Helpful Hallway

Lab Embedded systems

Tuur Baele
Pieter De Block
Rijad Šehović
Thomas Wauters

May 6, 2020

KU LEUVEN

Contents

1 Introduction

Because the hallways of the campus can be crowded, a system is designed that can monitor a hallway and report the results. These results can be used to notify people in the campus to avoid certain crowded hallways. A more precise description of requirements of the system is given in the system requirements. After the requirements all the parts of the design are described.

All code used in this project is available at
`https://github.com/thomaswauters1/Helpful-Hallway-Public`

2 System requirements

2.1 Functional Technical

- Monitoring presence of people (Are they blocking the hallway?)
- The intensity of the crowd (How hard is it to pass through this hallway?)
- Is it too noisy in the hallway? (useful in exam situations)
- Map (navigation)
- Where are the people standing?
- Deep Sleep in the weekends / night (20:00-6:00)
- If a group of 5 or less humans is detected then every 30 seconds there is an upload.
- If a larger group than 5 people is detected then every 10 seconds an upload.

2.2 Non-Functional Technical

- Max Size: 20cm x 10cm x 10cm
- A good fixation of the system, so that it cannot fall on the head of the pedestrians
- Weight: max 2kg
- Temperature atmosphere: between $-10\,^{\circ}\mathrm{C}$ and $50\,^{\circ}\mathrm{C}$
- Active status led
- Battery life of 3 months
- Costs: 50euro

2.3 Non-Technical

- Not too prominent enclosure
- Making sure Odisee doesn't interrupt/complain

3 System overview

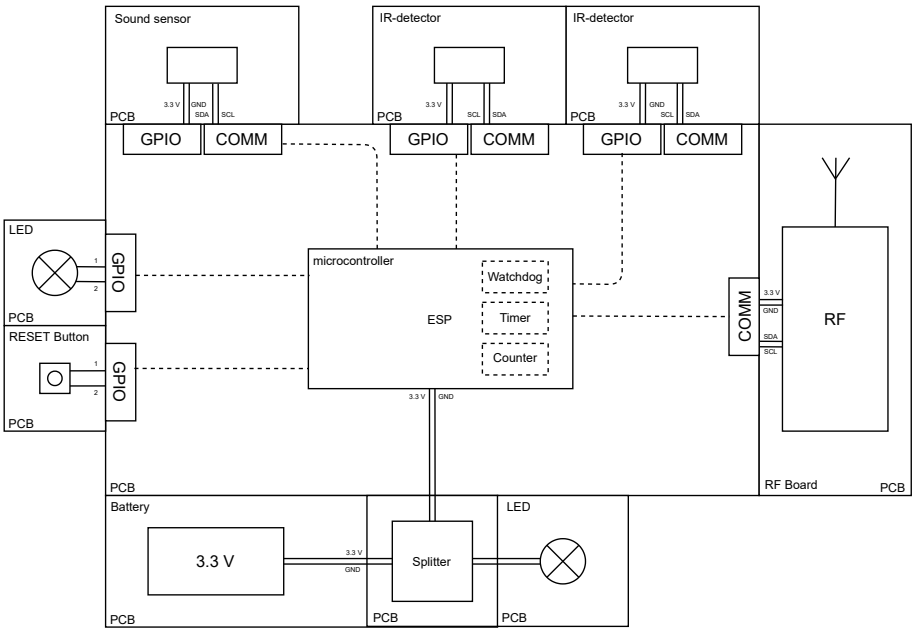

Figure 1: System overview/architecture

4 Hardware

4.1 PCB

An important part of the project is the development of the hardware. We chose to make a main PCB (Printed Circuit Board) and two separate small PCB on which the IR cameras/sensors will be placed. This allows a better positioning of the cameras depending on the hallway. On fig. 2 you can see the top of the main PCB where all components are placed and on fig. 3 you can see the PCB for the cameras.

