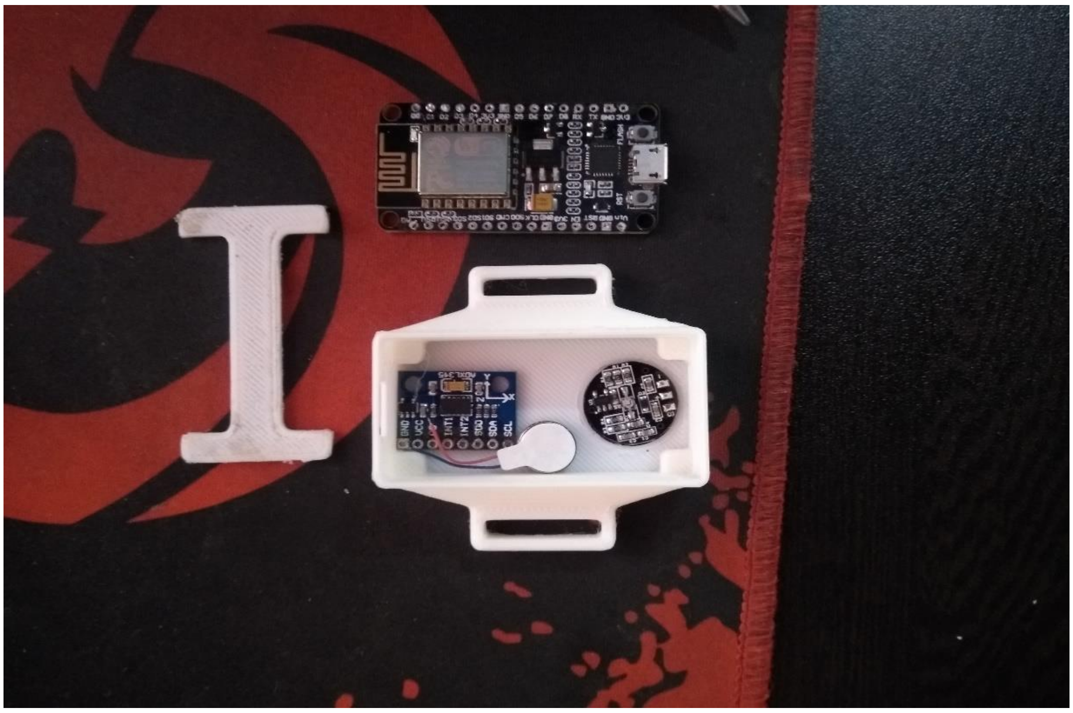
- Size:
- Length: 25 mm
- Width:20 mm
- Height:6 mm
- Weight: 5 grams
- Can be recharged and reused again.
- Gives 1C power rating for small size used in wearables, small smartphones and micro drones.

LITHIUM BATTERY CHARGING MODULE – TP 4056

- Charging current: 1A Adjustable (Through RPROG)
- Working temperature: -10 Degree to +85 Degree
- The charging board can also be powered by pin (IN + and IN-) and Onboard MINI USB head can directly link computer USB port charging
- Input voltage: 4V-8V maximum output charging current: 1000mA
- Dimensions: 25 * 19 * 10mm



CASE DESIGN



Ergonomics is a key aspect to consider while designing a product, as HapTap is a device that will be used by people extensively. The following are the factors we considered while designing the case that encloses the device.

- Wearability & comfort:
  One of the most important factors for wearable devices concerns the freedom from discomfort. Ideally users should eventually forget they are wearing a device after some time wearing it. Comfort involves an acceptable weight, shape, size, temperature, texture. Comfortable devices need to fit users enabling normal and even abnormal movements, without constraints, this requires consideration of a user populations anthropometrics and even bio-mechanical principles.

