## Appendix G – PROGRAM SOURCE CODES

**Linear Regression Algorithm**

print("LINEAR REGRESSION ALGORITHM IMPLEMENTATION")

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

dataset=pd.read\_csv('hypothetical.csv')

# split dataset

X = data.drop(columns =['StdOrder', 'RunOrder' , 'Anti-absconding Decision'])

y = data.drop(columns =['StdOrder', 'RunOrder' , 'Heart Rate', 'Temperature', 'Accelerometer', 'Removal Detection' , "Indoor Position"])

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X,y,random\_state=0,test\_size=0.2)

model = LinearRegression()

# fit model

model.fit(X, y)

y\_pred = model.predict(X\_test)

# summarize prediction

print("\nTest Data:")

print(y\_test)

print("\nLinear Regression Prediction:")

print(y\_pred)

# evaluate the model and collect the scores

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedKFold

from numpy import absolute

from numpy import std

cv = RepeatedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

n\_scores = cross\_val\_score(model, X, y, scoring='neg\_mean\_absolute\_error', cv=cv, n\_jobs=-1)

# force the scores to be positive

n\_scores = absolute(n\_scores)

# summarize performance

MSE=mean\_squared\_error(y\_test,y\_pred)

print("\nLinear Regression Performance Metrics:")

print('MSE (std): %.3f (%.3f)' % (MSE, std(n\_scores)))

r2=r2\_score(y\_test, y\_pred)

print('R-squared: %.3f' % (r2))

**KNN Algorithm**

print("KNN ALGORITHM IMPLEMENTATION")

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsRegressor

ddataset=pd.read\_csv('hypothetical.csv')

# split dataset

X = data.drop(columns =['StdOrder', 'RunOrder' , 'Anti-absconding Decision'])

y = data.drop(columns =['StdOrder', 'RunOrder' , 'Heart Rate', 'Temperature', 'Accelerometer', 'Removal Detection' , "Indoor Position"])

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X,y,random\_state=0,test\_size=0.2)

model = KNeighborsRegressor(n\_neighbors=7)

# fit model

model.fit(X, y)

y\_pred = model.predict(X\_test)

# summarize prediction

print("\nTest Data:")

print(y\_test)

print("\nKNN Prediction:")

print(y\_pred)

# evaluate the model and collect the scores

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedKFold

from numpy import absolute

from numpy import std

cv = RepeatedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

n\_scores = cross\_val\_score(model, X, y, scoring='neg\_mean\_absolute\_error', cv=cv, n\_jobs=-1)

# force the scores to be positive

n\_scores = absolute(n\_scores)

# summarize performance

MSE=mean\_squared\_error(y\_test,y\_pred)

print("\nKNN Performance Metrics:")

print('MSE (std): %.3f (%.3f)' % (MSE, std(n\_scores)))

r2=r2\_score(y\_test, y\_pred)

print('R-squared: %.3f' % (r2))

**Decision Tree Algorithm**

print("DECISION TREE ALGORITHM IMPLEMENTATION")

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

dataset=pd.read\_csv('hypothetical.csv')

# split dataset

X = data.drop(columns =['StdOrder', 'RunOrder' , 'Anti-absconding Decision'])

y = data.drop(columns =['StdOrder', 'RunOrder' , 'Heart Rate', 'Temperature', 'Accelerometer', Removal Detection' , "Indoor Position"])

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X,y,random\_state=0,test\_size=0.2)

model = DecisionTreeRegressor()

# fit model

model.fit(X, y)

y\_pred = model.predict(X\_test)

# summarize prediction

print("\nTest Data:")

print(y\_test)

print("\nDecision Tree Prediction:")

print(y\_pred)

# evaluate the model and collect the scores

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedKFold

from numpy import absolute

from numpy import std

cv = RepeatedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

n\_scores = cross\_val\_score(model, X, y, scoring='neg\_mean\_absolute\_error', cv=cv, n\_jobs=-1)

# force the scores to be positive

n\_scores = absolute(n\_scores)

