**Project Objectives**

**The aim of the project is to design Automatic heat safety alarm for cars using Wireless Sensor Nodes**

We have to implement an automatic vehicle safety sensor to detect alarmingly high temperature levels when the vehicle is parked AND an unattended child is inside the car in distress. A mote should be programmed with an application that will detect if ALL of the following conditions are true:

* The vehicle is stationary for a certain amount of time (e.g. > 15 minutes)
* The temperature inside the car is greater than a threshold (e.g. 120ºF)
* There is a child in the car (e.g. sound of crying or loud sounds)

The sensor should then

1. Sound an alarm,
2. Send a text message (or email) to the owner of the vehicle, and
3. Activate an actuator to open the windows using the MDA.

**Principle of Operation:**

We have designed a finite state machine to implement our project. The states used are as follows: -

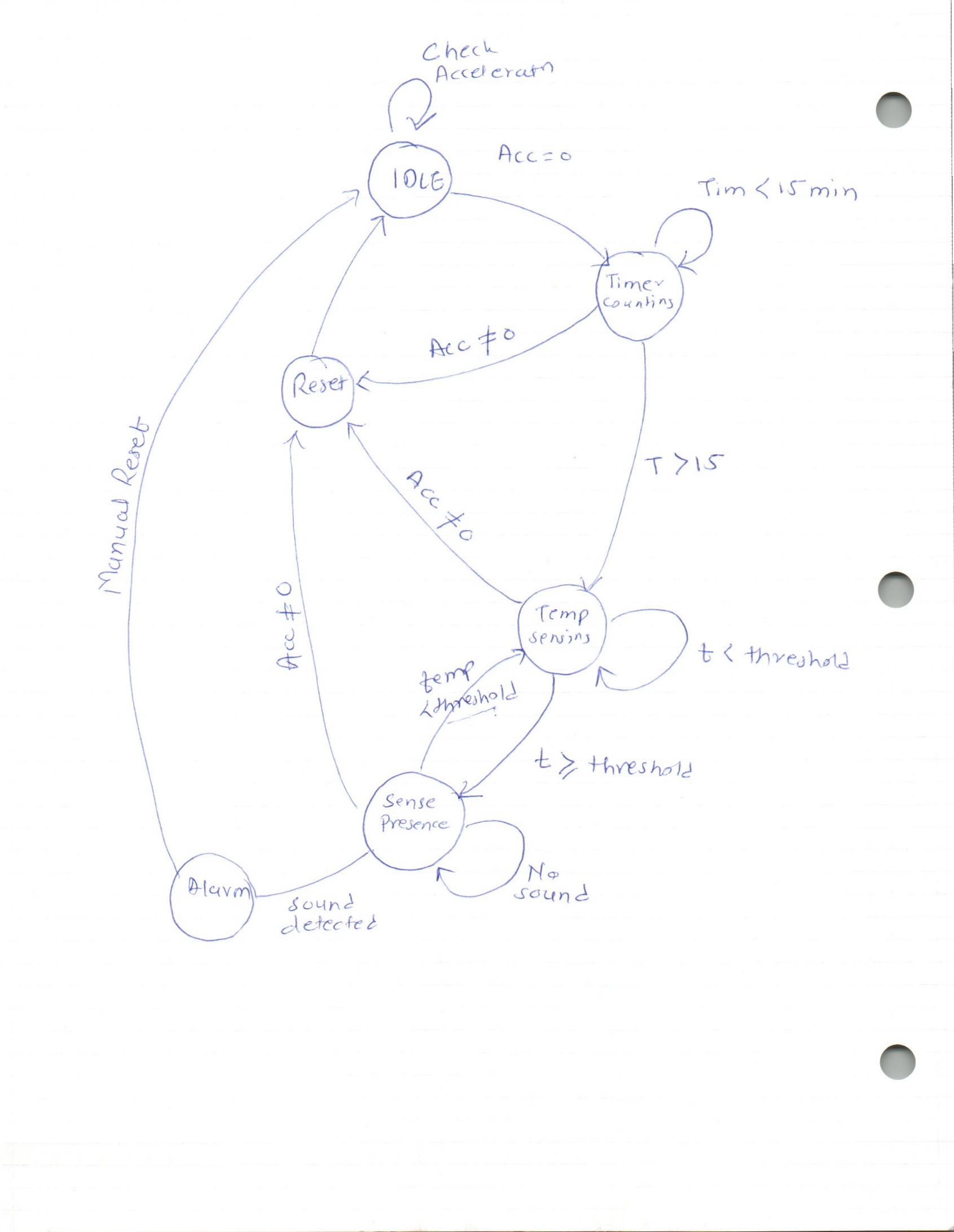
**State-1: Idle**

We read the Accelerometer values and check if the car is stationary. If the accelerometer reading is between a set threshold level it transitions into the State-2

**State-2: Time Counting**

In this state we are checking if the accelerometer reading remains in the threshold limit for a certain period of time. If so we make the transition into the next state.