- Intuitive and simple user interface design:
  A straightforward, simple, responsive and intuitive interface (the user can immediately understand how interaction occurs, simply and easily) enhances the usability levels of the device, this includes all aspects of a user’s interaction form a supporting website through to the device. This also relates to affordance where the physical aspects of a control or device give the user visual or haptic clues as to how it operates and follows a user’s mental model and providing a source for easy error recovery allowing them to

operate the device efficiently. Contextual awareness of the device may be used to reduce cognitive load.

- Accessibility:
  Wearable devices will hopefully bring about opportunities to provide more accessible solutions to end users who have situation-induced impairments or disabilities. This could be achieved with haptics, voice control, gesture recognition and so on.

- Human-centred design:
  Being an ergonomics and human factors engineer I will always advocate that the key to a successful design is the inclusion of ergonomics and users throughout a iterative design process, including consideration of the varying user groups, their physical and cognitive limitations and all the contexts and environments of use, the wearable device will be used in and even social or cultural considerations. Prototypes should be evaluated during non-expert, long term realistic user trials, methods such as journaling can provide useful insight.

- Safety & reliability:
  Wearable devices must not bring any harm (physical, social, psychological) to its wearer, so problems such as overheating or electric shock need to be avoided. Although being a new field of ergonomics and human factors there is relatively little data on the physiological impact of wearing such devices over long periods of time. One concern is that they can also provide a constant source of distraction from the task at hand or information overload, this could be of particular concern whilst driving, for example. A wearable device must be accurate, timely and reliable in order to build a user's confidence and trust in such a device, any error tolerances must be clearly identified to the user.

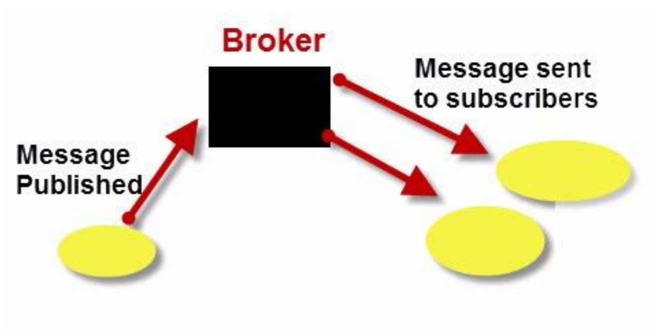
- Social acceptance and aesthetics:
  Overall acceptable has increased over the years, although such wearable devices are still not fully established as part of our daily lives, but I am sure that sometime in the future wearable devices including those in clothing with be part of everyone's wardrobe. Where systems are visible it may be socially more acceptable to provide visible indication that the device is on/off.

NODE RED IOT SOFTWARE

- Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.
- Node-RED provides a browser-based flow editor, which can be used to create JavaScript functions.
- The flows created in Node-RED are stored using JSON(JavaScript Object Notation)

MQTT PROTOCOL

- MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information. The protocol, which uses a publish/subscribe communication pattern, is used for machine-to-machine (M2M) communication and plays an important role in the internet of things (IoT).



