Project Report on

SMART SENSING TO ASSIST ENERGY MONITORING SYSTEM FOR INDUSTRIAL REFRIGERATION SYSTEMS

Under the guidance of

Takis Zourntos

ESE-4009 Embedded System Design Project

(January 2021 – April 2021)

Submitted By:
Archana Kalathil Venu(C0765532)
Geethu Anna Jacob(C0760834)
Navya Garapathy(C0765780)
Vaibhav Jivani(C0765530)

INDEX

PROJECT CONCEPT

Our project aims to provide on-site real-time measurements of various parameters to a cloud-based system that investigates the energy performance or energy efficiency of Industrial Refrigeration System (IRS). The importance of such an energy monitoring system stems from the global need to save energy, which has got several implications, like:

1. Reducing energy costs
2. Reducing carbon emissions and the environmental damage that they cause.
3. Reduce risk - the more the energy we consume, the greater the risk of an energy crisis or an energy price hike in the very near future.

This project plans to create a consistent technology that every IRS could rely on in monitoring their overall energy usage and to make informed decisions concerning energy efficiency. The system will create timely reports on the overall energy performance and that way, provides feedback on any improvements made through optimized energy saving/conservation technologies. This tool aims to help implement the following:

- Metering our energy consumption and collecting the data. Detailed interval energy consumption data on a weekly or monthly basis makes it possible to see patterns of energy wastage.
- Finding new strategies and opportunities to save energy, and estimating how much energy each strategy could save. Targeting the opportunities to save energy Once we have

identified the opportunities, we need to take actions to target them. This might require investing time and money, say, for upgrading insulation or equipment, etc.

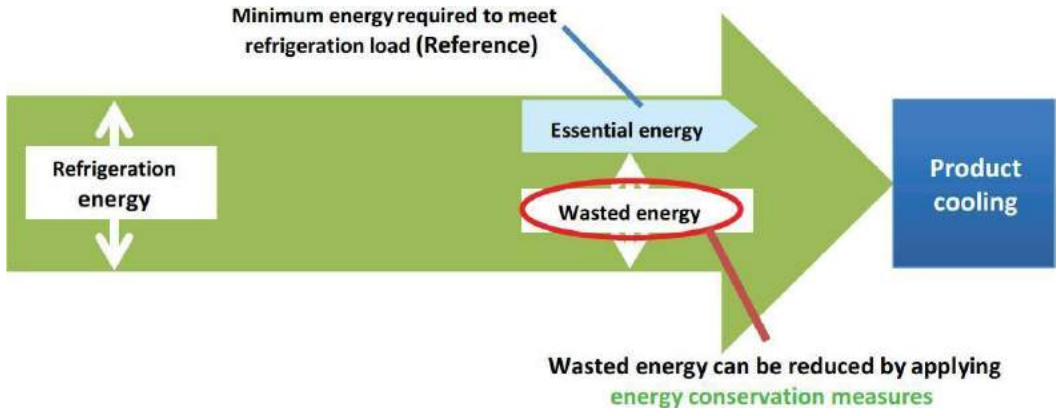
- Tracking our progress at saving energy. This can be done by comparing the past and present energy performance factors in the reports generated and stored in the cloud (with all the gathered data) on a time-to-time basis.

CONCEPTUAL OVERVIEW

Our project essentially consists of an IoT network of sensors, aimed to work in conjunction with the energy performance monitoring system. On-site measurement of several dynamic parameters like temperature, no of people entered, occupancy etc is made with this network of sensors and required information is then sent to the AWS (Amazon Web Services) platform for further processing and storage. The project is especially designed for Refrigerated Distribution facilities, Refrigerated Processing Facilities, Refrigerated Processing Systems and Refrigerated Storage Warehouses.

This real-time data is used for the calculation of various refrigeration loads as well as the total essential energy (this is the minimum energy required by a facility when using the best available technology existing in the market). This total essential energy is later compared with the actual(billed) refrigeration energy to find the Benchmark Energy Factor (BEF), which is a measure of the total energy wastage in the system.

BEF = Overall Refrigeration Energy/ Total Essential Energy


Minimum energy required to meet refrigeration load (Reference)
Refrigeration energy
Essential energy
Wasted energy
Product cooling
Wasted energy can be reduced by applying energy conservation measures

SYSTEM LEVEL ARCHITECTURE

The overall HVAC system in any facility might consist of one or more suction systems (the refrigeration system of a zone, or group of zones, with their associated evaporators that feed refrigeration vapors to an individual or group of compressors). Each zone can either be a cooler or a freezer (the zone is considered to be a freezer if their target temperature is less than -9.4 degree celsius, else a cooler). In addition, the facility has a loading dock area where the products are loaded on and off the trucks for transportation.

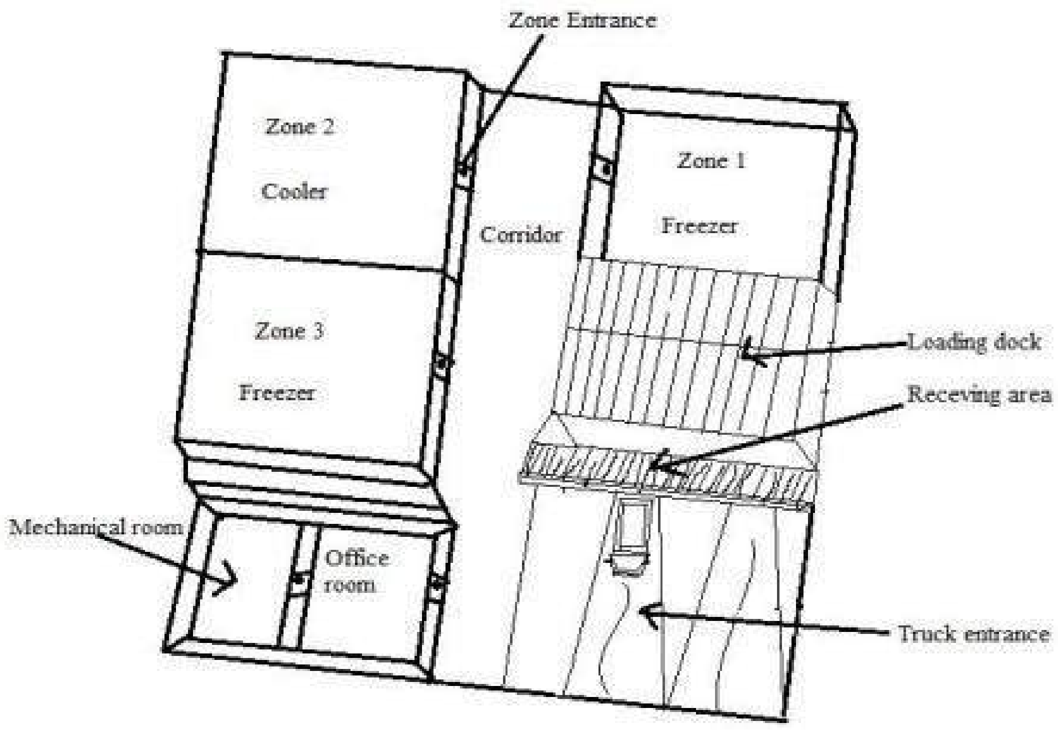


Fig : Typical Industrial Refrigeration facility layout

In this project, our main focus is on measuring the dynamic parameters within a specific zone. For this, we are going to use an IoT (Internet of Things) system based on a network of various sensors for measuring a particular zone's temperature, occupancy, etc. This forms the base layer or the device layer; which is responsible for collecting all the required data. We use several sensors for this purpose as will be described in the following sections. The second level might consist of several lower level microcontroller boards (teensy boards) connected to these devices which aggregate the data, make the conversions, if any. The processed data can either be manipulated for various control operations or can be sent over to the higher level microcontroller through a Bluetooth network. In addition, the higher level microcontroller is connected to a camera, HDMI display and a relay circuitry for lighting control. The camera will help us find the occupancy of the zone. The microcontroller employs computer vision and image processing for counting the number of people entering and exiting from the zone. The time the zone is occupied is also tracked through a software timer running on the board. This third layer acts as a hub and communicates all the gathered data with a cloud database (like Amazon web services, S3). At a further stage of the project, we will build an HDMI display connected to the higher level microcontroller for displaying live data or measurements within the zone using Qt.

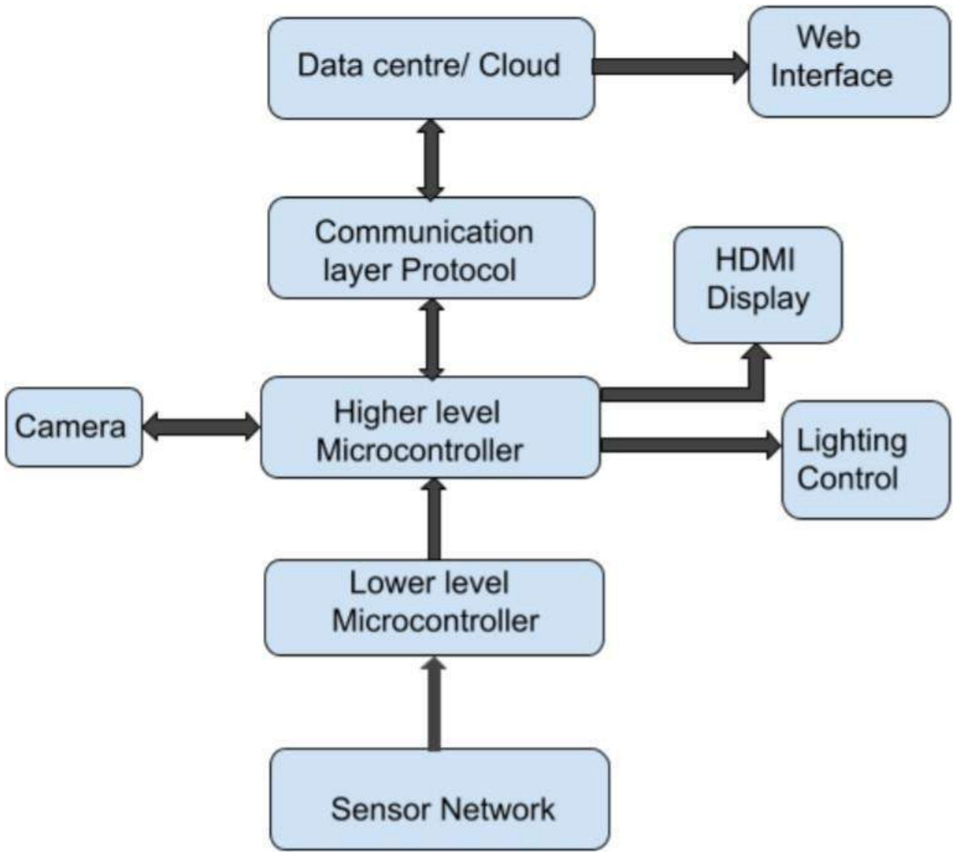

Data centre/ Cloud
Web Interface
Communication layer Protocol
HDMI Display
Camera
Higher level Microcontroller
Lighting Control
Lower level Microcontroller
Sensor Network