# summarize performance

MSE=mean\_squared\_error(y\_test,y\_pred)

print("\nDecision Tree Performance Metrics:")

print('MSE (std): %.3f (%.3f)' % (MSE, std(n\_scores)))

r2=r2\_score(y\_test, y\_pred)

print('R-squared: %.3f' % (r2))

**Random Forest Algorithm**

print("RANDOM FOREST ALGORITHM IMPLEMENTATION")

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

dataset=pd.read\_csv('hypothetical.csv')

# split dataset

X = data.drop(columns =['StdOrder', 'RunOrder' , 'Anti-absconding Decision'])

y = data.drop(columns =['StdOrder', 'RunOrder' , 'Heart Rate', 'Temperature', 'Accelerometer', Removal Detection' , "Indoor Position"])

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X,y,random\_state=0,test\_size=0.2)

model = RandomForestRegressor()

# fit model

model.fit(X, y.values.ravel())

y\_pred = model.predict(X\_test)

# summarize prediction

print("\nTest Data:")

print(y\_test)

print("\nRandom Forest Prediction:")

print(y\_pred)

# evaluate the model and collect the scores

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedKFold

from numpy import absolute

from numpy import std

cv = RepeatedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

n\_scores = cross\_val\_score(model, X, y, scoring='neg\_mean\_absolute\_error', cv=cv, n\_jobs=-1)

# force the scores to be positive

n\_scores = absolute(n\_scores)

# summarize performance

MSE=mean\_squared\_error(y\_test,y\_pred)

print("\nRandom Forest Performance Metrics:")

print('MSE (std): %.3f (%.3f)' % (MSE, std(n\_scores)))

r2=r2\_score(y\_test, y\_pred)

print('R-squared: %.3f' % (r2))

**udp\_uwb\_anchor**

#include <SPI.h>

#include "DW1000Ranging.h"

#define ANCHOR\_ADD "82:17:5B:D5:A9:9A:E2:9C"

#define SPI\_SCK 18

#define SPI\_MISO 19

#define SPI\_MOSI 23

#define DW\_CS 4

// connection pins

const uint8\_t PIN\_RST = 27; // reset pin

const uint8\_t PIN\_IRQ = 34; // irq pin

const uint8\_t PIN\_SS = 4; // spi select pin

void setup()

{

Serial.begin(115200);

delay(1000);

//init the configuration

SPI.begin(SPI\_SCK, SPI\_MISO, SPI\_MOSI);

DW1000Ranging.initCommunication(PIN\_RST, PIN\_SS, PIN\_IRQ); //Reset, CS, IRQ pin

//define the sketch as anchor. It will be great to dynamically change the type of module

DW1000Ranging.attachNewRange(newRange);

DW1000Ranging.attachBlinkDevice(newBlink);

DW1000Ranging.attachInactiveDevice(inactiveDevice);

//Enable the filter to smooth the distance

//DW1000Ranging.useRangeFilter(true);

//we start the module as an anchor

// DW1000Ranging.startAsAnchor("82:17:5B:D5:A9:9A:E2:9C", DW1000.MODE\_LONGDATA\_RANGE\_ACCURACY);

DW1000Ranging.startAsAnchor(ANCHOR\_ADD, DW1000.MODE\_LONGDATA\_RANGE\_LOWPOWER, false);

// DW1000Ranging.startAsAnchor(ANCHOR\_ADD, DW1000.MODE\_SHORTDATA\_FAST\_LOWPOWER);

// DW1000Ranging.startAsAnchor(ANCHOR\_ADD, DW1000.MODE\_LONGDATA\_FAST\_LOWPOWER);

// DW1000Ranging.startAsAnchor(ANCHOR\_ADD, DW1000.MODE\_SHORTDATA\_FAST\_ACCURACY);

// DW1000Ranging.startAsAnchor(ANCHOR\_ADD, DW1000.MODE\_LONGDATA\_FAST\_ACCURACY);

// DW1000Ranging.startAsAnchor(ANCHOR\_ADD, DW1000.MODE\_LONGDATA\_RANGE\_ACCURACY);

}

void loop()

{

DW1000Ranging.loop();

}

void newRange()

{

Serial.print("from: ");

Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);

Serial.print("\t Range: ");

Serial.print(DW1000Ranging.getDistantDevice()->getRange());