If the car moves, then it goes back to the Idle state through Reset state.

****

**State-3: Temperature Sensing**

In this state the device monitors the temperature and checks if it is exceeding a certain limit. Mote also keeps checking the accelerometer reading at the same time as in the previous states.

If the temperature exceeds the limit, then the mote makes transition into the next state.

**State-4: Sound Sensing**

In this state the mote is checking if there’s a human presence in the car.

In order to do that the mote monitors the microphone readings. If the noise level exceeds a certain threshold it moves into the next state, Alarm.

In all the states we are also checking the values of sensors from previous states.

**State-5: Alarm**

If the values from all the sensors exceed their respective limits then the mote with sensor board sounds a buzzer and sends signal to the base-station, which sends an email, text-message and places a call with the help of a computer.

**Work Done:**

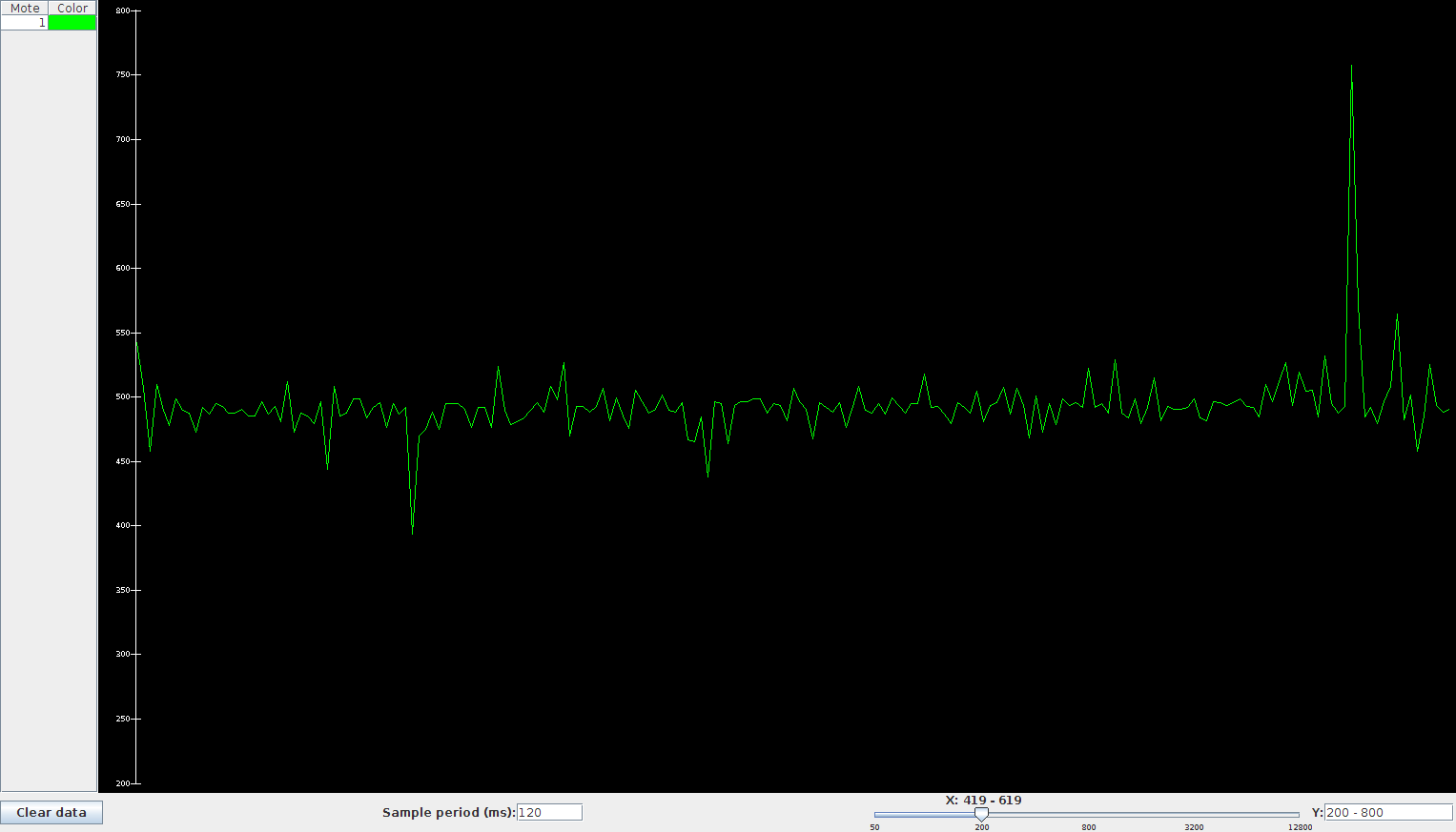
**Hardware:**

1. The hardware used for this project includes
2. MICA2 Motes:- 3 (Sensing Unit, Base-Station, Board with Actuator Unit)
3. MIB 520 Programmer Board
4. MTS 300 Sensor Board
5. MDA 300 CA Data Acquisition Board
6. LEDS
7. Laptop Computer (Running a script to send email)

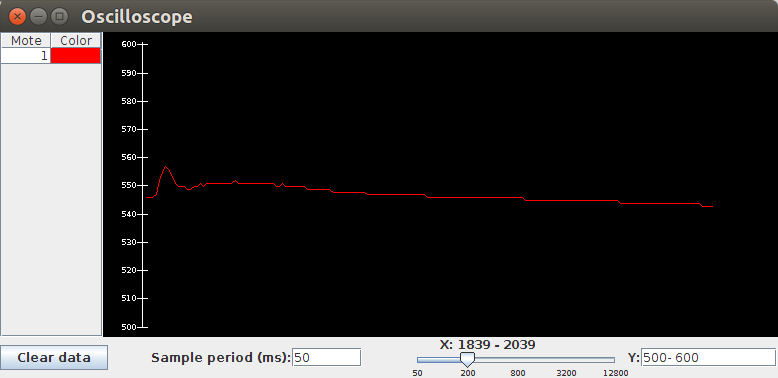
**Readings:** Here are some readings while calibrating three sensors.

Noise Detected

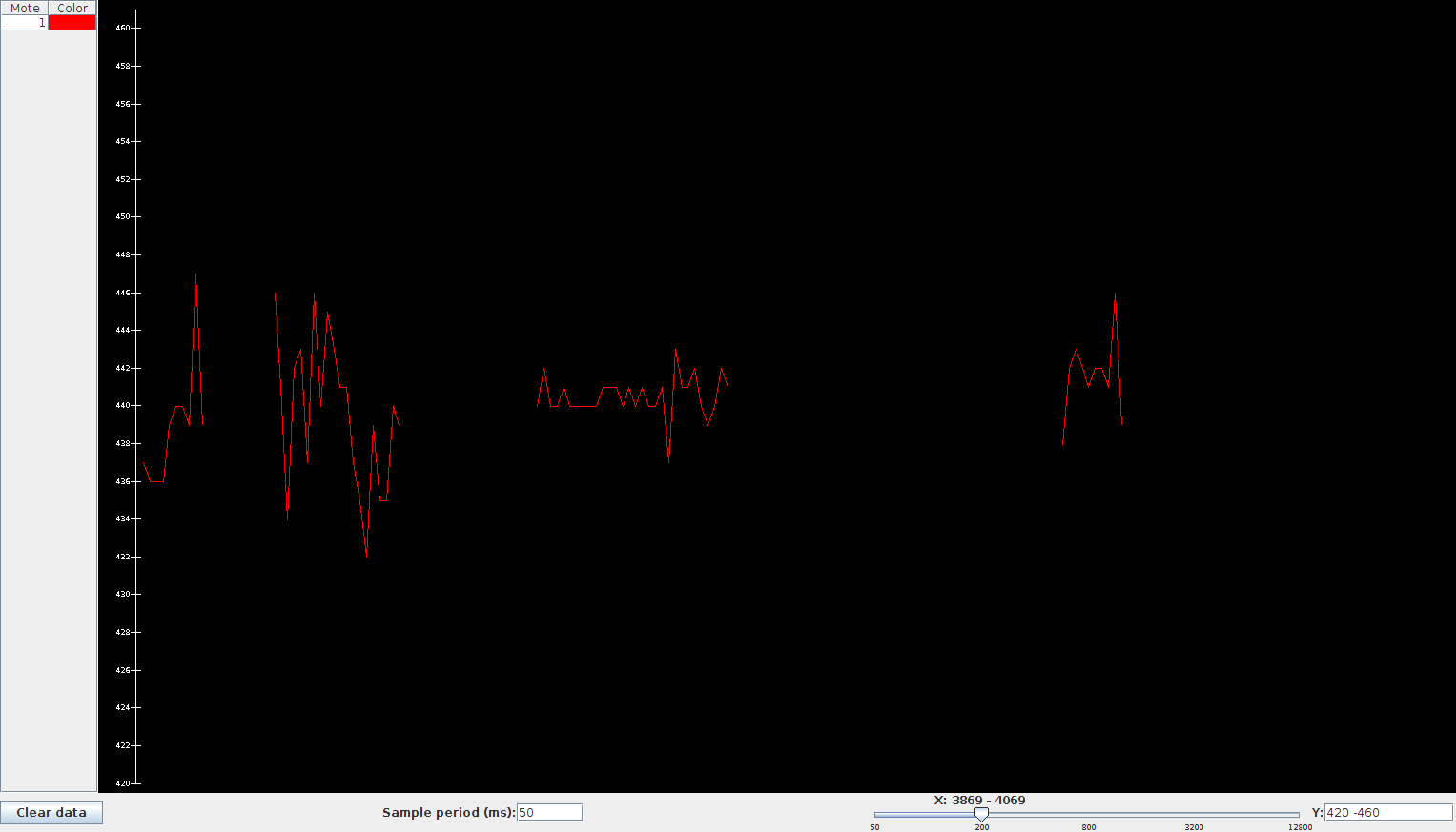
Microphone Sensor:



Temperature Sensor:



Accelerometer Sensor:



**Code Snippets:**

* The following code is edited in OscilloscopeAppC.nc, a configuration file, and is used for mote with sensor board:

configuration OscilloscopeAppC { }

implementation

{

components OscilloscopeC as Osc1,

OscilloscopeC as Osc2,

OscilloscopeC as Osc3,

MainC, ActiveMessageC, LedsC,

new TimerMilliC() as Timer,

new TimerMilliC() as Timer2,

new TimerMilliC() as Timer3,

new AccelXC() as Sensor1, // Accelerometer sensor as Sensor1

new TempC() as Sensor2, // Temperature sensor as Sensor2

new MicC() as Sensor3, // Mic as Sensor3

new AMSenderC(AM\_OSCILLOSCOPE),

new AMReceiverC(AM\_OSCILLOSCOPE),

SounderC; // Buzzer

Osc1.Boot -> MainC;

Osc1.RadioControl -> ActiveMessageC;

Osc1.AMSend -> AMSenderC;

Osc1.Receive -> AMReceiverC;

Osc1.Timer -> Timer;

Osc1.Timer2 -> Timer2;

Osc1.Timer3 -> Timer3;

Osc1.Read -> Sensor1; // Wiring done for sensor

Osc1.Read2 -> Sensor2; // Wiring done for sensor 2

Osc1.Read3 -> Sensor3; // Wiring done for sensor 3

Osc1.Leds -> LedsC;

Osc1.Mts300Sounder -> SounderC; // Wiring done for Sounder i.e. buzzer

}

* The following code is edited into the OscilloscopeC.nc, an implementation file, on the mote with the sensor node.

module OscilloscopeC @safe()

{

uses {

interface Boot;

interface SplitControl as RadioControl;

interface AMSend;

interface Receive;

interface Timer<TMilli> as Timer;

// Interface declared as Read for taking the Accelerometer readings

interface Read<uint16\_t> as Read;

// Interface declared as Read2 for taking the Temperature readings

interface Read<uint16\_t> as Read2;

// Interface declared as Read3 for taking the Mic readings

interface Read<uint16\_t> as Read3;

interface Leds;

interface Mts300Sounder; // interface declared for buzzer

}

}

* The following code is editied in the Timer.fired() function of the implementation file OscilloscopeC.nc

event void Timer.fired()

{

if (reading == NREADINGS) //previous conidtion

{

if(alarmOn)

{

if (!sendBusy && sizeof local <= call AMSend.maxPayloadLength())

{

// Don't need to check for null because we've already checked length

// above

memcpy(call AMSend.getPayload(&sendBuf, sizeof(local)), &local, sizeof local);

if (call AMSend.send(AM\_BROADCAST\_ADDR, &sendBuf, sizeof local) == SUCCESS)

sendBusy = TRUE;