Fig : General Block diagram of the project

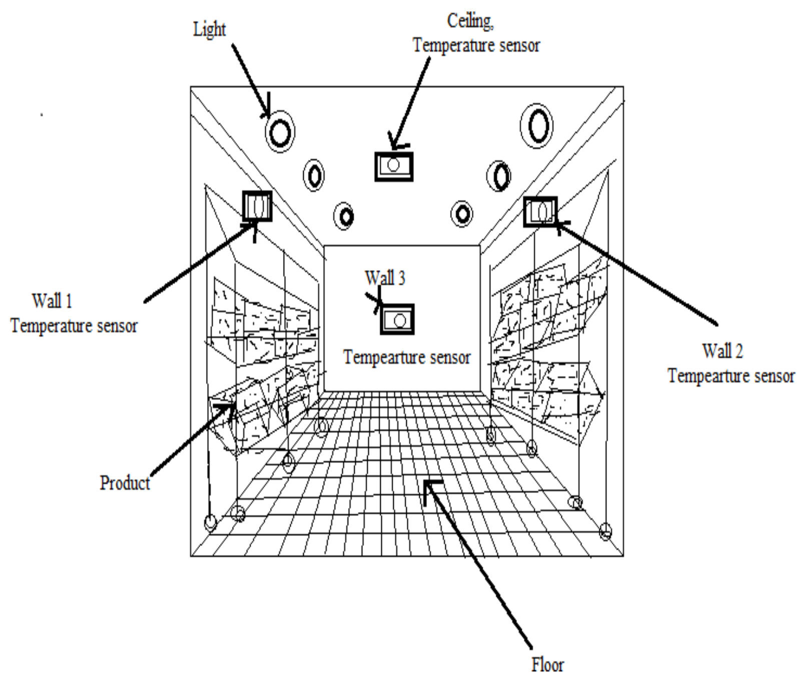


Fig : Interior view of the zone space showing few of the sensors

We employ a temperature sensor to the ceiling of the zone. It is connected to a separate lower level microcontroller board. The door sensor and relay have been connected to the lower level microcontroller, which helps in controlling the zone lightning and sensing door status. The camera is mounted to the higher level microcontroller unit, to count the number of people entering the zone. Furthermore the display is connected to the same unit as well to display the real time data.

We also try to predict the failure or lower efficiency of the evaporator fan motors operating in the zone by recording their sound. We used a microphone sensor to record live evaporator fan motor running sound. This is also interfaced via one among the wall mounted lower level microcontrollers, which then compares this sound profile with the existing database (it contains different sound profiles for normal motor running conditions as well as for motor running at lower efficiencies or faulty conditions). Once any variation in the existing amplitude or frequency is observed, the microcontroller board detects it. It then alerts the central hub

(Higher level microcontroller) which then creates an alert on the display.

Also, a door sensor will be placed near the zone door. If the motion sensor detects a human in its range, it will automatically turn on the lights in the zone. As the lights turn on, the ceiling mounted camera starts recording and the microcontroller connected to it will process the images and counts the number of people entering and exiting the zone space.

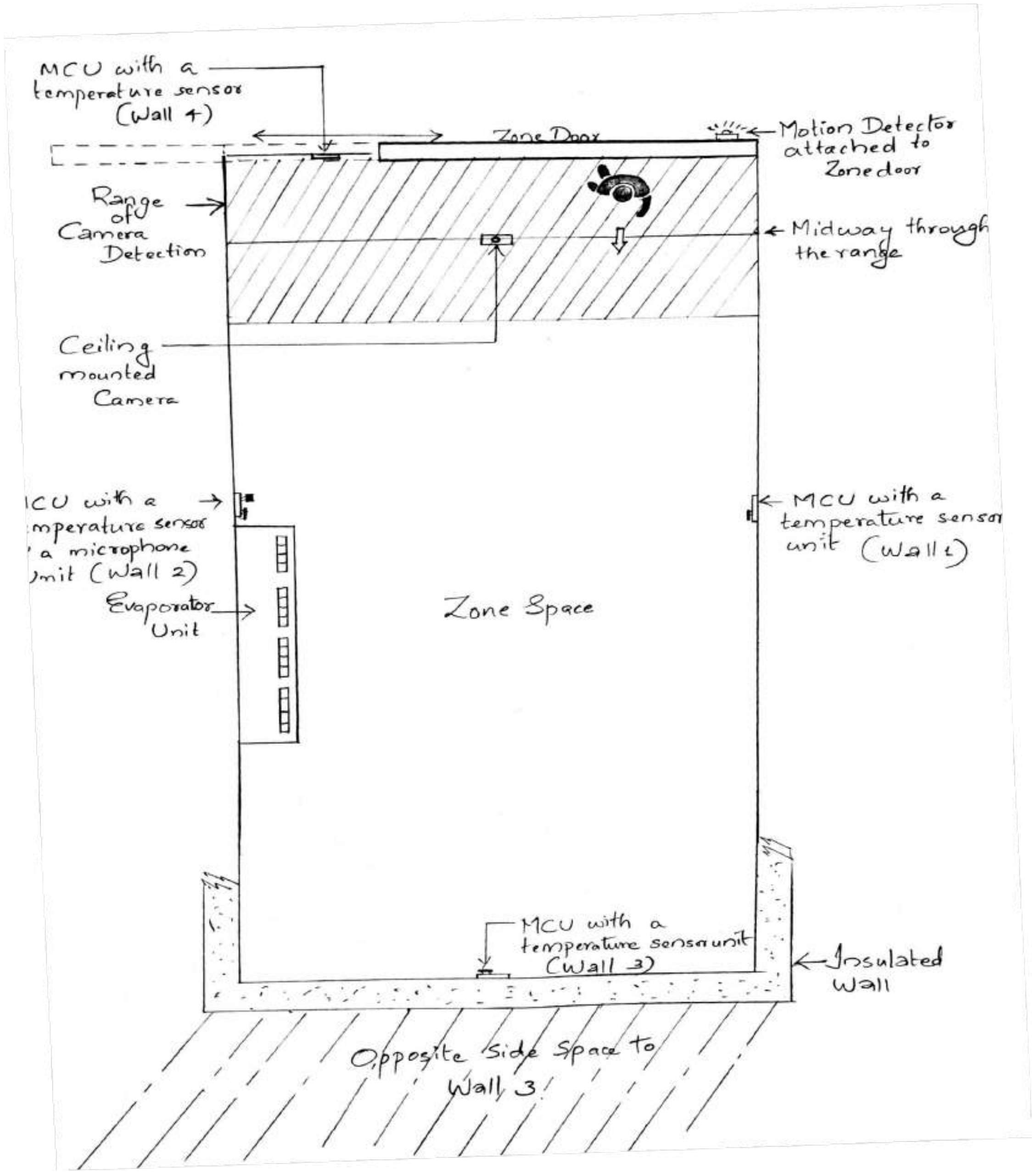

MCU with a temperature sensor (Wall 4)
Zone Door
Motion Detector attached to Zonedoor
Range of Camera Detection
Midway through the range
Ceiling mounted Camera
CU with a mperature sensor a microphone Unit (Wall 2)
MCU with a temperature sensor unit (Wall 1)
Evaporator Unit
Zone Space
MCU with a temperature sensor unit (Wall 3)
Insulated Wall
Opposite Side Space to Wall 3

Fig: Top view of the zone

The HDMI display shows the real-time measurements in the zone at any given time. The following data may be displayed :

- The current temperature of the zone
- The number of people in the zone at the time
- The total hours of occupancy till the time
- Whether the evaporators for the zone are running fine or not.

This helps to ensure consistency in temperature all over the zone at all times.

HARDWARE DESIGN

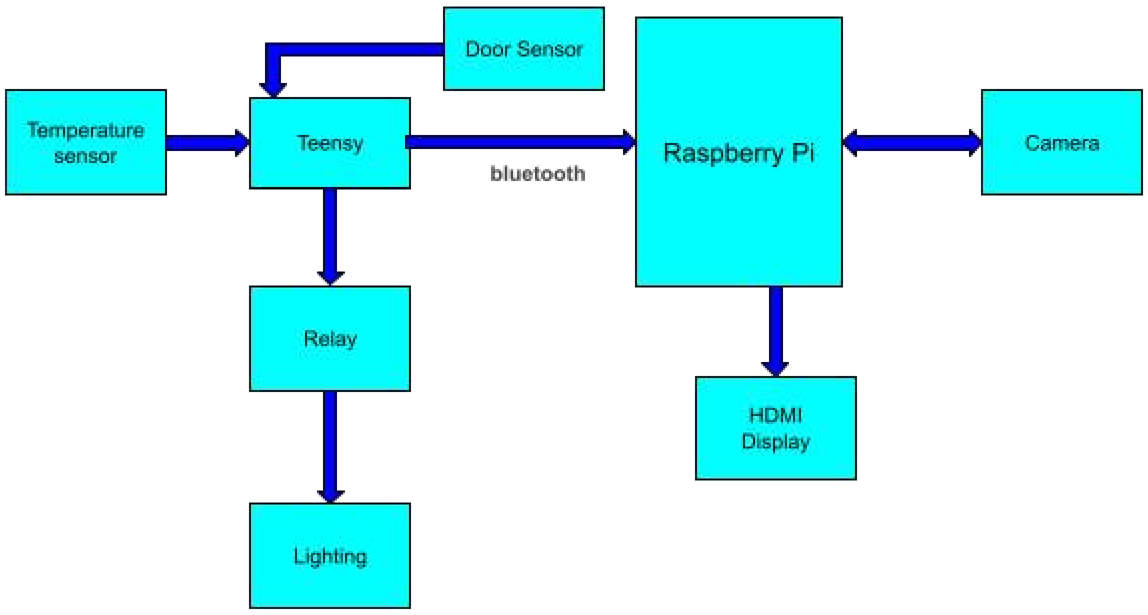


Fig : Hardware design for the project

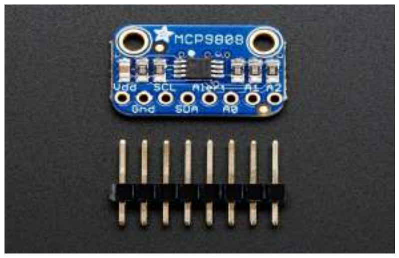
We are using a temperature sensor fixed on the ceiling. There is a microphone sensor fixed to the wall near the motors so that it can record the motor sound and we can check for any abnormalities in the motor operating conditions.

Each of the sensors are interfaced to the teency board via i2c protocol. All the teency boards are then wirelessly connected to the Raspberry pi 4, which act as the central hub through a bluetooth mesh network.