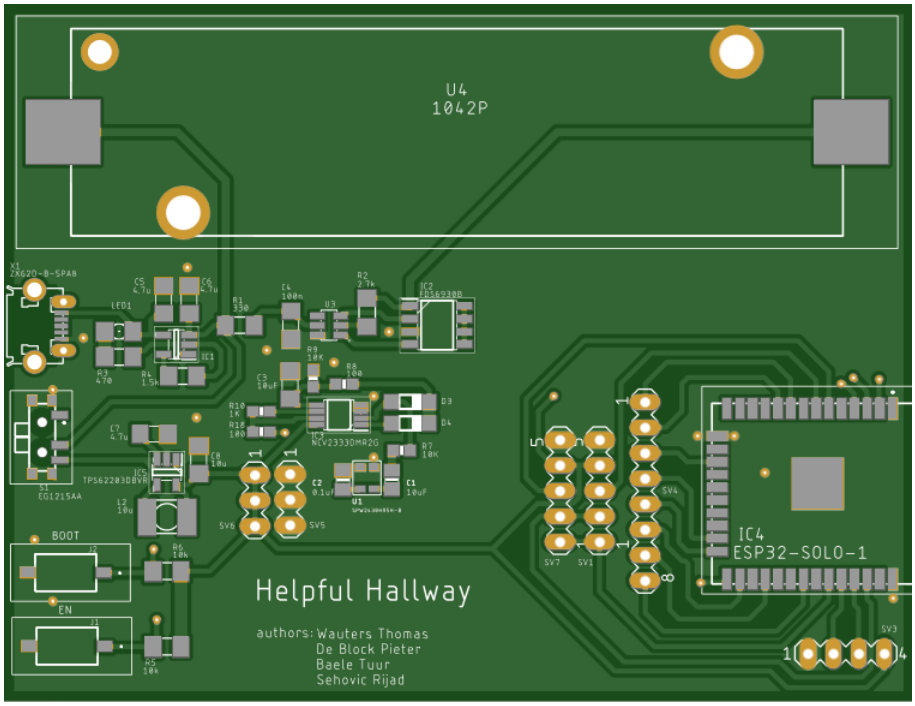

Figure 2: Top view of main PCB

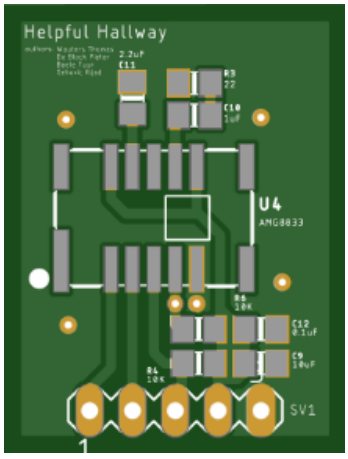

Figure 3: Top view of camera PCB

4.2 Micro-controller

For the micro-controller we have chosen for the ESP32-SOLO-1 from Espressif which is shown on fig. 4. This model has a single core and 5 power modes which allows us to decrease the power usage of the device a lot. The device is based on Espressifs Xtensa architecture and is powerful enough for our application. We chose this processor specifically because of the integrated wifi capabilities. An important thing to consider concerning the wifi, was that the campus network is an enterprise network, which is not supported by a lot of wifi chips, but this SoC does.

Figure 4: ESP32 micro-controller

4.3 Battery

We want the device to be easy to install and when it is placed no one should have to look after it for a long time. This means that the battery has to be able to store al lot of energy in a relatively small space. It also has to have a low self-discharge, if this was not the case we would lose too much energy when the device isn't deployed. We also wanted to be able to charge the battery when they run out of power. The best choice for these conditions is a 18650 lithium ion cell. Another advantage of these cells is that because these are used for a lot of applications they are relatively cheap and widely available. A disadvantage of this battery is that when it's discharged too much, the battery is damaged. While cells that have built-in protections exist, we chose to implement this on the board for safety. For this we use a AP9101CK6 protection IC along with two mosfets for turning of the device off. The charging of the device is also implemented in the device. For this we have placed a micro-usb connector on the board to make this very easy for the user. We use a MCP73831T charge controller IC to make sure the battery is charged properly.