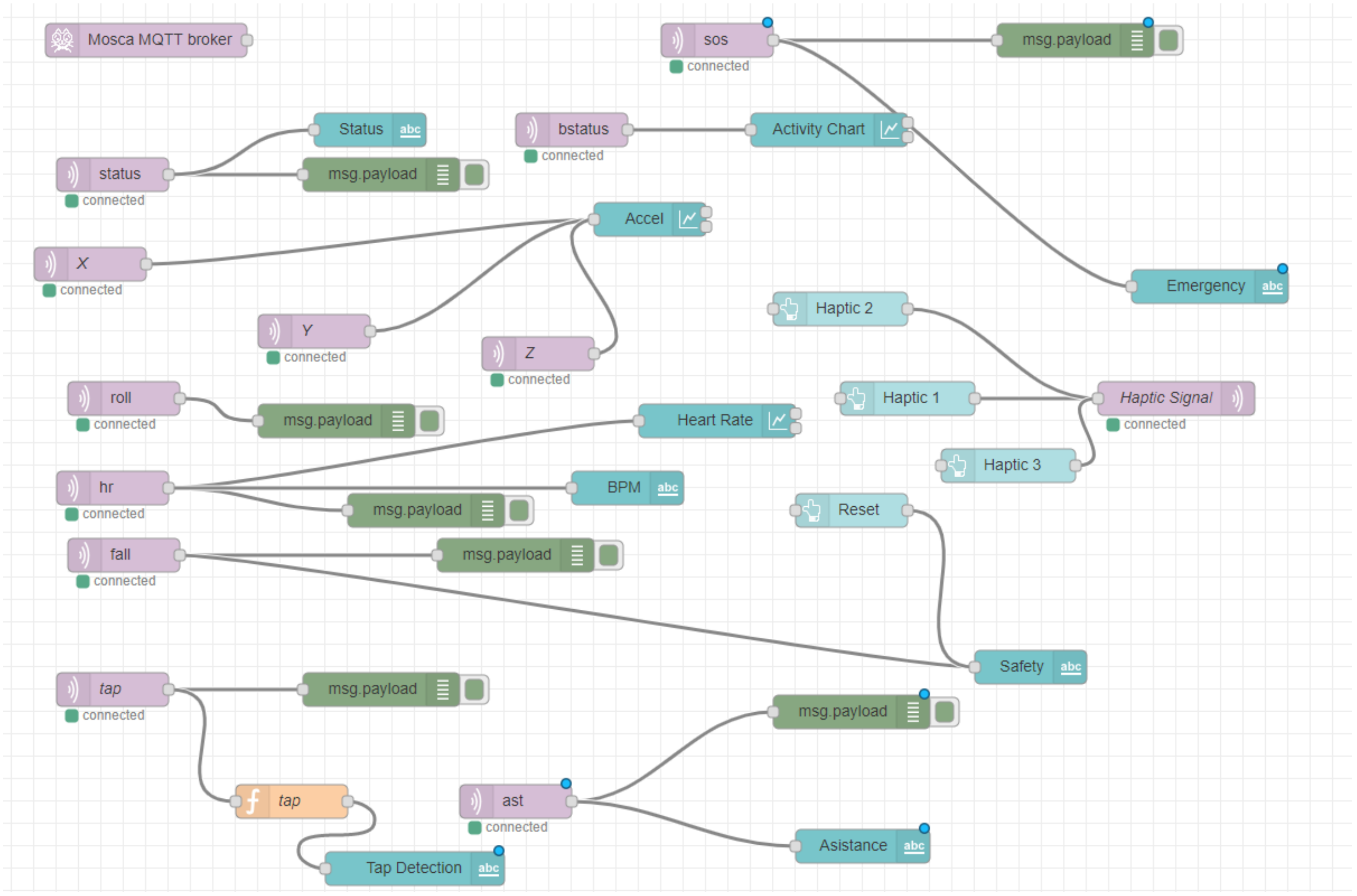
MQTT- Publish Subscribe Model

- This model makes it possible to send messages to 0,1 or multiple clients.

- A useful analogy is TV or radio.

- A TV broadcaster broadcasts a TV program using a specific channel and a viewer tunes into this channel to view the broadcast.There is no direct connection between the broadcaster and the viewer.