A USB camera mounted on the ceiling near the door is also interfaced to the Raspberry pie to keep track of the number of people entering and exiting the zone. A door sensor at the door triggers the Teensy. Which then Teensy communicates with the Pi via bluetooth in case of any human intervention into the zone and the Pi then controls the lighting inside the zone via a relay circuitry.

Required Components

- Temperature sensor



We are using MCP9808 high accuracy temperature sensor from adafruit. It has a typical accuracy of ±0.25°C over the sensor's -40°C to +125°C range and precision of +0.0625°C. They work great with any microcontroller using standard i2c. Its voltage range is between 2.7v to 5.5v and its operating current is 200 micro A.

- Teeny - Version 4.0

- Raspberry pi 4

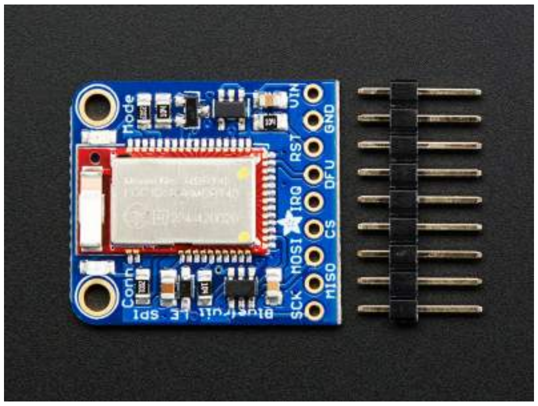
The Raspberry Pi 4 Model B is the newest Raspberry Pi computer made. It has an updated 64-bit quad core processor running at 1.5GHz with built-in metal heatsink, USB 3 ports, dual-band 2.4GHz and 5GHz wireless LAN, faster Gigabit Ethernet, and PoE capability via a separate PoE HAT. It is available with 2GB or 4GB RAM. It has an operating temperature between 0 degree to 50 degree celsius. It has an input voltage of 5V dc via usb C connector as well as through GPIO pins.

- Door Sensor

The sensor is essentially a reed switch, encased in an ABS plastic shell. Normally the reed is 'open' (no connection between the two wires). The other half is a magnet. When the magnet is less than 13mm (0.5") away, the reed switch closes. Rated current is about 100 mA max and voltage is 200 VDC max and distance upto 15mm max.

- Bluetooth module

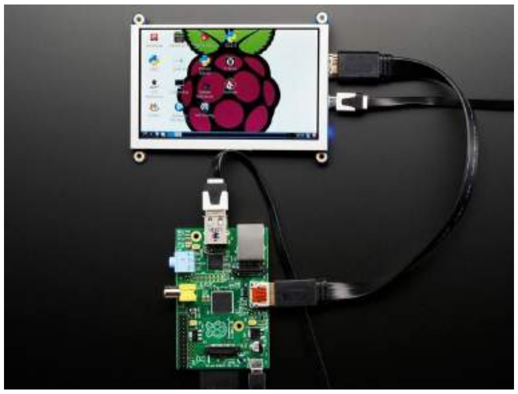
We are using Bluefruit LE SPI friend (BLE) from Adafruit. It makes it easy to add Bluetooth Low Energy connectivity to anything with 4 or 5 GPIO pins. It comes with ARM Cortex M0 core processor running at 16MHz.It has 256KB flash memory and 32KB SRAM. Data transporting via SPI at up to 4MHz clock speed.It also has 5V-safe and with On-board 3.3V voltage regulation.



- USB Camera

- HDMI Display

The HDMI display from adafruit. We can power the display from a usb cable. With the default 5" 800x480 display and 50mA backlight current, the current draw is 500mA total. It is possible to reduce that down 370mA by running the backlight at half-brightness (25mA). This display is not a touch screen.



SOFTWARE DESIGN

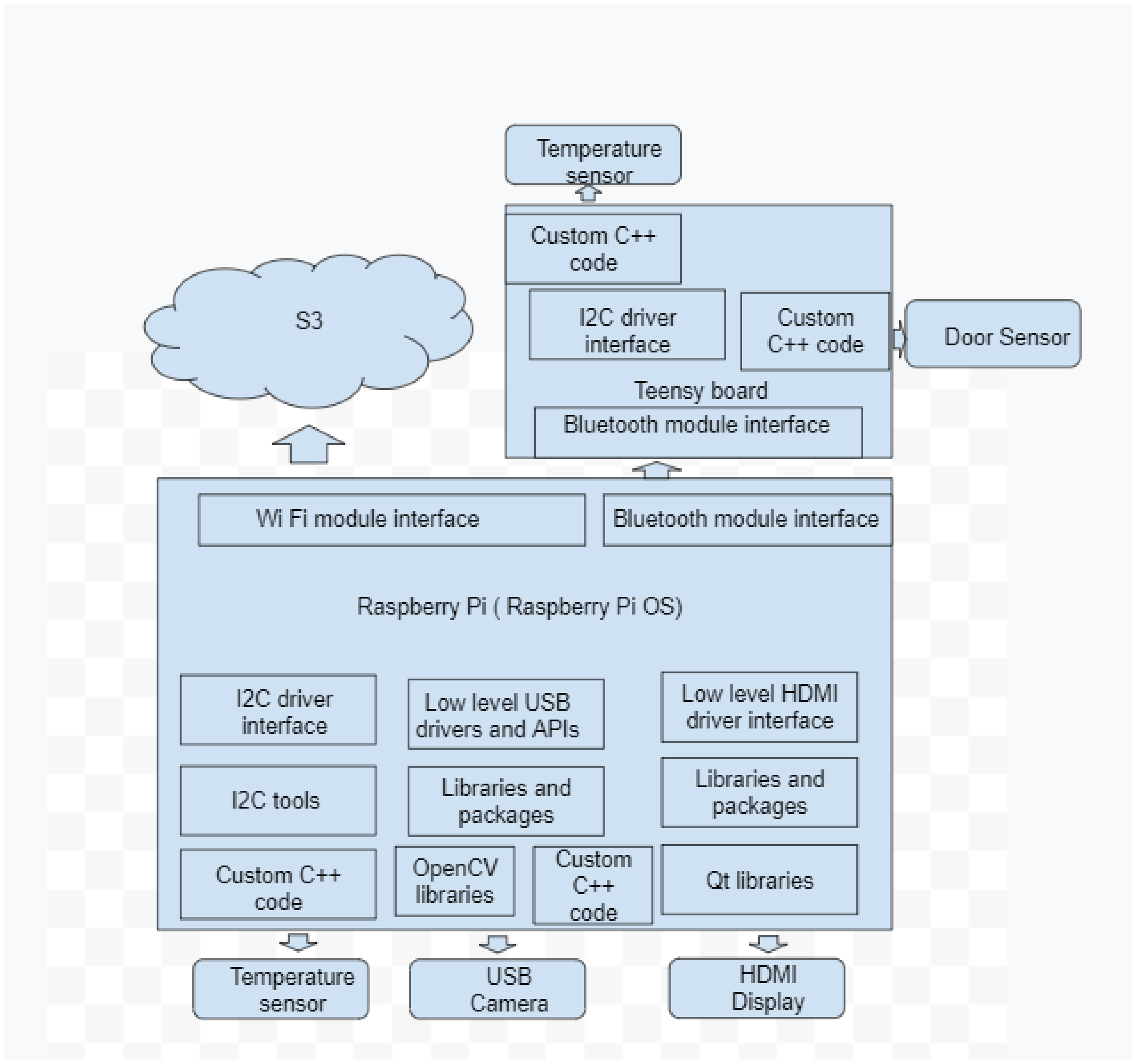
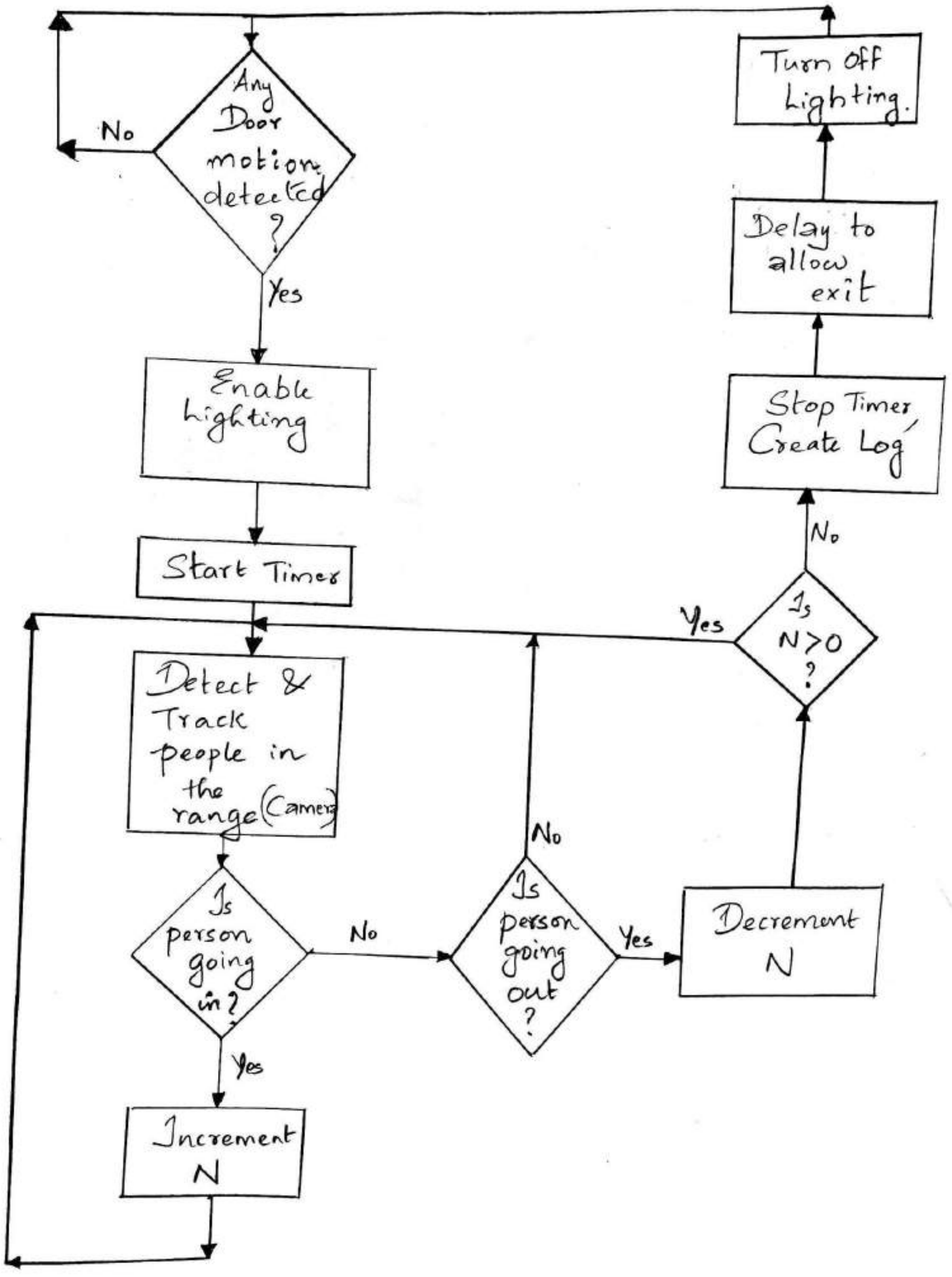


Fig : Software design of the project

In the project, we make use of the I2C serial interface for communicating with the temperature sensor, microphone as well as the door sensor. For keeping track of the average number of people in the zone as well as the time for which the zone is occupied, we make use of a camera controlled by the raspberry Pi and OpenCV object detection and tracking algorithms based on C++. The camera is chosen to provide more accurate information compared to all other people's counter technologies.