4.4 Power usage

The battery can store 3000mAh of energy. We measured that the micro-controller uses about 20 µA when it is in sleep mode. When the IR-sensor is in sleep it uses around 156 µA and this will be further explaned in chapter 8.3. When the device is measuring and sending data the system uses about 100 mA and this for about 2 seconds although this is based on datasheets and online information and needs to be tested further [1][2][3]. We suspect that the device will have to wake from its sleep 50 times a day on average. For the losses from the buck converter, self discharge of the battery and other losses from the hardware we assume an other 20 µA is used, although this should also be measured further.

Per measuring cycle around $100\,\mathrm{mA} \cdot 2\,\mathrm{s} = 0.056\,\mathrm{mA\,h}$.
The total battery life can be calculated as follows:

$$(20\,\mu\mathrm{A} + 20\,\mu\mathrm{A} + 2 \cdot 156\,\mu\mathrm{A})\,\mathrm{t_h} + 0.056\,\mathrm{mA\,h} \cdot \frac{50/\mathrm{day}}{24\mathrm{h}/\mathrm{day}} \cdot \mathrm{t_h} = 3000\,\mathrm{mA\,h} \tag{1}$$

$$\Rightarrow \mathrm{t_h} = 6401.13\,\mathrm{h} \approx 226\mathrm{days} \tag{2}$$

5 Wireless connectivity

Wi-fi is chosen for the wireless communication because the system will be installed in the hallways of campus where there is always a strong Wi-fi signal. The second decision was to use MQTT (Message Queuing Telemetry Transport), an so called IoT connectivity protocol because it is light, low power and has efficient distribution of information.

5.1 MQTT

Basic topographic of MQTT protocol is show on fig. 5.

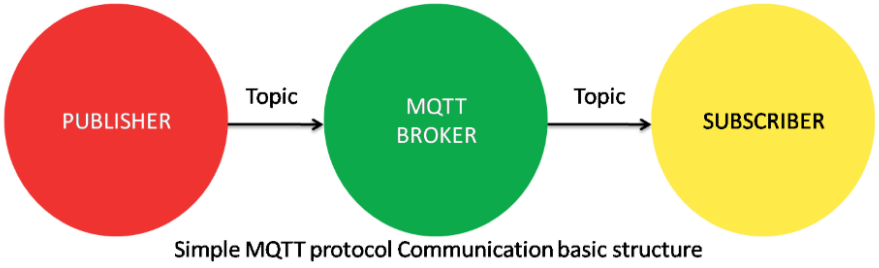

Figure 5: Basic topographic of MQTT protocol

5.2 Code

As can be seen on the diagram, the is kept extremely simple and clean because the purpose is to publish information as soon as possible and go back to sleep. This is important because the WiFi connection is using the most power in whole system. For connecting purposes a method from predefined library named 'Wifi.h' is used. The method directly receives SSID, password, channel and BSSID of our access point to try to speed it up as much as possible to be faster back to sleeping mode.