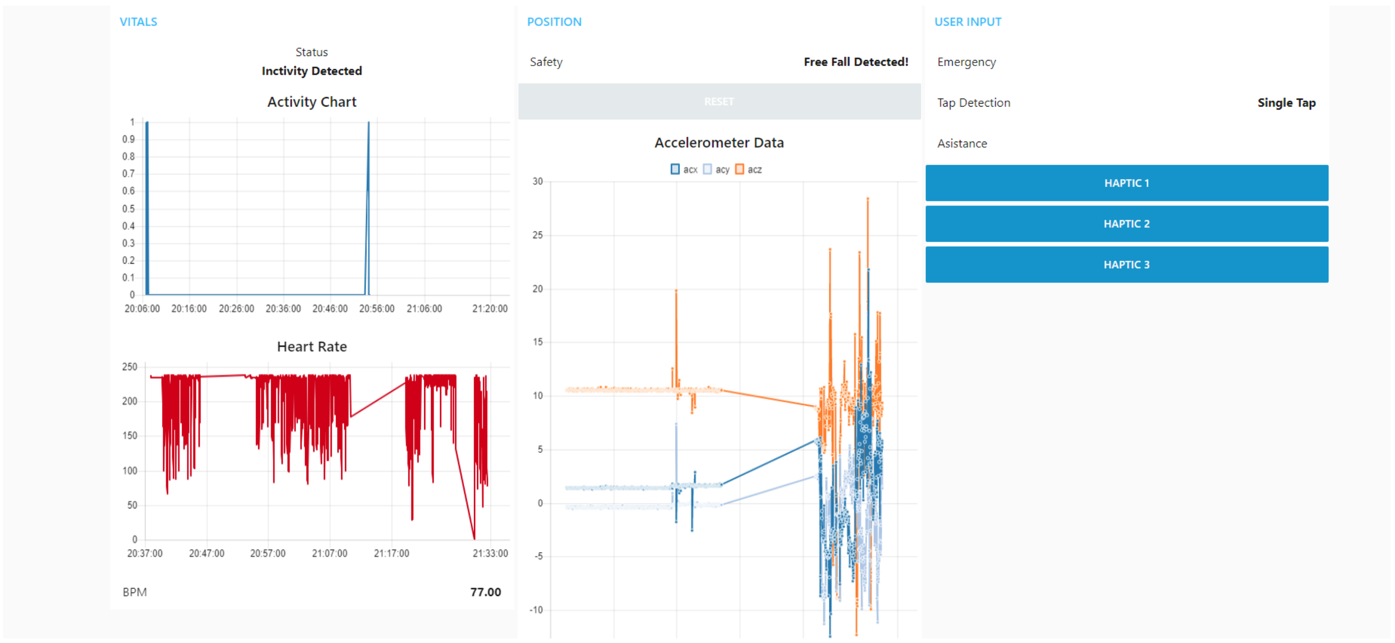
NODE RED FLOW DIAGRAM



PROTOTYPE WORKING

- The accelerometer sensor records the activity status, fall detection and location axis of the device as a numeric data and the node red software converts the numeric data into a graph and displays it in the user interface panel.
- The heart rate sensor records the heart beat and the nod red software displays the value in terms of graph.

USER INTERFACE



Status
Inctivity Detected

BPM 77.00

Safety Free Fall Detected!

Emergency

Tap Detection Single Tap

Asistance

The above user interface has the feature to monitor

- Accelerometer Position
- Heart Pulse Rate
- Taps
- Activity
- Free Fall Detection