Initially, the door sensor attached to the zone door acts as a trigger to wake the teensy up from its sleep mode. Then teensy will communicate to the Pi via bluetooth, once any door opening of the zone is detected. It will control the relay to turn on the lighting in the zone, so that the camera can now start detecting and tracking persons. A software timer is also started to keep track for the time the zone is occupied.

For counting, we leverage both the object detection as well as object tracking algorithms. We might need to download several libraries including OpenCV for this purpose. Since object detection is a fairly computationally intensive process, we do that only every N frames to check if a new object has entered the range of detection or not. Then for the rest of the time, we track the object based on its previous coordinates and determine the direction of motion.

The above flowchart explains the basic algorithm used for counting. Here, N gives the number of people inside the zone at any given time. We update the value of N after every hour (or any time frame of interest). At the end of each day, will average out the total N over 24 to get the average hourly N for the day.

We will start a timer as soon as someone enters the zone and will keep the timer running until N=0 inside the zone. This log is stored and is later on totalled up to get the total number of hours the zone gets occupied. Once the Pi detects N to be zero, it will send control signals to disable the lighting and in itself will go into a low power mode.

The acoustic signals recorded via the microphone will be sent over to the AWS cloud through the raspberry Pi. The signals could be filtered or transformed (by applying algorithms to enhance the result) to identify any differences in the frequency spectrum of the sound signal. The signal may then be sent over to the AWS cloud every hour (or any time frame of interest). A cloud-based motor condition monitoring system is then implemented where a machine learning algorithm is trained with the previously collected data to filter out the signal and then to analyze its frequency components. There might be sufficient information not only to distinguish between healthy and faulty states, but also to determine the specific fault.But we had a trouble while transferring data from microphone to cloud. Some false data has been uploaded. So, in future we would like to rectify this issue by studying more detail about it.

GANTT CHART

Gantt chart represents overall progress and planning of the project. For the project, we have used ClickUp application which has a lot of features and is very organised for project planning. The following link redirects to the chart.

https://share.clickup.com/g/h/84x5n-28/13e82c343e6a032

DEMO-DAY EXPERIENCE

Vaibhav: “ 4 Months ! 103 Days !! 2472 Hours !!! 8899200 Minutes !!!! All this time of hard work was presented in just 12 minutes of presentation! All of our hard work and Archana’s stewardship paid off. Until the last moment we were working on the project, hoping to do a few more new upgrades in the coding part for fun as we knew what changes to make in detail. I was a little worried that things might go south. Fortunately, everything went flawless. I was a little more confident when Takis acknowledged our work. We’re all thankful to Takis for being there when we needed him to guide us. “

Archana: Even if the project seemed to be a challenge for us from the beginning, the learnings from it were countless, especially the project management ones and troubleshooting skills. At some point, I felt like we were made capable to do it by Prof. Takis, with his comprehensive guidance which started right from our first semester in this course. We are so lucky and grateful to have been mentored by him. Even if my personal overall experience was not so easy, the end results were really rewarding. During the demo day, it was a real pleasure to answer our junior’s questions and share our experiences.

APPENDIX

SCHEMATICS

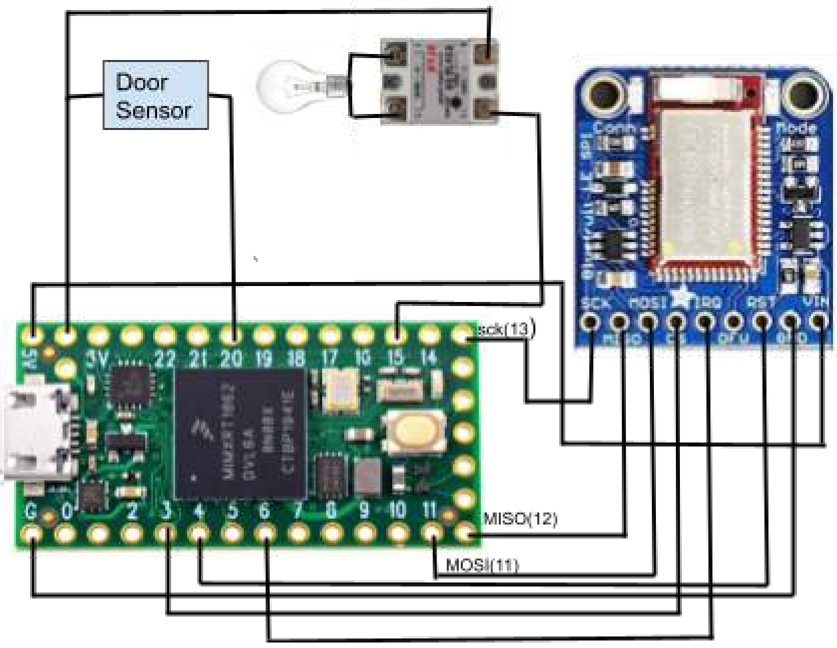


fig.Teensy setup with door sensor and relay

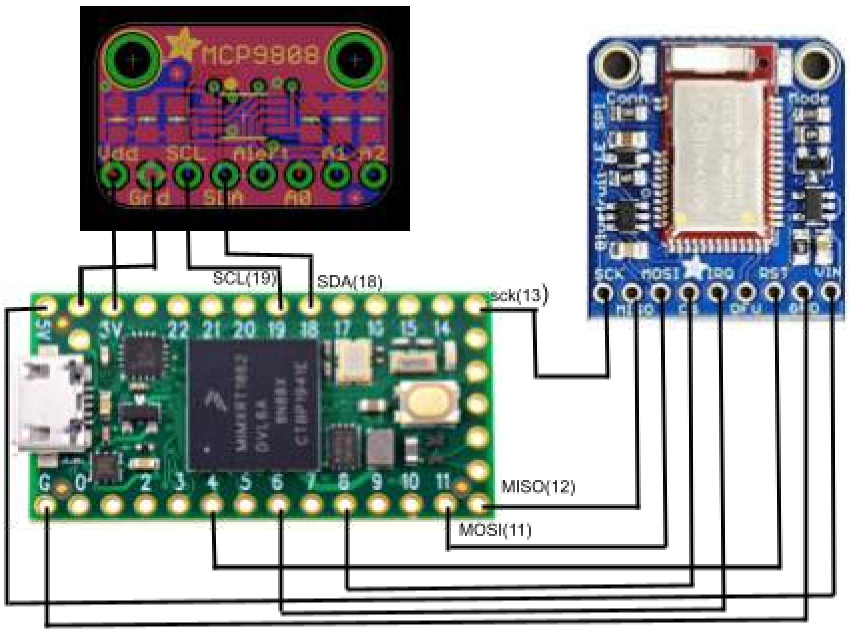


Fig. Teensy setup with temperature sensor

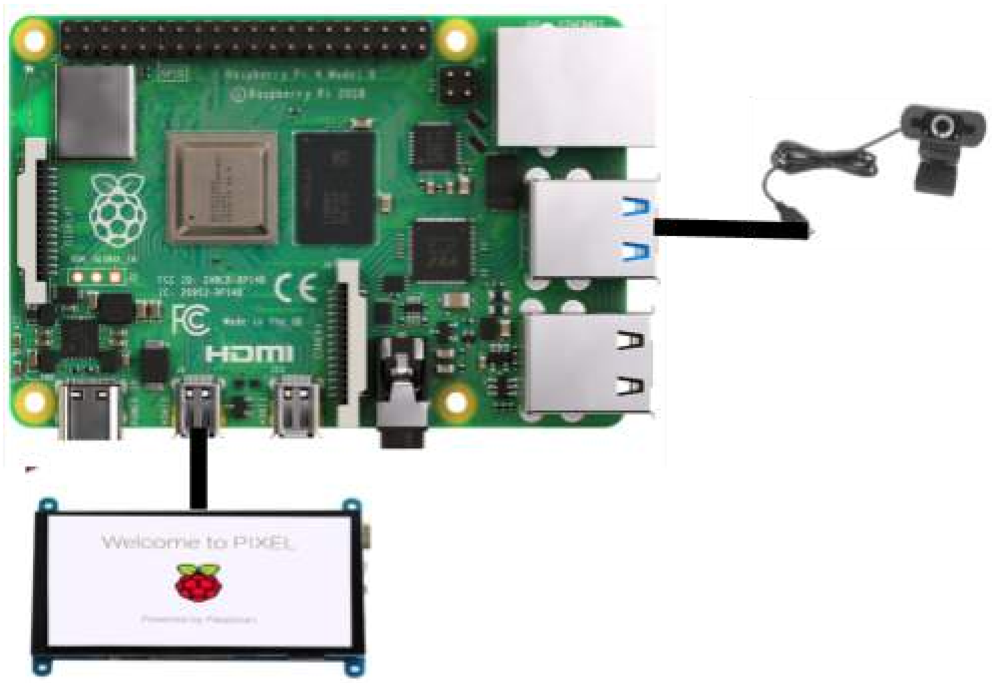


Fig. Raspberry Pi connected with hdmi display and usb camera

CODE ON TEENSY :