Serial.print(" m");

Serial.print("\t RX power: ");

Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());

Serial.println(" dBm");

}

void newBlink(DW1000Device \*device)

{

Serial.print("blink; 1 device added ! -> ");

Serial.print(" short:");

Serial.println(device->getShortAddress(), HEX);

}

void inactiveDevice(DW1000Device \*device)

{

Serial.print("delete inactive device: ");

Serial.println(device->getShortAddress(), HEX);

}

**udp\_uwb\_tag**

#include <SPI.h>

#include <SPI.h>

#include <DW1000Ranging.h>

#include <WiFi.h>

#include "link.h"

#define SPI\_SCK 18

#define SPI\_MISO 19

#define SPI\_MOSI 23

#define DW\_CS 4

#define PIN\_RST 27

#define PIN\_IRQ 34

const char\* ssid = "HUAWEI-2.4G-Y7Bv";//Aries' House

const char\* password = "DxmwQ2w3";

const char \*host = "192.168.100.16";

WiFiClient client;

struct MyLink \*uwb\_data;

int index\_num = 0;

long runtime = 0;

String all\_json = "";

void setup()

{

Serial.begin(115200);

WiFi.mode(WIFI\_STA);

WiFi.setSleep(false);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED)

{

delay(500);

Serial.print(".");

}

Serial.println("Connected");

Serial.print("IP Address:");

Serial.println(WiFi.localIP());

if (client.connect(host, 80))

{

Serial.println("Success");

client.print(String("GET /") + " HTTP/1.1\r\n" +

"Host: " + host + "\r\n" +

"Connection: close\r\n" +

"\r\n");

}

delay(1000);

//init the configuration

SPI.begin(SPI\_SCK, SPI\_MISO, SPI\_MOSI);

DW1000Ranging.initCommunication(PIN\_RST, DW\_CS, PIN\_IRQ);

DW1000Ranging.attachNewRange(newRange);

DW1000Ranging.attachNewDevice(newDevice);

DW1000Ranging.attachInactiveDevice(inactiveDevice);

//we start the module as a tag

DW1000Ranging.startAsTag("7D:00:22:EA:82:60:3B:9C", DW1000.MODE\_LONGDATA\_RANGE\_LOWPOWER);

uwb\_data = init\_link();

}

void loop()

{

DW1000Ranging.loop();

if ((millis() - runtime) > 1000)

{

make\_link\_json(uwb\_data, &all\_json);

send\_udp(&all\_json);

runtime = millis();

}

}

void newRange()

{

char c[30];

Serial.print("from: ");

Serial.print(DW1000Ranging.getDistantDevice()->getShortAddress(), HEX);

Serial.print("\t Range: ");

Serial.print(DW1000Ranging.getDistantDevice()->getRange());

Serial.print(" m");

Serial.print("\t RX power: ");

Serial.print(DW1000Ranging.getDistantDevice()->getRXPower());

Serial.println(" dBm");

fresh\_link(uwb\_data, DW1000Ranging.getDistantDevice()->getShortAddress(), DW1000Ranging.getDistantDevice()->getRange(), DW1000Ranging.getDistantDevice()->getRXPower());

}

void newDevice(DW1000Device \*device)

{

Serial.print("ranging init; 1 device added ! -> ");

Serial.print(" short:");

Serial.println(device->getShortAddress(), HEX);

add\_link(uwb\_data, device->getShortAddress());

}

void inactiveDevice(DW1000Device \*device)

{

Serial.print("delete inactive device: ");

Serial.println(device->getShortAddress(), HEX);

delete\_link(uwb\_data, device->getShortAddress());

}

void send\_udp(String \*msg\_json)

{

if (client.connected())

{

client.print(\*msg\_json);

Serial.println("UDP send");

}

}

**uwb\_position\_display**

import time

import turtle

import cmath

import socket

import json

from cmath import sqrt

hostname = socket.gethostname()

UDP\_IP = socket.gethostbyname(hostname)

print("\*\*\*Local ip:" + str(UDP\_IP) + "\*\*\*")