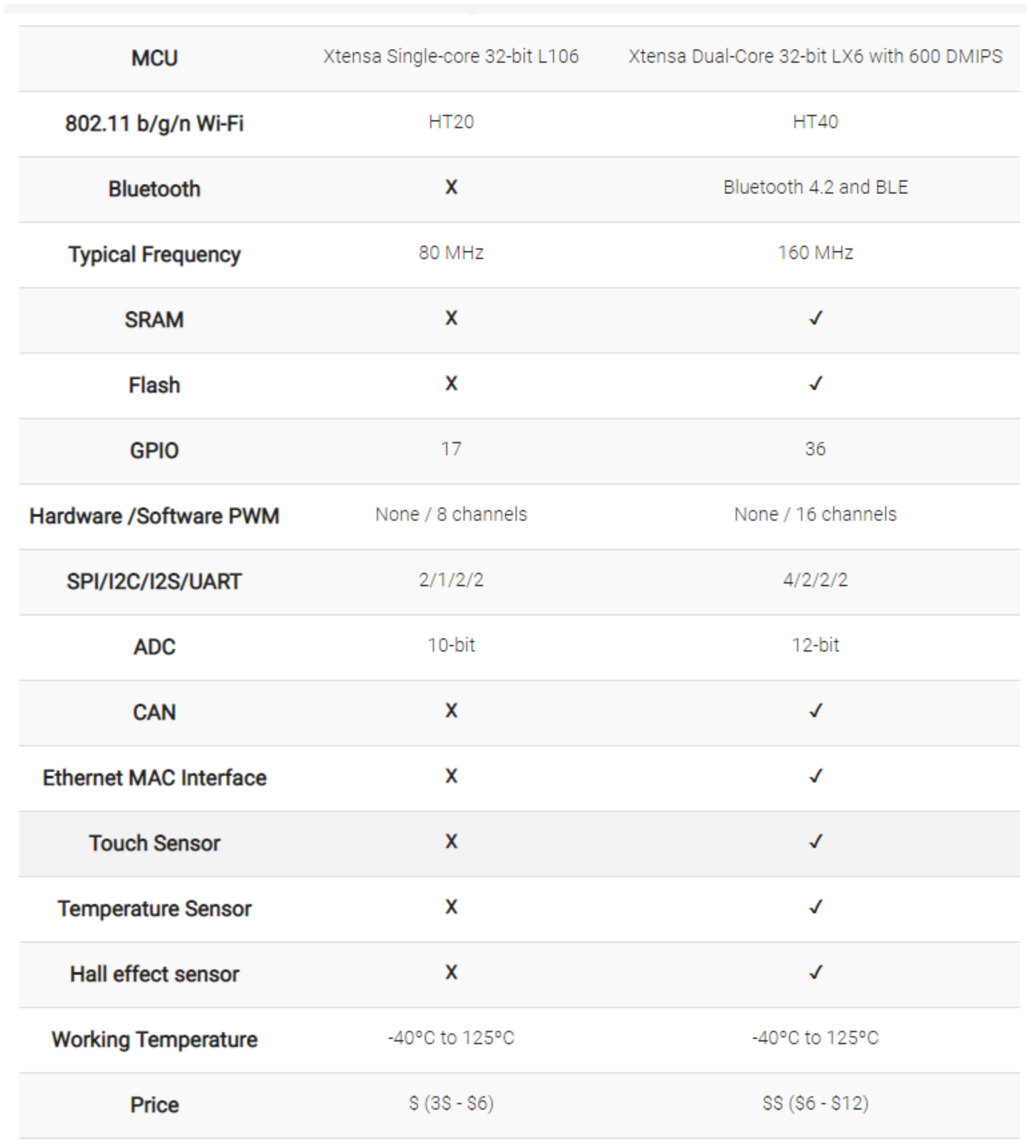
ARDUINO CODING

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <ADXL345.h>
#include <Ticker.h>


Ticker flipper;
volatile int BPM;                   // used to hold the pulse rate
volatile int Signal;                // holds the incoming raw data
volatile int IBI = 600;             // holds the time between beats, must be
seeded!
volatile boolean Pulse = false;     // true when pulse wave is high, false when
it's low
volatile boolean QS = false;        // becomes true when Arduoino finds a beat.
int count =0;
ADXL345 accelerometer;

const char* ssid = "Fazil Hathway";
const char* password = "mfazil786";
const char* mqtt_server = "192.168.1.5";
WiFiClient espClient;
PubSubClient client(espClient);

int b1 = D7;

long now = millis();
long lastMeasure = 0;

void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("WiFi connected - ESP IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
      client.setServer(mqtt_server, 1883);
    if (client.connect("ESP8266Client")) {
      Serial.println("connected");
      // Subscribe or resubscribe to a topic
      // You can subscribe to more topics (to control more LEDs in this example)
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void setup() {

  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  // Initialize ADXL345
  pinMode(b1,OUTPUT);

  interruptSetup();
  Serial.println("Initialize ADXL345");

  delay(3000);

  if (!accelerometer.begin())
  {
    Serial.println("Could not find a valid ADXL345 sensor, check wiring!");
    delay(500);}
     accelerometer.setRange(ADXL345_RANGE_16G);
    //Activity Settings
    accelerometer.setActivityThreshold(2.0);    // Recommended 2 g
    accelerometer.setInactivityThreshold(2.0);  // Recommended 2 g
    accelerometer.setTimeInactivity(5);         // Recommended 5 s
    accelerometer.setActivityXYZ(1);
    accelerometer.setInactivityXYZ(1);