BluefruitConfig.h : Configuration file

```
      // COMMON SETTINGS
// ------------------------------------------------------------
------------------------------
// These settings are used in both SW UART, HW UART and SPI
mode
// ------------------------------------------------------------
------------------------------
#define BUFSIZE                        160   // Size of the
read buffer for incoming data
#define VERBOSE_MODE                   true  // If set to
'true' enables debug output

*/
// SHARED SPI SETTINGS
// ------------------------------------------------------------
------------------------------
// The following macros declare the pins to use for HW and SW
SPI communication.
// SCK, MISO and MOSI should be connected to the HW SPI pins on
the Uno when
// using HW SPI.  This should be used with nRF51822 based
Bluefruit LE modules
// that use SPI (Bluefruit LE SPI Friend).
// ------------------------------------------------------------
------------------------------
#define BLUEFRUIT_SPI_CS               3
#define BLUEFRUIT_SPI_IRQ              7
#define BLUEFRUIT_SPI_RST              4    // Optional but
recommended, set to -1 if unused
```

bleuart_datamode_MCP9808.ino : Code to read temperature data from MCP9808 module and send it to Pi over BLE module

```
     /***********************************************************
***********
 This is an example for our nRF51822 based Bluefruit LE modules

 Pick one up today in the adafruit shop!

 Adafruit invests time and resources providing this open source
code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 MIT license, check LICENSE for more information
 All text above, and the splash screen below must be included in
 any redistribution
****************************************************************
*****/

#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include "Adafruit_BLE.h"
#include "Adafruit_MCP9808.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"
#include <Audio.h>


AudioInputI2S            i2s1;
AudioRecordQueue         queue1;         //xy=281,63
AudioConnection          patchCord1(i2s1, 0, queue1, 0);

#if SOFTWARE_SERIAL_AVAILABLE
  #include <SoftwareSerial.h>
```

```
#endif

// Create the MCP9808 temperature sensor object
Adafruit_MCP9808 tempsensor = Adafruit_MCP9808();

/*=================================================================
===========
    APPLICATION SETTINGS

    FACTORYRESET_ENABLE       Perform a factory reset when
running this sketch

                              Enabling this will put your
Bluefruit LE module
                              in a 'known good' state and clear
any config
                              data set in previous sketches or
projects, so
                              running this at least once is a
good idea.

                              When deploying your project,
however, you will
                              want to disable factory reset by
setting this
                              value to 0.  If you are making
changes to your
                              Bluefruit LE device via AT
commands, and those
                              changes aren't persisting across
resets, this
                              is the reason why.  Factory reset
will erase
                              the non-volatile memory where
config data is
                              stored, setting it back to factory
default
                              values.

                              Some sketches that require you to
```

```
bond to a
                              central device (HID mouse,
keyboard, etc.)
                              won't work at all with this
feature enabled
                              since the factory reset will clear
all of the
                              bonding data stored on the chip,
meaning the
                              central device won't be able to
reconnect.
    MINIMUM_FIRMWARE_VERSION  Minimum firmware version to have
some new features
    MODE_LED_BEHAVIOUR        LED activity, valid options are
                              "DISABLE" or "MODE" or "BLEUART"
or
                              "HWUART"  or "SPI"  or "MANUAL"

----------------------------------------------------------------
-------*/
    #define FACTORYRESET_ENABLE         1
    #define MINIMUM_FIRMWARE_VERSION    "0.6.6"
    #define MODE_LED_BEHAVIOUR          "MODE"
/*==============================================================
===========*/


/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and
then user selected CS/IRQ/RST */
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS,
BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
}

/***************************************************************
***********/
```

```
/*!
    @brief  Sets up the HW an the BLE module (this function is
called
            automatically on startup)
*/
/**************************************************************
***********/
void setup(void)
{
  while (!Serial);  // required for Flora & Micro
  delay(500);
  Serial.begin(115200);

  Serial.println(F("Adafruit Bluefruit Data Mode Example
modified to send temperature data"));

Serial.println(F("-----------------------------------------------
------------------------------"));

  /* Initialise the module */
  Serial.print(F("Initialising the Bluefruit LE module: "));

  if ( !ble.begin(VERBOSE_MODE) )
  {
    error(F("Couldn't find Bluefruit, make sure it's in CoMmanD
mode & check wiring?"));
  }
  Serial.println( F("OK!") );

  if ( FACTORYRESET_ENABLE )
  {
    /* Perform a factory reset to make sure everything is in a
known state */
    Serial.println(F("Performing a factory reset: "));
    if ( ! ble.factoryReset() ){
      error(F("Couldn't factory reset"));
    }
  }

  /* Disable command echo from Bluefruit */
  ble.echo(false);

  Serial.println("Requesting Bluefruit info:");
  /* Print Bluefruit information */
  ble.info();
  ble.verbose(false);  // debug info is a little annoying after
this point!

  /* Wait for connection */
  while (! ble.isConnected()) {
      delay(500);
  }

  if (!tempsensor.begin(0x18)) {
    Serial.println("Couldn't find MCP9808! Check your
connections and verify the address is correct.");
    while (1);
  }

   Serial.println("Found MCP9808!");

 tempsensor.setResolution(3); // sets the resolution mode of
reading, the modes are defined in the table below:
  // Mode Resolution SampleTime
  //  0    0.5°C       30 ms
  //  1    0.25°C      65 ms
  //  2    0.125°C     130 ms
  //  3    0.0625°C    250 ms

  Serial.println(F("******************************"));

  // LED Activity command is only supported from 0.6.6
  if ( ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION) )
  {
    // Change Mode LED Activity
    Serial.println(F("Change LED activity to "
MODE_LED_BEHAVIOUR));
    ble.sendCommandCheckOK("AT+HWModeLED=" MODE_LED_BEHAVIOUR);
  }
  // Set module to DATA mode
  Serial.println( F("Switching to DATA mode!") );
  ble.setMode(BLUEFRUIT_MODE_DATA);

  Serial.println(F("******************************"));

}


/*****************************************************************
***********/
/*!
    @brief  Constantly poll for new command or response data
*/
/*****************************************************************
***********/
void loop(void)
{
  /************************** TEMPERATURE DATA
*************************************/

    char inputs[BUFSIZE+1] = "T "; // size of inputs : 161
bytes

    // Read out the temperature.
    float temp_c = tempsensor.readTempC();
    char temp[10];
    sprintf(temp, "%f", temp_c);
    strcat(inputs, temp);

    // Send characters to Bluefruit
    Serial.println(inputs);

    // Send input data to host via Bluefruit
    ble.println(inputs);

   //wait 200 milliseconds
    delay(200);

}
```

rx_bleuart_datamode.ino : Code to control lighting based on input from Pi through BLE module

```
     /*********************************************************************
 This is an example for our nRF51822 based Bluefruit LE modules
 Pick one up today in the adafruit shop!
 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!
 MIT license, check LICENSE for more information
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

#include <Arduino.h>
#include <SPI.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"

#if SOFTWARE_SERIAL_AVAILABLE
  #include <SoftwareSerial.h>
#endif

/*=========================================================================
    APPLICATION SETTINGS
    FACTORYRESET_ENABLE       Perform a factory reset when
running this sketch

                              Enabling this will put your
Bluefruit LE module

                              in a 'known good' state and clear
any config

                              data set in previous sketches or
projects, so

                              running this at least once is a
good idea.

                              When deploying your project,
however, you will

                              want to disable factory reset by
setting this

                              value to 0.  If you are making
changes to your

                              Bluefruit LE device via AT
commands, and those

                              changes aren't persisting across
resets, this

                              is the reason why.  Factory reset
will erase

                              the non-volatile memory where
config data is

                              stored, setting it back to
factory default

                              values.

                              Some sketches that require you to
bond to a

                              central device (HID mouse,
keyboard, etc.)

                              won't work at all with this
feature enabled

                              since the factory reset will
clear all of the

                              bonding data stored on the chip,
meaning the
                              central device won't be able to
reconnect.
    MINIMUM_FIRMWARE_VERSION  Minimum firmware version to have
some new features
    MODE_LED_BEHAVIOUR        LED activity, valid options are
                              "DISABLE" or "MODE" or "BLEUART"
or
                              "HWUART"  or "SPI"  or "MANUAL"

---------------------------------------------------------------
--------*/
    #define FACTORYRESET_ENABLE         1
    #define MINIMUM_FIRMWARE_VERSION    "0.6.6"
    #define MODE_LED_BEHAVIOUR          "MODE"
/*=============================================================
============*/

// Create the bluefruit object

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and
then user selected CS/IRQ/RST */
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS,
BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);


// A small helper
void error(const __FlashStringHelper*err) {
  //Serial.println(err);
  while (1);
}

// pins to use
bool lights_on = 0;
bool door_closed = 1;
const int DOOR_SENSOR_PIN = 20; // teensy pin connected to door
sensor's pin
const int RELAY_PIN       = 15;  // teensy pin connected to the
IN pin of relay
bool doorState;
```

```
/**************************************************************
************/
/*!
    @brief  Sets up the HW an the BLE module (this function is
called
            automatically on startup)
*/
/**************************************************************
************/
void setup(void)
{
  while (!Serial);  // required for Flora & Micro
  delay(500);

  // setting pin modes
  pinMode(DOOR_SENSOR_PIN, INPUT_PULLUP); // set teensy pin to
input pull-up mode
  pinMode(RELAY_PIN, OUTPUT);

  Serial.begin(115200);
  Serial.println(F("Adafruit Bluefruit Command <-> Data Mode
Example"));

Serial.println(F("---------------------------------------------
---"));

  /* Initialise the module */

  if ( !ble.begin(VERBOSE_MODE) )
  {
    error(F("Couldn't find Bluefruit, make sure it's in CoMmanD
mode & check wiring?"));
  }

  if ( FACTORYRESET_ENABLE )
  {
    /* Perform a factory reset to make sure everything is in a
known state */
    Serial.println(F("Performing a factory reset: "));
```

```
      if ( ! ble.factoryReset() ){
        error(F("Couldn't factory reset"));
      }
    }

    /* Disable command echo from Bluefruit */
    ble.echo(false);

    Serial.println("Requesting Bluefruit info:");
    /* Print Bluefruit information */
    ble.info();
    ble.verbose(false);  // debug info is a little annoying after
  this point!

    doorState = digitalRead(DOOR_SENSOR_PIN); // read state
    // wait until door gets closed
    while(doorState)
    {
        delay(100);
        doorState = digitalRead(DOOR_SENSOR_PIN); // read state
    }

  }

  /***************************************************************
  ************/
  /*!
      @brief  Constantly poll for new command or response data
  */
  /***************************************************************
  ************/
  void loop(void)
  {

      if(!lights_on)
      {
          doorState = digitalRead(DOOR_SENSOR_PIN); // read state
          // wait until door gets opened
          while(!doorState)
          {
            delay(100);
            doorState = digitalRead(DOOR_SENSOR_PIN); // read
state
          }
          digitalWrite(RELAY_PIN, HIGH);
          lights_on = 1;
      }
      else
      {
          /* Wait for connection */
        while (! ble.isConnected())
        {
        delay(500);
        }

        // LED Activity command is only supported from 0.6.6
        if ( ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION) )
        {
          // Change Mode LED Activity
          ble.sendCommandCheckOK("AT+HWModeLED="
MODE_LED_BEHAVIOUR);
        }

        ble.setMode(BLUEFRUIT_MODE_DATA);
          // Wait for data to become available
        while(!ble.available())
        {
          delay(100);
        }

          char c = ble.read();
               // Sample character sent from Pi to indicate
people count is zero
          if(c == 'a')
          {
              // wait for the door to get closed; so that lights
get turned off only after everyone exiting the zone
              doorState = digitalRead(DOOR_SENSOR_PIN);
              while(doorState)
              {
```

```
                        delay(100);
                        doorState = digitalRead(DOOR_SENSOR_PIN); //
read state
                    }
                        // turn OFF lights
                    lights_on = 0;
                    delay(2000);
                    digitalWrite(RELAY_PIN, LOW);
                }

            }
        }
```