}

if (!sendBusy)

report\_problem();

alarmOn = FALSE;

}

if (!suppressCountChange)

local.count++;

suppressCountChange = FALSE;

}

// Read the Acceleromter, if success toggle the red LED

if (call Read.read() != SUCCESS)

report\_problem();

else

call Leds.led0Toggle(); //Red led, for accelerometer

}

event void Read.readDone(error\_t result, uint16\_t data)

{

if (result != SUCCESS)

/\* Checking if the result is success or error, if error showing the error on leds with report problem \*/

{

data = 0xffff;

report\_problem();

}

else // If result is success, then

{

/\* Checking if reading count is less than NREADINGS, this variable is set to 10 in Oscilloscope.h file \*/

if (reading < NREADINGS)

{ /\* Here we are taking the avg of 10 readings, since accelerometer data varies, we are taking reading and averaging it \*/

/\* Adding the current accelerometer reading to the array ('readings') in local (which is Oscilloscope\_t structure) \*/

local.readings[reading++] = data;

if(reading>1)

// Averaging the accelerometer data avgAccData = (avgAccData + data)/2;

else

avgAccData = data;

}

/\* Checking if reading count is equal to NREADINGS, then we are checking if accCounter reading is in between the set threshold\*/

if(reading == NREADINGS)

{

// If reading is in the set threshold then increasing the counter

if(avgAccData>0x01B0 && avgAccData<0x01BB)

{

/\* accCounter is of type uint8\_t so max value is 255, so put the condition so that the counter won’t overflow \*/

if(accCounter < 255)

accCounter++;

}

else /\* if the average reading of accelerometer is out of the threshold limit, then the car is not stationary and the reseting the accCounter counter and making alarm = false \*/

{

//call Leds.led2Toggle();

accCounter = 0;

alarmOn = FALSE;

}

avgAccData = 0;

}

if(accCounter>2)

/\* If accCounter has reached to the time threshold set for stationary car, then checking the temperature reading \*/

{

if (call Read2.read() != SUCCESS)

/\* Calling command to read the Temperature reading, also if it results in success we are toggling the Green LED \*/

report\_problem();

else

call Leds.led1Toggle();

}

}

}

event void Read2.readDone(error\_t result, uint16\_t data)

{

if (result != SUCCESS)

{

data = 0xffff;

report\_problem();

}

else

{

/\* Monitoring the temperature reading, if it exceeds the set threshold then checking for any activity with the help of mic readings \*/

if(data>0x0202)

{

/\* Calling command to read the Mic reading, also if it results in success we are toggling the Yellow LED \*/

if (call Read3.read() != SUCCESS)

report\_problem();

else

{

call Leds.led2Toggle();

}

}

}

}

event void Read3.readDone(error\_t result, uint16\_t data)

{

if (result != SUCCESS)

{

data = 0xffff;

report\_problem();

}

else

{

/\* Checking if mic reading exceeeds certain amount of threshold limit then sounding the buzzer\*/

if(data<0x01C2 || data>0x0226)

{

// Increase the counter

// Turning ON the buzzer for 100 ms

call Mts300Sounder.beep(100);

/\* If currenly alarm is not ON then, turning on the Alarm and setting the sendEmail flag \*/

if(!alarmOn)

{

alarmOn = TRUE;

sendEmail = TRUE;

}

}

/\* If send email flag is true then sending a signal to base station to trigger the mail, and actuator mote to lower down the windows\*/

if(sendEmail)

{

if (!sendBusy && sizeof local <= call AMSend.maxPayloadLength())

{

/\* Don't need to check for null because we've already checked length above \*/

memcpy(call AMSend.getPayload(&sendBuf, sizeof(local)), &local, sizeof local);

if (call AMSend.send(AM\_BROADCAST\_ADDR, &sendBuf, sizeof local) == SUCCESS) // Calling the command to send the packet to base station and Actuator mote

sendBusy = TRUE;

}

if (!sendBusy)

{

report\_problem();

sendEmail = FALSE; // sendEmail flag = false

}

}

}

}

**Results:**

* We have successfully implemented the in-car heat detection and alarm system using WSN.
* The mote is sensing motion, temperature and sound accurately.
* As the proof of concept we are lighting an LED using MDA data acquisition board, which can be replaced by an action such as sliding down the window for the real life scenarios.
* The base-station performs as a mediator between the Sensing Unit and the computer running the script, which sends an **email**.
* We are using a service called IFTTT in order to send **text message** and as a more reliable alert, **a phone call**.