UDP\_PORT = 80

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.bind((UDP\_IP, UDP\_PORT))

sock.listen(1)

data, addr = sock.accept()

distance\_a1\_a2 = 3.0

meter2pixel = 100

range\_offset = 0.9

def screen\_init(width=1200, height=800, t=turtle):

t.setup(width, height)

t.tracer(False)

t.hideturtle()

t.speed(0)

def turtle\_init(t=turtle):

t.hideturtle()

t.speed(0)

def draw\_line(x0, y0, x1, y1, color="black", t=turtle):

t.pencolor(color)

t.up()

t.goto(x0, y0)

t.down()

t.goto(x1, y1)

t.up()

def draw\_fastU(x, y, length, color="black", t=turtle):

draw\_line(x, y, x, y + length, color, t)

def draw\_fastV(x, y, length, color="black", t=turtle):

draw\_line(x, y, x + length, y, color, t)

def draw\_cycle(x, y, r, color="black", t=turtle):

t.pencolor(color)

t.up()

t.goto(x, y - r)

t.setheading(0)

t.down()

t.circle(r)

t.up()

def fill\_cycle(x, y, r, color="black", t=turtle):

t.up()

t.goto(x, y)

t.down()

t.dot(r, color)

t.up()

def write\_txt(x, y, txt, color="black", t=turtle, f=('Arial', 12, 'normal')):

t.pencolor(color)

t.up()

t.goto(x, y)

t.down()

t.write(txt, move=False, align='left', font=f)

t.up()

def draw\_rect(x, y, w, h, color="black", t=turtle):

t.pencolor(color)

t.up()

t.goto(x, y)

t.down()

t.goto(x + w, y)

t.goto(x + w, y + h)

t.goto(x, y + h)

t.goto(x, y)

t.up()

def fill\_rect(x, y, w, h, color=("black", "black"), t=turtle):

t.begin\_fill()

draw\_rect(x, y, w, h, color, t)

t.end\_fill()

pass

def clean(t=turtle):

t.clear()

def draw\_ui(t):

write\_txt(-300, 250, "UWB Positon", "black", t, f=('Arial', 32, 'normal'))

fill\_rect(-400, 200, 800, 40, "black", t)

write\_txt(-50, 205, "WALL", "yellow", t, f=('Arial', 24, 'normal'))

def draw\_uwb\_anchor(x, y, txt, range, t):

r = 20

fill\_cycle(x, y, r, "green", t)

write\_txt(x + r, y, txt + ": " + str(range) + "M",

"black", t, f=('Arial', 16, 'normal'))

def draw\_uwb\_tag(x, y, txt, t):

pos\_x = -250 + int(x \* meter2pixel)

pos\_y = 150 - int(y \* meter2pixel)

r = 20

fill\_cycle(pos\_x, pos\_y, r, "blue", t)

write\_txt(pos\_x, pos\_y, txt + ": (" + str(x) + "," + str(y) + ")",

"black", t, f=('Arial', 16, 'normal'))

def read\_data():

line = data.recv(1024).decode('UTF-8')

uwb\_list = []

try:

uwb\_data = json.loads(line)

print(uwb\_data)

uwb\_list = uwb\_data["links"]

for uwb\_archor in uwb\_list:

print(uwb\_archor)

except:

print(line)

print("")

return uwb\_list

def tag\_pos(a, b, c):

# p = (a + b + c) / 2.0

# s = cmath.sqrt(p \* (p - a) \* (p - b) \* (p - c))

# y = 2.0 \* s / c

# x = cmath.sqrt(b \* b - y \* y)

cos\_a = (b \* b + c\*c - a \* a) / (2 \* b \* c)

x = b \* cos\_a

y = b \* cmath.sqrt(1 - cos\_a \* cos\_a)

y = y - 0.5

# edit values here

return round(x.real, 1), round(y.real, 1)

def uwb\_range\_offset(uwb\_range):

temp = uwb\_range

return temp

def main():

t\_ui = turtle.Turtle()

t\_a1 = turtle.Turtle()

t\_a2 = turtle.Turtle()

t\_a3 = turtle.Turtle()

turtle\_init(t\_ui)

turtle\_init(t\_a1)

turtle\_init(t\_a2)

turtle\_init(t\_a3)