CODE ON RASPBERRY PI

ble_scan.c : Code to receive temperature data from teensy through BLE

```
     #include <pthread.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/queue.h>
#include <time.h>
#include <string.h>

#include "gattlib.h"

#define BLE_SCAN_TIMEOUT   4

typedef void (*ble_discovered_device_t)(const char* addr, const
char* name);

// We use a mutex to make the BLE connections synchronous
static pthread_mutex_t g_mutex = PTHREAD_MUTEX_INITIALIZER;

LIST_HEAD(listhead, connection_t) g_ble_connections;
struct connection_t {
    pthread_t thread;
    char* addr;
    LIST_ENTRY(connection_t) entries;
};

// function to create a substring from a string
void substring(char s[], char sub[], int p, int l){
    int c = 0;
    while ( c < l) {
      sub[c] = s[p+c-1];
      c++;
    }
    sub[c] = '\0';
}

void delay(int number_of_seconds)
{
     // Converting time into milli_seconds
     int milli_seconds = 1000 * number_of_seconds;

     // Storing start time
     clock_t start_time = clock();

     // looping till required time is not achieved
     while (clock() < start_time + milli_seconds);
}

 void *ble_connect_device(void *arg) {
    struct connection_t *connection = arg;
    char* addr = connection->addr;
    gatt_connection_t* gatt_connection;
    gattlib_primary_service_t* services;
    gattlib_characteristic_t* characteristics;
    int services_count, characteristics_count;
    char uuid_str[MAX_LEN_UUID_STR + 1];
    int ret, i;
    size_t len;

    pthread_mutex_lock(&g_mutex);

    printf("------------START %s ---------------\n", addr);
```

```
      // Connecting to the device
    gatt_connection = gattlib_connect(NULL, addr,
GATTLIB_CONNECTION_OPTIONS_LEGACY_DEFAULT);

    if (gatt_connection == NULL) {
      fprintf(stderr, "Failed to connect to the bluetooth
device.\n");
      goto connection_exit;
    } else {
      puts("Succeeded to connect to the bluetooth device.");
    }

// Connecting to characteristics of the service
    ret = gattlib_discover_char(gatt_connection,
&characteristics, &characteristics_count);
    if (ret != 0) {
      fprintf(stderr, "Fail to discover characteristics.\n");
      goto disconnect_exit;
    }

    for (i = 0; i < characteristics_count; i++) {
      gattlib_uuid_to_string(&characteristics[i].uuid,
uuid_str, sizeof(uuid_str));

      // Finding the uart rx characteristics of the device
(found using bluetoothctl tool)
   if(strcmp(uuid_str, "6e400003-b5a3-f393-e0a9-e50e24dcca9e")
== 0)
   {
      printf("characteristic[%d] properties:%02x
value_handle:%04x uuid:%s\n", i, characteristics[i].properties,
characteristics[i].value_handle, uuid_str);

     float temp;
     char first_string[5];
     char second_string[15];

      uint8_t* buffer = NULL;

      // Wait until we receive data on the characteristics
```

```
        do{
            ret = gattlib_read_char_by_uuid(gatt_connection,
&characteristics[i].uuid, (void **)&buffer, &len);
          }while(buffer[0] == '\0');

        if (ret != GATTLIB_SUCCESS) {

            if (ret == GATTLIB_NOT_FOUND) {
               fprintf(stderr, "Could not find GATT
Characteristic with UUID %s. "
               "You might call the program with
'--gatt-discovery'.\n", uuid_str);
            }
                 else {
               fprintf(stderr, "Error while reading GATT
Characteristic with UUID %s (ret:%d)\n", uuid_str, ret);
               free(buffer);
            }
             goto disconnect_exit;

          }

        substring(buffer, first_string, 1, 2);
        substring(buffer, second_string, 3, len-4);
        FILE* t_ptr = fopen("temperature.txt", "a");
        if (t_ptr == NULL)
        {
              printf("\nError creating file!!");
              exit(1);
        }
        if( strcmp(first_string, "T ") == 0)
        {
              temp = atof(second_string);
               printf(t_ptr, "%f\n", temp);
               printf("Current temperature in the zone: ");
               printf("%f", temp);
        }

        fclose(t_ptr);
        printf("\n");
        free(buffer);
        delay(100);// 100 ms seconds delay
       }
    }
    free(characteristics);

disconnect_exit:
    gattlib_disconnect(gatt_connection);

connection_exit:
    printf("------------DONE %s ---------------\n", addr);
    pthread_mutex_unlock(&g_mutex);
    return NULL;
}

 void ble_discovered_device(void *adapter, const char* addr,
const char* name, void *user_data) {
    struct connection_t *connection;
    int ret;

    if (name) {
      printf("Discovered %s - '%s'\n", addr, name);
    } else {
      printf("Discovered %s\n", addr);
    }
     // finding the right BLE module using its device address
    if(strcmp(addr, "C2:87:6C:41:3C:E1") == 0)
    {
      connection = malloc(sizeof(struct connection_t));
      if (connection == NULL) {
           fprintf(stderr, "Fail to allocate connection.\n");
           return;
      }
      connection->addr = strdup(addr);
      ret = pthread_create(&connection->thread, NULL,
ble_connect_device, connection);
      if (ret != 0) {
           fprintf(stderr, "Fail to create BLE connection
thread.\n");
           free(connection);
```

```
                return;
            }
            LIST_INSERT_HEAD(&g_ble_connections, connection,
entries);
        }
}

int main(int argc, const char *argv[]) {
    remove("temperature.txt");
    const char* adapter_name;
    void* adapter;
    int ret;

    if (argc == 1) {
        adapter_name = NULL;
    } else if (argc == 2) {
        adapter_name = argv[1];
    } else {
        fprintf(stderr, "%s [<bluetooth-adapter>]\n", argv[0]);
        return 1;
    }
    LIST_INIT(&g_ble_connections);

    ret = gattlib_adapter_open(adapter_name, &adapter);
    if (ret) {
        fprintf(stderr, "ERROR: Failed to open adapter.\n");
        return 1;
    }

    pthread_mutex_lock(&g_mutex);
    ret = gattlib_adapter_scan_enable(adapter,
ble_discovered_device, BLE_SCAN_TIMEOUT, NULL /* user_data */);
    if (ret) {
        fprintf(stderr, "ERROR: Failed to scan.\n");
        goto EXIT;
    }

    gattlib_adapter_scan_disable(adapter);

    puts("Scan completed");
    pthread_mutex_unlock(&g_mutex);

    // Wait for the thread to complete
    while (g_ble_connections.lh_first != NULL) {
      struct connection_t* connection =
g_ble_connections.lh_first;
      pthread_join(connection->thread, NULL);
      LIST_REMOVE(g_ble_connections.lh_first, entries);
      free(connection->addr);
      free(connection);
    }

EXIT:
    gattlib_adapter_close(adapter);
    return ret;
      }
```

CMakeLists.txt : CMake file to build ble_scan.c

```
      #
#  GattLib - GATT Library
#
#  Copyright (C) 2016-2017  Olivier Martin
<olivier@labapart.org>
#
#
#  This program is free software; you can redistribute it
and/or modify
#  it under the terms of the GNU General Public License as
published by
#  the Free Software Foundation; either version 2 of the
License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be
useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty
```

```
of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public
License
#  along with this program; if not, write to the Free Software
#  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
02110-1301  USA
#

cmake_minimum_required(VERSION 2.6)

find_package(PkgConfig REQUIRED)

pkg_search_module(GATTLIB REQUIRED gattlib)
pkg_search_module(PCRE REQUIRED libpcre)

set(ble_scan_SRCS ble_scan.c)

add_executable(ble_scan ${ble_scan_SRCS})
target_link_libraries(ble_scan ${GATTLIB_LIBRARIES}
${GATTLIB_LDFLAGS} ${PCRE_LIBRARIES} pthread)
```