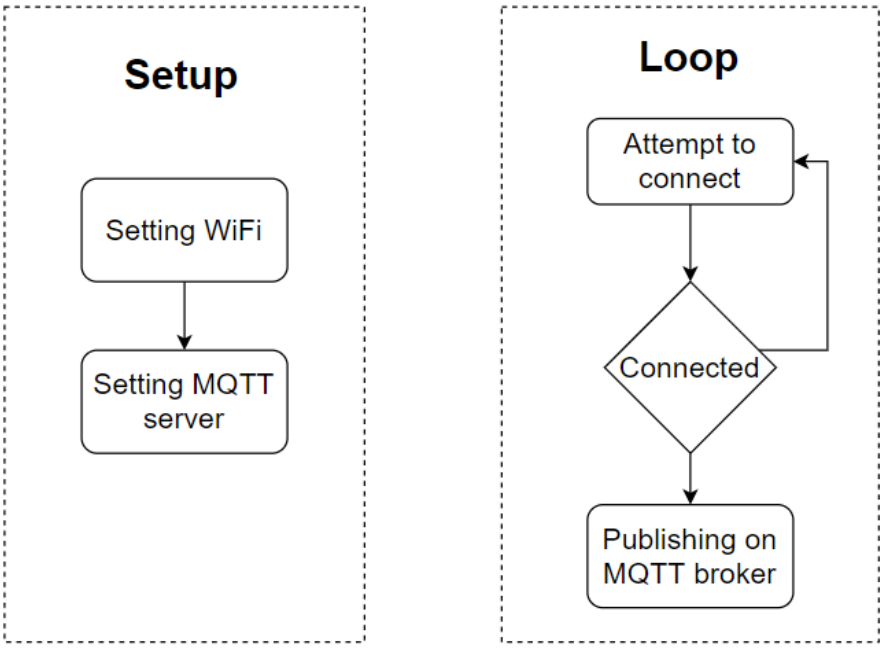

Figure 6: Code diagram

Hereby is whole code used for wireless connectivity purposes:

```
#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>
#define MQTTpubQos 0

const char* ssid = "xxxx";       //replace with current WiFi name (eduroam)
const char* password = "xxxx";    //current WiFi password

const char* mqtt_server = "helpfulhallway.duckdns.org";   //Raspberrys address
const char* mqtt_topic = "HelpfulHallway";

WiFiClient espClient;
PubSubClient client(espClient);
const uint8_t* bssid = (const uint8_t*) "8c:3b:ad:27:75:fb";  //change to
    eduroam info 66:d1:54:54:48:d3
int chann = 1;                                                 //channel 11

void setup() {
  Serial.begin(115200);
  setCpuFrequencyMhz(80);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

```

```
  WiFi.begin(ssid, password, chann, bssid);
  while (WiFi.status() != WL_CONNECTED) {
    delay(5);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {

  while (!client.connected());
  client.loop();
  char testData[] = "";
  client.publish("test", testData);
}
```

6 Back end

To implement the back end part of our project, we chose an Raspberry Pi. On this server several things will be implemented. Because all nodes send their data as an mqtt client, we also need an mqtt broker. For this we use the open-source software Mosquitto. As a database, we have chosen for InfluxDB. This is a time series database, this stores the data as time value pairs, which is very convenient for our application. To get the data to the database we have written a Python program that receives the messages from the nodes converts it to a json format and sends it to the database. To visualize the data we use the open-source software Grafana. The nodes have calculated a value that corresponds with the traffic in a hallway. The software enables us to display this value as a color on the right locations on a floor-plan of the campus as you can see on picture 7 and is available on `helpfulhallway.duckdns.org:3000` .

Figure 7: Screenshot of the application

7 Audio sensor

A sensor is needed which can be used to frequently (less than every second) monitor the hallway to detect if there is people in the hallway. For this application a audio sensor is chosen for two main reasons. Sound is an trustworthy medium for detecting the presence of people in a hallway and it can be monitored with low power. The latter is and import issue in the device since it will be battery powered. In the following discussion it will become clear that most of the design decisions are made with low power usage in mind.

7.1 Technology

These are the import specifications:

- Omnidirectional
- Small($< 1cm^3$)
- Sensitive between $100Hz - 10kHz$

Since the device has to be as small as possible and low power, an MEMS technology sensor is suitable. The chosen sensor is called a Top Port SiSonic Microphone [4] displayed on fig. 8.

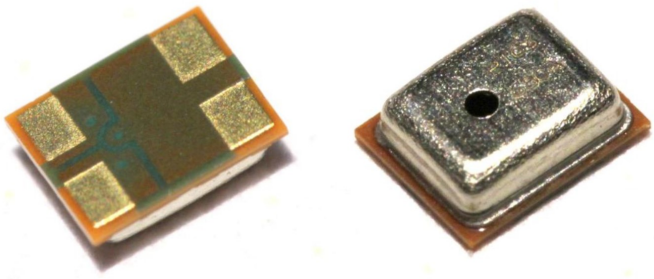

Figure 8: Top Port SiSonic Microphone Package [4]

To detect the presence of people in hallways the frequency range of interest is more narrow than the human hearing range of $20Hz - 20kHz$. Frequencies higher than $10kHz$ and below $100Hz$ are redundant. The range $100Hz - 10kHz$ is ideal for detecting voice activity which is roughly around $1kHz$ but also other sounds like footsteps. Figure 9 shows the flat response of the chosen sensor in the range of interest.