   //Fall Detection
     // Values for Free Fall detection
  accelerometer.setFreeFallThreshold(0.5); // Recommended 0.3 -0.6 g
  accelerometer.setFreeFallDuration(0.1);  // Recommended 0.1 s
/*
    // Set tap detection on Z-Axis
  accelerometer.setTapDetectionX(0);       // Don't check tap on X-Axis
  accelerometer.setTapDetectionY(0);       // Don't check tap on Y-Axis
  accelerometer.setTapDetectionZ(1);       // Check tap on Z-Axis
  // or
*/
   accelerometer.setTapDetectionXYZ(1);  // Check tap on X,Y,Z-Axis

  accelerometer.setTapThreshold(2.5);      // Recommended 2.5 g
  accelerometer.setTapDuration(0.02);      // Recommended 0.02 s
  accelerometer.setDoubleTapLatency(0.10); // Recommended 0.10 s
  accelerometer.setDoubleTapWindow(0.30);  // Recommended 0.30 s



  // Select INT 1 for get activities
  accelerometer.useInterrupt(ADXL345_INT1);


   checkSetup();
}

void checkSetup()
{
  Serial.print("Activity Threshold = ");
Serial.println(accelerometer.getActivityThreshold());
```

```
  Serial.print("Inactivity Threshold = ");
Serial.println(accelerometer.getInactivityThreshold());
  Serial.print("Time Inactivity = ");
Serial.println(accelerometer.getTimeInactivity());

  Serial.print("Look activity on axis = ");
  if (accelerometer.getActivityX()) { Serial.print(" X "); }
  if (accelerometer.getActivityY()) { Serial.print(" Y "); }
  if (accelerometer.getActivityZ()) { Serial.print(" Z "); }
  Serial.println();

  Serial.print("Look inactivity on axis = ");
  if (accelerometer.getInactivityX()) { Serial.print(" X "); }
  if (accelerometer.getInactivityY()) { Serial.print(" Y "); }
  if (accelerometer.getInactivityZ()) { Serial.print(" Z "); }
  Serial.println();


  Serial.print("Free Fall Threshold = ");
Serial.println(accelerometer.getFreeFallThreshold());
  Serial.print("Free Fall Duration = ");
Serial.println(accelerometer.getFreeFallDuration());



    Serial.print("Look tap on axis = ");
  if (accelerometer.getTapDetectionX()) { Serial.print(" X "); }
  if (accelerometer.getTapDetectionY()) { Serial.print(" Y "); }
  if (accelerometer.getTapDetectionZ()) { Serial.print(" Z "); }
  Serial.println();

  Serial.print("Tap Threshold = ");
Serial.println(accelerometer.getTapThreshold());
  Serial.print("Tap Duration = "); Serial.println(accelerometer.getTapDuration());
  Serial.print("Double Tap Latency = ");
Serial.println(accelerometer.getDoubleTapLatency());
  Serial.print("Double Tap Window = ");
Serial.println(accelerometer.getDoubleTapWindow());
}
void loop() {

   if (!client.connected())
   {
    Serial.println("Reconnecting");
   client.setServer(mqtt_server, 1883);}
  if(!client.loop())
    client.connect("ESP8266Client");
    // Read values for activities
  Vector norm = accelerometer.readNormalize();

  // Read activities
  Activites activ = accelerometer.readActivites();

  if (activ.isActivity)
  {
    Serial.println("Activity Detected");

    client.publish("status", "Activity Detected");
    client.publish("bstatus", "1");
     client.loop();
  }

  if (activ.isInactivity)
  {
    Serial.println("Inactivity Detected");

    client.publish("status", "Inctivity Detected");
    client.publish("bstatus", "0");client.loop();
  }
    if (activ.isFreeFall)
  {
    Serial.println("Free Fall Detected!");

    client.publish("fall", "Free Fall Detected!");client.loop();
  }

    if (activ.isDoubleTap)
  {
    Serial.println("Double Tap Detected");

    client.publish("tap", "dtap");client.loop();