ble_write.c : Code to transmit a character to Teensy from Pi through BLE (used to turn off lighting in the zone)

```
     #include <pthread.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/queue.h>
#include <time.h>
#include <assert.h>

#include "gattlib.h"

#define BLE_SCAN_TIMEOUT   4
```

```
typedef void (*ble_discovered_device_t)(const char* addr, const
char* name);

// We use a mutex to make the BLE connections synchronous
static pthread_mutex_t g_mutex = PTHREAD_MUTEX_INITIALIZER;

LIST_HEAD(listhead, connection_t) g_ble_connections;
struct connection_t {
    pthread_t thread;
    char* addr;
    LIST_ENTRY(connection_t) entries;
};


void delay(int number_of_seconds)
{
     // Converting time into milli_seconds
     int milli_seconds = 1000 * number_of_seconds;

     // Storing start time
     clock_t start_time = clock();

     // looping till required time is not achieved
     while (clock() < start_time + milli_seconds);
}

 void *ble_connect_device(void *arg) {
    struct connection_t *connection = arg;
    char* addr = connection->addr;
    gatt_connection_t* gatt_connection;
    gattlib_primary_service_t* services;
    gattlib_characteristic_t* characteristics;
    int services_count, characteristics_count;
    char uuid_str[MAX_LEN_UUID_STR + 1];
    int ret, i;
    size_t len;

    pthread_mutex_lock(&g_mutex);

    printf("------------START %s ---------------\n", addr);
    gatt_connection = gattlib_connect(NULL, addr,
GATTLIB_CONNECTION_OPTIONS_LEGACY_DEFAULT);
    if (gatt_connection == NULL) {
      fprintf(stderr, "Failed to connect to the bluetooth
device.\n");
      goto connection_exit;
    } else {
      puts("Succeeded to connect to the bluetooth device.");
    }

    ret = gattlib_discover_char(gatt_connection,
&characteristics, &characteristics_count);
    if (ret != 0) {
      fprintf(stderr, "Fail to discover characteristics.\n");
      goto disconnect_exit;
    }
    for (i = 0; i < characteristics_count; i++) {
      gattlib_uuid_to_string(&characteristics[i].uuid,
uuid_str, sizeof(uuid_str));

      if(strcmp(uuid_str,
"6e400002-b5a3-f393-e0a9-e50e24dcca9e") == 0)
      {
            printf("characteristic[%d] properties:%02x
value_handle:%04x uuid:%s\n", i, characteristics[i].properties,
characteristics[i].value_handle,
                 uuid_str);
           char value_data = 'a';
            ret = gattlib_write_char_by_uuid(gatt_connection,
&characteristics[i].uuid, &value_data, sizeof(value_data));
            if (ret != GATTLIB_SUCCESS) {

                 if (ret == GATTLIB_NOT_FOUND) {
                      fprintf(stderr, "Could not find GATT
Characteristic with UUID %s. "
                           "You might call the program with
'--gatt-discovery'.\n", uuid_str);
                 } else {
                      fprintf(stderr, "Error while writing GATT
Characteristic with UUID %s (ret:%d)\n",
                           uuid_str, ret);
                 }
            goto disconnect_exit;
            }
      }
    }
    free(characteristics);

disconnect_exit:
    gattlib_disconnect(gatt_connection);

connection_exit:
    printf("------------DONE %s ---------------\n", addr);
    pthread_mutex_unlock(&g_mutex);
    return NULL;
}

 void ble_discovered_device(void *adapter, const char* addr,
const char* name, void *user_data) {
    struct connection_t *connection;
    int ret;

    if (name) {
      printf("Discovered %s - '%s'\n", addr, name);
    } else {
      printf("Discovered %s\n", addr);
    }
    if(strcmp(addr, "E7:BC:8B:41:05:F3") == 0)
    {
      connection = malloc(sizeof(struct connection_t));
      if (connection == NULL) {
            fprintf(stderr, "Fail to allocate connection.\n");
            return;
      }
      connection->addr = strdup(addr);

      ret = pthread_create(&connection->thread, NULL,
ble_connect_device, connection);
      if (ret != 0) {
            fprintf(stderr, "Fail to create BLE connection
thread.\n");
            free(connection);
            return;
        }
        LIST_INSERT_HEAD(&g_ble_connections, connection,
entries);
    }
}

int main(int argc, const char *argv[]) {
    const char* adapter_name;
    void* adapter;
    int ret;

    if (argc == 1) {
        adapter_name = NULL;
    } else if (argc == 2) {
        adapter_name = argv[1];
    } else {
        fprintf(stderr, "%s [<bluetooth-adapter>]\n", argv[0]);
        return 1;
    }

    LIST_INIT(&g_ble_connections);

    ret = gattlib_adapter_open(adapter_name, &adapter);
    if (ret) {
        fprintf(stderr, "ERROR: Failed to open adapter.\n");
        return 1;
    }

    pthread_mutex_lock(&g_mutex);
    ret = gattlib_adapter_scan_enable(adapter,
ble_discovered_device, BLE_SCAN_TIMEOUT, NULL /* user_data */);
    if (ret) {
        fprintf(stderr, "ERROR: Failed to scan.\n");
        goto EXIT;
    }
    gattlib_adapter_scan_disable(adapter);

    puts("Scan completed");
    pthread_mutex_unlock(&g_mutex);

    // Wait for the thread to complete
    while (g_ble_connections.lh_first != NULL) {
      struct connection_t* connection =
g_ble_connections.lh_first;
      pthread_join(connection->thread, NULL);
      LIST_REMOVE(g_ble_connections.lh_first, entries);
      free(connection->addr);
      free(connection);
    }

EXIT:
    gattlib_adapter_close(adapter);
    return ret;
}
```

CMakeLists.txt : CMake file to build ble_write.c

```
      #
#   GattLib - GATT Library
#
#   Copyright (C) 2016-2017  Olivier Martin
<olivier@labapart.org>
#
#
#   This program is free software; you can redistribute it
and/or modify
#   it under the terms of the GNU General Public License as
published by
#   the Free Software Foundation; either version 2 of the
License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be
useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty
of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public
License
#  along with this program; if not, write to the Free Software
#  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
02110-1301  USA
#

cmake_minimum_required(VERSION 2.6)

find_package(PkgConfig REQUIRED)

pkg_search_module(GATTLIB REQUIRED gattlib)
pkg_search_module(PCRE REQUIRED libpcre)

set(ble_write_SRCS ble_write.c)

add_executable(ble_write ble_write.c)
target_link_libraries(ble_write ${GATTLIB_LIBRARIES}
${GATTLIB_LDFLAGS} ${PCRE_LIBRARIES} pthread)
```

checking_N_and_tracking_time.cpp : Code to track time and call the program ble_write.c (code to send signal to Teensy when people count is zero)

```
     #include <iostream>
#include <fstream>
#include <chrono>
#include <stdio.h>
#include <unistd.h>
//using namespace cv;
using namespace std;

// global variables
     unsigned int second = 1000000; // microseconds to second
```

```
      int hours, minutes, seconds;
      bool timer_on = false;
      double total_time_in_s = 0;
      float Navg = 0;
      std::chrono::steady_clock::time_point Tstart;
      std::chrono::steady_clock::time_point Tend;

      // Function to convert seconds to Hours, minutes and
seconds
void convertSecToHours()
{
    minutes = seconds / 60;
    hours = (int) minutes / 60;
    minutes = int(minutes%60);
    seconds = int(seconds%60);
      }

int main(int argc, char **argv) {
            // files to read data
       string filename = "N_1value.txt";
       ifstream fin;
        int N;
        int i;
       if ( remove("T_hours.txt") !=0 )
            perror("Error deleting T_hours.txt!!");
       else
            puts("T_hours.txt successfully deleted");


       for(i=0; i<55; ++i)
        {

          fin.open(filename);
          if(fin.is_open()){
               fin >> N;
              fin.close();
              std::cout << endl << "Current No. of people in the
zone : " << N << std::endl;

                  if (N > 0)
```

```
                {
                        if( !timer_on)
                        {
                            // start timer
                            Tstart = 
std::chrono::steady_clock::now();
                            timer_on = true;
                            cout << "\n...........Turning
on Timer!.........";
                        }
                }
                else if (N == 0)
                {
                        if(timer_on)
                        {
                            cout << "\n......Turning off
Timer!....";
                            Tend = 
std::chrono::steady_clock::now();
                            std::chrono::duration<double>
elapsed_seconds = Tend-Tstart;

                            std::cout << "elapsed time: "
<< elapsed_seconds.count();
                            seconds += 
elapsed_seconds.count();
                             convertSecToHours();

                            ofstream T_file;
                             T_file.open("T_hours.txt",
std::ios_base::app);
                            T_file << hours <<":" <<
minutes << ":" << seconds << endl;
                            T_file.close();
                            timer_on = false;
                            system("./ble_write");
                        }
                 }
             }
        else
```

```
            {
                 cout << "\nError: file could not be opened!!";
                 exit(1);
            }

          usleep(0.5*second);
         }

        return 0;
          }
```

put_object.cpp : Code to add an object (file) to Amazon S3 bucket

```
           // Copyright Amazon.com, Inc. or its affiliates. All
      Rights Reserved.
      // SPDX - License - Identifier: Apache - 2.0

      //snippet-start:[s3.cpp.put_object.inc]
      #include <iostream>
      #include <fstream>
      #include <sys/stat.h>
      #include <aws/core/Aws.h>
      #include <aws/s3/S3Client.h>
      #include <aws/s3/model/PutObjectRequest.h>
      #include <awsdoc/s3/s3_examples.h>
      //snippet-end:[s3.cpp.put_object.inc]

      /*
      ///////////////////////////////////////////////////////////////
      /////////////
       * Purpose: Adds an object to an Amazon S3 bucket.
       *
       * Prerequisites: An Amazon S3 bucket and the object to be
      added.
       *
       * Inputs:
       * - bucketName: The name of the bucket.
       * - objectName: The name of the object.
       * - region: The AWS Region for the bucket.
```

```
 *
 * Outputs: true if the object was added to the bucket;
otherwise, false.
 *
///////////////////////////////////////////////////////////////
////////// */

// snippet-start:[s3.cpp.put_object.code]
bool AwsDoc::S3::PutObject(const Aws::String& bucketName,
      const Aws::String& objectName,
      const Aws::String& region)
{
      // Verify that the file exists.
      struct stat buffer;

      if (stat(objectName.c_str(), &buffer) == -1)
      {
      std::cout << "\nError: PutObject: File '" <<
            objectName << "' does not exist." << std::endl;

      return false;
      }

      Aws::Client::ClientConfiguration config;

      if (!region.empty())
      {
      config.region = region;
      }

      Aws::S3::S3Client s3_client(config);

      Aws::S3::Model::PutObjectRequest request;
      request.SetBucket(bucketName);
      request.SetKey(objectName);

      std::shared_ptr<Aws::IOStream> input_data =
      Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
            objectName.c_str(),
            std::ios_base::in | std::ios_base::binary);
```

```
        request.SetBody(input_data);

        Aws::S3::Model::PutObjectOutcome outcome =
        s3_client.PutObject(request);

        if (outcome.IsSuccess()) {

        std::cout << "\nAdded object '" << objectName << "' to
bucket '"
                << bucketName << "'.";
        return true;
        }
        else
        {
        std::cout << "\nError: PutObject: " <<
                outcome.GetError().GetMessage() << std::endl;

        return false;
        }
}

int main(int argc, char* argv[])
{
        Aws::SDKOptions options;
        Aws::InitAPI(options);
        {
        const Aws::String bucket_name = "temptesting01";
        const Aws::String object_name = argv[1];
        const Aws::String region = "us-east-2";

        if (!AwsDoc::S3::PutObject(bucket_name, object_name,
region)) {

                return 1;
        }
        }
        Aws::ShutdownAPI(options);

        return 0;
```

```
    }
    // snippet-end:[s3.cpp.put_object.code]
```