a1\_range = 0.0

a2\_range = 0.0

draw\_ui(t\_ui)

while True:

node\_count = 0

list = read\_data()

for one in list:

if one["A"] == "1782":

clean(t\_a1)

a1\_range = uwb\_range\_offset(float(one["R"]))

draw\_uwb\_anchor(-250, 150, "A1782(0,0)", a1\_range, t\_a1)

node\_count += 1

if one["A"] == "1783":

clean(t\_a2)

a2\_range = uwb\_range\_offset(float(one["R"]))

draw\_uwb\_anchor(-250 + meter2pixel \* distance\_a1\_a2,

150, "A1783(" + str(distance\_a1\_a2)+")", a2\_range, t\_a2)

node\_count += 1

if node\_count == 2:

x, y = tag\_pos(a2\_range, a1\_range, distance\_a1\_a2)

if x > 3:

x = 3

if y < 0:

y = 0

if x < 0:

x = 0

# edit values here

print(x, y)

clean(t\_a3)

draw\_uwb\_tag(x, y, "TAG", t\_a3)

time.sleep(0.1)

turtle.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**QS\_Monitoring\_Web\_server**

#include <WiFi.h>

#include <AsyncTCP.h>

#include <ESPAsyncWebServer.h>

#include <Adafruit\_MLX90614.h>

#include <Wire.h>

#include "MAX30105.h"

#include "heartRate.h"

Adafruit\_MLX90614 mlx = Adafruit\_MLX90614();

int minusTemp = 5.7;//for temperature calibration

unsigned long startMillis; //some global variables available anywhere in the program

unsigned long currentMillis;

MAX30105 particleSensor;

const byte RATE\_SIZE = 4; //Increase this for more averaging. 4 is good.

byte rates[RATE\_SIZE]; //Array of heart rates

byte rateSpot = 0;

long lastBeat = 0; //Time at which the last beat occurred

#define SDA\_2 33

#define SCL\_2 32

const char\* ssid = "HUAWEI-2.4G-Y7Bv";//Aries' House

const char\* password = "DxmwQ2w3";

const char \*host = "192.168.100.16";

// Create AsyncWebServer object on port 80

AsyncWebServer server(80);

// Create an Event Source on /events

AsyncEventSource events("/events");

// Timer variables

unsigned long lastTime = 0;

unsigned long timerDelay = 8000;

float temperature;

float heartrate;

int beatAvg;

float ir = 0;

String deviceState = "Device was removed.";

// Initialize WiFi

void initWiFi() {

WiFi.mode(WIFI\_STA);

WiFi.begin(ssid, password);

Serial.print("Connecting to WiFi ..");

while (WiFi.status() != WL\_CONNECTED) {

Serial.print('.');

delay(1000);

}

Serial.println(WiFi.localIP());

}

String processor(const String& var) {

//getSensorReadings();

//Serial.println(var);

if (var == "TEMPERATURE") {

return String(temperature);

}

if (var == "HEART RATE") {

return String(heartrate);

}

if (var == "REMOVAL DETECTION") {

return String(ir);

}

return String();

}

const char index\_html[] PROGMEM = R"rawliteral(

<!DOCTYPE HTML><html>

<head>

<title>ESP Web Server</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">

<link rel="icon" href="data:,">

<style>

html {font-family: Arial; display: inline-block; text-align: center;}

p { font-size: 1.2rem;}

body { margin: 0;}

.topnav { overflow: hidden; background-color: #00008B; color: white; font-size: 1rem; }

.content { padding: 20px; }

.card { background-color: white; box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5); }

.cards { max-width: 800px; margin: 0 auto; display: grid; grid-gap: 2rem; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); }

.reading { font-size: 1.4rem; }

</style>

</head>

<body>

<div class="topnav">

<h1>Quarantined Subject Monitoring - WEB SERVER</h1>

</div>

<div class="content">

<div class="cards">

<div class="card">

<p><i class="fas fa-thermometer-half" style="color:#059e8a;"></i> TEMPERATURE</p><p><span class="reading"><span id="temp">%TEMPERATURE%</span> &deg;C</span></p>