  } else
  if (activ.isTap)
  {
    Serial.println("Tap Detected");

    client.publish("tap", "stap");client.loop();
  }
  int pitch = -(atan2(norm.XAxis, sqrt(norm.YAxis*norm.YAxis +
norm.ZAxis*norm.ZAxis))*180.0)/M_PI;
  int roll  = (atan2(norm.YAxis, norm.ZAxis)*180.0)/M_PI;
  String pr = "Pitch = "+String(pitch)+" Roll = "+String(roll);
  static char data[7];
  dtostrf(pitch, 6, 2,data);
//  client.publish("pitch", data);client.loop();

//
//  static char accelx[7];
//  dtostrf(norm.XAxis, 6, 2,accelx);
//  client.publish("acx",accelx);client.loop();
//
//  static char accely[7];
//  dtostrf(norm.YAxis, 6, 2,accely);
//  client.publish("acy",accely);client.loop();
//
//  static char accelz[7];
//  dtostrf(norm.ZAxis, 6, 2,accelz);
//  client.publish("acz",accelz);client.loop();
 Serial.println(BPM);
static char bpm[7];
dtostrf(BPM, 6, 2,bpm);
Serial.print("BPM :");
Serial.println(bpm);
if(count==2)
{client.publish("hr", bpm);
 count=0;}
 count=count+1;
  delay(500);
}
void p1()
{
  digitalWrite(b1,HIGH);
  delay(300);
  digitalWrite(b1,LOW);
  delay(300);
  digitalWrite(b1,HIGH);
  delay(300);
  digitalWrite(b1,LOW);

  }

void p2()
{
  digitalWrite(b1,HIGH);
  delay(1500);
  digitalWrite(b1,LOW);

  }

 void p3()
 {
  digitalWrite(b1,HIGH);
  delay(300);
  digitalWrite(b1,LOW);
  delay(600);
  digitalWrite(b1,HIGH);
  delay(1000);
  digitalWrite(b1,LOW);

  delay(1000);

 digitalWrite(b1,HIGH);
  delay(300);
  digitalWrite(b1,LOW);
  delay(600);
  digitalWrite(b1,HIGH);
  delay(1000);
  digitalWrite(b1,LOW);

  delay(1000);

   digitalWrite(b1,HIGH);
  delay(300);
  digitalWrite(b1,LOW);
  delay(600);
  digitalWrite(b1,HIGH);
  delay(1000);
  digitalWrite(b1,LOW);

  }
```

FUTURE SCOPE

- These Six Haptic Vibrators can be arranged in user's hands .
- Every Braille representation of a letter is done by 6 dots
- An information message can be translated in braille form.
- The change of Node MCU from ESP32 to ESP8266. It adds a CPU core, faster Wi-Fi, more GPIOs, and supports Bluetooth 4.2 and Bluetooth low energy. Additionally, the ESP32 comes with touch sensitive pins, and built-in hall effect sensor and temperature sensor.
- Comparison of ESP8266 and ESP 32



MCU	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6 with 600 DMIPS
802.11 b/g/n Wi-Fi	HT20	HT40
Bluetooth	X	Bluetooth 4.2 and BLE
Typical Frequency	80 MHz	160 MHz
SRAM	X	✓
Flash	X	✓
GPIO	17	36
Hardware /Software PWM	None / 8 channels	None / 16 channels
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	X	✓
Ethernet MAC Interface	X	✓
Touch Sensor	X	✓
Temperature Sensor	X	✓
Hall effect sensor	X	✓
Working Temperature	-40°C to 125°C	-40°C to 125°C
Price	$ (3$ - $6)	$$ ($6 - $12)