CMakeLists.txt : To build put_object.cpp

```
        # Copyright Amazon.com, Inc. or its affiliates. All Rights
Reserved.
# SPDX-License-Identifier: Apache-2.0

# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.8)

# Set this project's name.
project("s3-examples")

# Set the C++ standard to use to build this target.
set(CMAKE_CXX_STANDARD 11)

# Enable CTest for testing these code examples.
include(CTest)

# Build shared libraries by default.
if(NOT BUILD_SHARED_LIBS)
        set(BUILD_SHARED_LIBS ON)
endif()

# Find the AWS SDK for C++ package.
#find_package(AWSSDK REQUIRED COMPONENTS s3 sts)
find_package(AWSSDK REQUIRED COMPONENTS s3)


# If the compiler is some version of Microsoft Visual C++, or
another compiler simulating C++,
# and building as shared libraries, then dynamically link to
those shared libraries.
if(MSVC AND BUILD_SHARED_LIBS)
        add_definitions(-DUSE_IMPORT_EXPORT)
        # Copy relevant AWS SDK for C++ libraries into the current
binary directory for running and debugging.
        list(APPEND SERVICE_LIST s3 sts)
```

```
      AWSSDK_CPY_DYN_LIBS(SERVICE_LIST ""
${CMAKE_CURRENT_BINARY_DIR})
endif()

# Add the code example-specific header files.
file(GLOB AWSDOC_S3_HEADERS
      "include/awsdoc/s3/*.h"
)

# Add the code example-specific source files.
file(GLOB AWSDOC_S3_SOURCE
      "*.cpp"
)

# Check whether the target system is Windows, including Win64.
if(WIN32)
      # Check whether the compiler is some version of Microsoft
Visual C++, or another compiler simulating C++.
      if(MSVC)
      source_group("Header Files\\awsdoc\\s3" FILES
${AWSDOC_S3_HEADERS})
      source_group("Source Files" FILES ${AWSDOC_S3_SOURCE})
      endif(MSVC)
endif()

foreach(file ${AWSDOC_S3_SOURCE})
      get_filename_component(EXAMPLE ${file} NAME_WE)

      # Build the code example executables.
      set(EXAMPLE_EXE run_${EXAMPLE})

      add_executable(${EXAMPLE_EXE} ${AWSDOC_S3_HEADERS}
${file})

      if(MSVC AND BUILD_SHARED_LIBS)
      target_compile_definitions(${EXAMPLE_EXE} PUBLIC
"USE_IMPORT_EXPORT")
      target_compile_definitions(${EXAMPLE_EXE} PRIVATE
"AWSDOC_S3_EXPORTS")
      endif()
      target_include_directories(${EXAMPLE_EXE} PUBLIC
      $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
      $<INSTALL_INTERFACE:include>)
      target_link_libraries(${EXAMPLE_EXE}
${AWSSDK_LINK_LIBRARIES}
      ${AWSSDK_PLATFORM_DEPS})

      if(BUILD_TESTING)
      # Enable testing for this directory and below.
      enable_testing()

      # Build the code example libraries.
      set(EXAMPLE_LIB ${EXAMPLE})

      add_library(${EXAMPLE_LIB} ${AWSDOC_S3_HEADERS} ${file} )

      if(MSVC AND BUILD_SHARED_LIBS)
            target_compile_definitions(${EXAMPLE_LIB} PUBLIC
"USE_IMPORT_EXPORT")
            target_compile_definitions(${EXAMPLE_LIB} PRIVATE
"AWSDOC_S3_EXPORTS")
      endif()

      target_include_directories(${EXAMPLE_LIB} PUBLIC

$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
            $<INSTALL_INTERFACE:include>)
      target_link_libraries(${EXAMPLE_LIB}
${AWSSDK_LINK_LIBRARIES}
            ${AWSSDK_PLATFORM_DEPS})

      # Build the code example unit tests.
      set(EXAMPLE_TEST test_${EXAMPLE})
      set(EXAMPLE_TEST_FILE
${CMAKE_CURRENT_SOURCE_DIR}/tests/test_${EXAMPLE}.cpp)

      if(EXISTS ${EXAMPLE_TEST_FILE})
            add_executable(${EXAMPLE_TEST} ${EXAMPLE_TEST_FILE}
)
```

```
            target_include_directories(${EXAMPLE_TEST} PUBLIC

$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
                $<INSTALL_INTERFACE:include>)
            target_link_libraries(${EXAMPLE_TEST} ${EXAMPLE_LIB}
)
            add_test(${EXAMPLE_TEST} ${EXAMPLE_TEST})
      endif()

      endif()
endforeach()
```

counter.py : Code based on OpenCV algorithm to detect persons entering and exiting zone and to count them

```
      import numpy as np
import time
import imutils
import cv2
import os

avg = None

      # put 0 instead of the video file name inside the
parenthesis for live camera capturing
video = cv2.VideoCapture("vid1_cropped.mp4")
xvalues = list()
motion = list()
count1 = 0
count2 = 0
N = 0
if os.path.exists("N_people.txt"):
    os.remove("N_people.txt") # delete any existing file
if os.path.exists("N_1value.txt"):
    os.remove("N_1value.txt") # delete any existing file

def find_majority(k):
      myMap = {}
```

```
        maximum = ( '', 0 ) # (occurring element, occurrences)
        for n in k:
        if n in myMap: myMap[n] += 1
        else: myMap[n] = 1

        # Keep track of maximum on the go
        if myMap[n] > maximum[1]: maximum = (n,myMap[n])

        return maximum

while 1:
        ret, frame = video.read()
        flag = True
        text=""
        if ret == True:
         frame = imutils.resize(frame, width=500)
        else:
         break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (21, 21), 0)

        if avg is None:
        print "[INFO] starting background model..."
        avg = gray.copy().astype("float")
        continue

        cv2.accumulateWeighted(gray, avg, 0.5)
        frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(avg))
        thresh = cv2.threshold(frameDelta, 5, 255,
cv2.THRESH_BINARY)[1]
        thresh = cv2.dilate(thresh, None, iterations=2)
   # (_, cnts, _) = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        (cnts, _) = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        for c in cnts:
        if cv2.contourArea(c) < 5000:
                continue
        (x, y, w, h) = cv2.boundingRect(c)
```

```
           xvalues.append(x)
          cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255),
    2)
          flag = False

           no_x = len(xvalues)

           if (no_x > 2):
           difference = xvalues[no_x - 1] - xvalues[no_x - 2]
           if(difference > 0):
                motion.append(1)
           else:
                motion.append(0)

           if flag is True:
           if (no_x > 5):
                val, times = find_majority(motion)
                if val == 1 and times >= 15:
                     count1 += 1
                else:
                     count2 += 1

           xvalues = list()
           motion = list()

           cv2.line(frame, (260, 0), (260,480), (0,255,0), 2)
           cv2.line(frame, (420, 0), (420,480), (0,255,0), 2)
           N = count2 - count1

              # write to files
           f = open("N_people.txt", "a")
           f1 = open("N_1value.txt", "w")
           f.write("{}\n".format(N))
           f1.write("{}\n".format(N))
           f.close()
           f1.close()
           cv2.putText(frame, "Out: {}".format(count1), (10, 20),
    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
           cv2.putText(frame, "In: {}".format(count2), (10, 40),
    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

```
        #f.write("In: {}\n".format(count1))
        #f.write("Out: {}\n".format(count2))
        #f.write("N: {}\n\n".format(N))
        cv2.imshow("Frame",frame)
        cv2.imshow("Gray",gray)
        cv2.imshow("FrameDelta",frameDelta)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
        break


video.release()
cv2.destroyAllWindows()
```

test1.sh : Bash script used to integrate all the program codes

```
        #!/bin/bash

        # compiling cpp file
g++ -o track.o checking_N_and_tracking_time.cpp

        # reading in temperature data
./ble_scan

        # running people counter
python counter.py &

        # doing time tracking and turning off of lighting based on
people count
./track.o &

wait # wait for all the processes to get complete

        # adding all objects to s3 bucket
./run_put_object "temperature.txt"
./run_put_object "T_hours.txt"
./run_put_object "N_people.txt"
printf "\nTime : "
```

```
date +"%H:%M:%S"

     # exporting all files to appropriate location on Github
page
cp "temperature.txt"
~/Smart-Sensing-network-through-IoT-for-IRS/Vaibhav/build-QtTry
2-Desktop-Debug/tempdata.txt
cp "T_hours.txt"
~/Smart-Sensing-network-through-IoT-for-IRS/Vaibhav/build-QtTry
2-Desktop-Debug/hours.txt
cp "N_people.txt"
~/Smart-Sensing-network-through-IoT-for-IRS/Vaibhav/build-QtTry
2-Desktop-Debug/personsdata.txt

cd ~/Smart-Sensing-network-through-IoT-for-IRS
git pull
git add .
git commit -m "...........Adding all data files to
git............."
git push

exit 0
```

dialog.cpp : Qt code to create GUI display from the data files

```
     #include "dialog.h"
#include "ui_dialog.h"
#include "fstream"
#include "iostream"
#include "string"
#include "sstream"
//using namespace std;

Dialog::Dialog(QWidget *parent) : QDialog(parent), ui(new
Ui::Dialog)
{
   ui->setupUi(this);
   ui-> lcdNumberTemp ->display("---------");
```

```
std::ifstream temp,person,hours;
temp.open("tempdata.txt");
person.open("personsdata.txt");
hours.open("hours.txt");

if(temp.fail())
{
    std::cout << "Error: File failed to open"<<std::endl;
}
else
{
    std::string text;
    while(getline (temp, text))
    {
        QString qstr = QString::fromStdString(text);
        Dialog::updateTemperature(qstr);
        text.clear();
    }
}
if(person.fail())
{
    std::cout << "Error: File failed to open"<<std::endl;
}
else
{
    std::string text1;
    while(getline (person, text1))
    {
        QString qstr1 = QString::fromStdString(text1);
        Dialog::updatePersons(qstr1);
    }
}

if(hours.fail())
{
    std::cout << "Error: File failed to open"<<std::endl;
}
else
{
        std::string text2;
        while(getline (hours, text2))
        {
            QString qstr2 = QString::fromStdString(text2);
            Dialog::updateHours(qstr2);
        }
    }

    temp.close();
    person.close();
    hours.close();

}

Dialog::~Dialog()
{
    delete ui;
}

void Dialog::updateTemperature(QString temp_sensor)
{
    ui->lcdNumberTemp->display(temp_sensor);
}

void Dialog::updatePersons(QString person_sensor)
{
    ui->lcdNumberPersons->display(person_sensor);
}

void Dialog::updateHours(QString time_hours)
{
    ui->lcdNumberHour->display(time_hours);
}
```

dialog.h : Header file for dialog.cpp

```
      #ifndef DIALOG_H
#define DIALOG_H
```

```
#include <QDialog>

QT_BEGIN_NAMESPACE
       namespace Ui { class Dialog; }
QT_END_NAMESPACE

       class Dialog : public QDialog
       {
               Q_OBJECT

               public:
               Dialog(QWidget *parent = nullptr);
               ~Dialog();

               private slots:
               void updateTemperature(QString);
               void updatePersons(QString);
               void updateHours(QString);

               private:
               Ui::Dialog *ui;
       };
#endif // DIALOG_H
```

main.cpp : Main code for Qt display

```
     #include "dialog.h"
#include <QApplication>

int main(int argc, char *argv[])
{
     QApplication a(argc, argv);
     Dialog w;
     w.setFixedSize(443,322);
     w.setWindowTitle("Capstone Display");
     w.show();
     return a.exec();
}
```

LINKS

Github link:

https://github.com/Capstone-redefined/Smart-Sensing-network-through-IoT-for-IRS

All project codes can be found at :
https://github.com/Capstone-redefined/Smart-Sensing-network-through-IoT-for-IRS/tree/main/Archana/Arduino
https://github.com/Capstone-redefined/Smart-Sensing-network-through-IoT-for-IRS/tree/main/Archana/Capstone
https://github.com/Capstone-redefined/Smart-Sensing-network-through-IoT-for-IRS/tree/main/Vaibhav/QtTry2

Youtube video links:

https://www.youtube.com/watch?v=X3Mn9leFYzM&t=10s
https://www.youtube.com/watch?v=l3yf8G_R