</div>

<div class="card">

<p><i class="fas fa-heartbeat fa-sm" style="color:#059e8a;"> </i> HEART RATE</p><p><span class="reading"><span id="heart">%HEART RATE%</span> BPM</span></p>

</div>

<div class="card">

<p><i class="fas fa-unlock-alt fa-sm" style="color:#059e8a;"></i> REMOVAL DETECTION</p><p><span class="reading"><span id="irV">%REMOVAL DETECTION%</span></span></p>

</div>

</div>

<script>

if (!!window.EventSource) {

var source = new EventSource('/events');

source.addEventListener('open', function(e) {

console.log("Events Connected");

}, false);

source.addEventListener('error', function(e) {

if (e.target.readyState != EventSource.OPEN) {

console.log("Events Disconnected");

}

}, false);

source.addEventListener('message', function(e) {

console.log("message", e.data);

}, false);

source.addEventListener('temperature', function(e) {

console.log("temperature", e.data);

document.getElementById("temp").innerHTML = e.data;

}, false);

source.addEventListener('heartrate', function(e) {

console.log("heartrate", e.data);

document.getElementById("heart").innerHTML = e.data;

}, false);

source.addEventListener('deviceState', function(e) {

console.log("deviceState", e.data);

document.getElementById("irV").innerHTML = e.data;

}, false);

}

</script>

</body>

</html>)rawliteral";

void setup() {

Serial.begin(115200);

Wire.begin();

Wire1.begin(SDA\_2, SCL\_2);

startMillis = millis(); //initial start time

while (!Serial);

initWiFi();

if (!mlx.begin(0x5A, &Wire1))

{

Serial.println("Error connecting to MLX sensor. Check wiring.");

while (1);

};

particleSensor.begin(Wire, I2C\_SPEED\_FAST, 0x57);

particleSensor.setup(); //Configure sensor with default settings

particleSensor.setPulseAmplitudeRed(0x0A); //Turn Red LED to low to indicate sensor is running

particleSensor.setPulseAmplitudeGreen(0); //Turn off Green LED

// Handle Web Server

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \* request) {

request->send\_P(200, "text/html", index\_html, processor);

});

// Handle Web Server Events

events.onConnect([](AsyncEventSourceClient \* client) {

if (client->lastId()) {

Serial.printf("Client reconnected! Last message ID that it got is: %u\n", client->lastId());

}

// send event with message "hello!", id current millis

// and set reconnect delay to 1 second

client->send("hello!", NULL, millis(), 10000);

});

server.addHandler(&events);

server.begin();

}

void loop() {

long ir = particleSensor.getIR();

temperature = mlx.readObjectTempC() - minusTemp;

currentMillis = millis();

if (checkForBeat(ir) == true)

{

//We sensed a beat!

long delta = millis() - lastBeat;

lastBeat = millis();

heartrate = 60 / (delta / 1000.0);

if (heartrate < 255 && heartrate > 20)

{

rates[rateSpot++] = (byte)heartrate; //Store this reading in the array

rateSpot %= RATE\_SIZE; //Wrap variable

//Take average of readings

beatAvg = 0;

for (byte x = 0 ; x < RATE\_SIZE ; x++)

beatAvg += rates[x];

beatAvg /= RATE\_SIZE;

}

}

Serial.print("IR=");

Serial.print(ir);

//Serial.print(", BPM=");

// Serial.print(beatsPerMinute);

Serial.print(", Avg BPM=");

Serial.print(beatAvg);

strtMillis = currentMillis;

if (ir < 50000){

Serial.print(" Device is removed.");

deviceState = "Device was removed.";

}

else {

Serial.print(" Device is attached.");

deviceState = "Device is attached.";

}

Serial.println();

if ((millis() - lastTime) > timerDelay) {

Serial.printf("Temperature = %.2f ºC \n", temperature);

Serial.println();

// Send Events to the Web Server with the Sensor Readings

events.send("ping", NULL, millis());

events.send(String(temperature).c\_str(), "temperature", millis());

events.send(String(heartrate).c\_str(), "heartrate", millis());

events.send(deviceState.c\_str(), "deviceState", millis());

lastTime = millis();

}

}