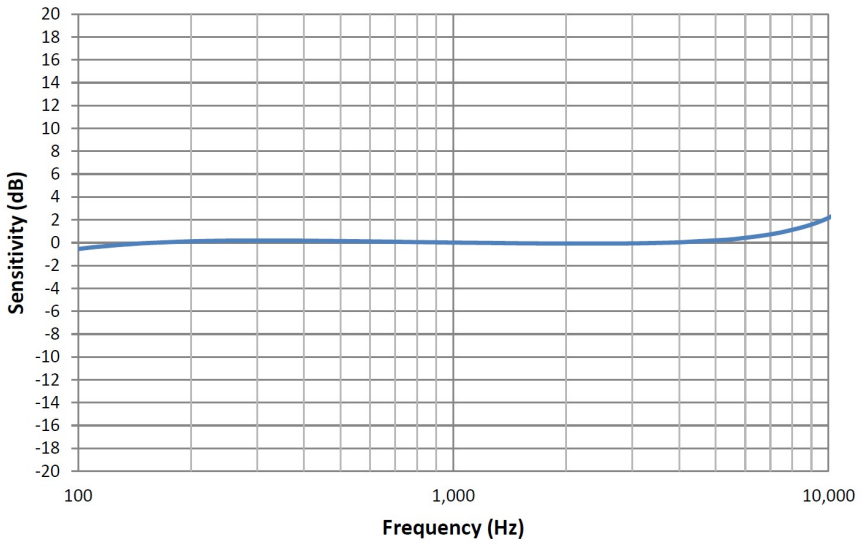

Figure 9: Microphone typical free field response normalized at 1kHz

The typical current of the sensor is $75-110\mu A@3.6V$. The signal of the mic also needs to be amplified and rectified so it can be processed by the ADC of the used micro-controller. In other words we are only interested in the positive envelope of the audio signal. For this purpose the circuit displayed on fig. 10 is used.

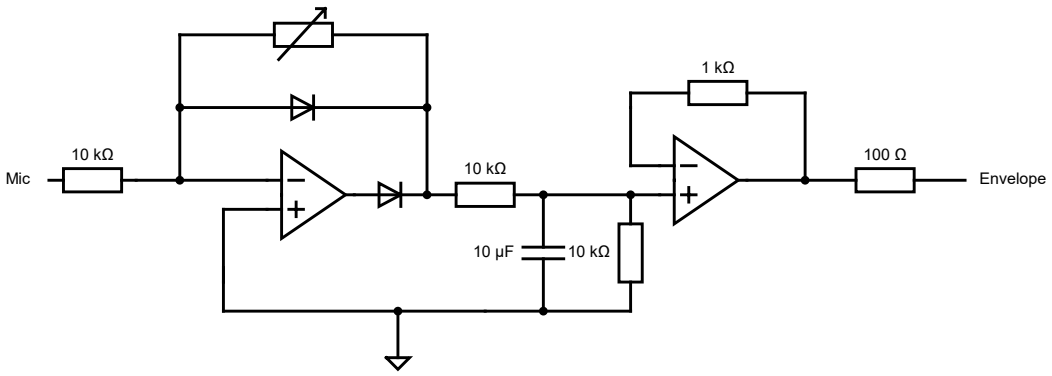

Figure 10: Microphone half-wave precision amplifier circuit [4]

To make this circuit low power, the most important part is the opamp [5]. The used opamp has low power consumption over a wide dynamic range. It has a low supply current of $17\mu A@3.3V$ The package used consists of two opamps which is ideal for realizing this circuit.

7.2 Monitoring System

For monitoring the audio the Ultra Low Power (ULP) co-processor of the ESP32 is used which is part of the RTC block as can be seen on fig. 11. Using the ULP co-processor the current can fluctuate between $10-100\mu$, depending on the routine of te co-processor. The routine of the co-

processor can be programmed using assembly code. The state where the co-processor is used is called Deep Sleep.

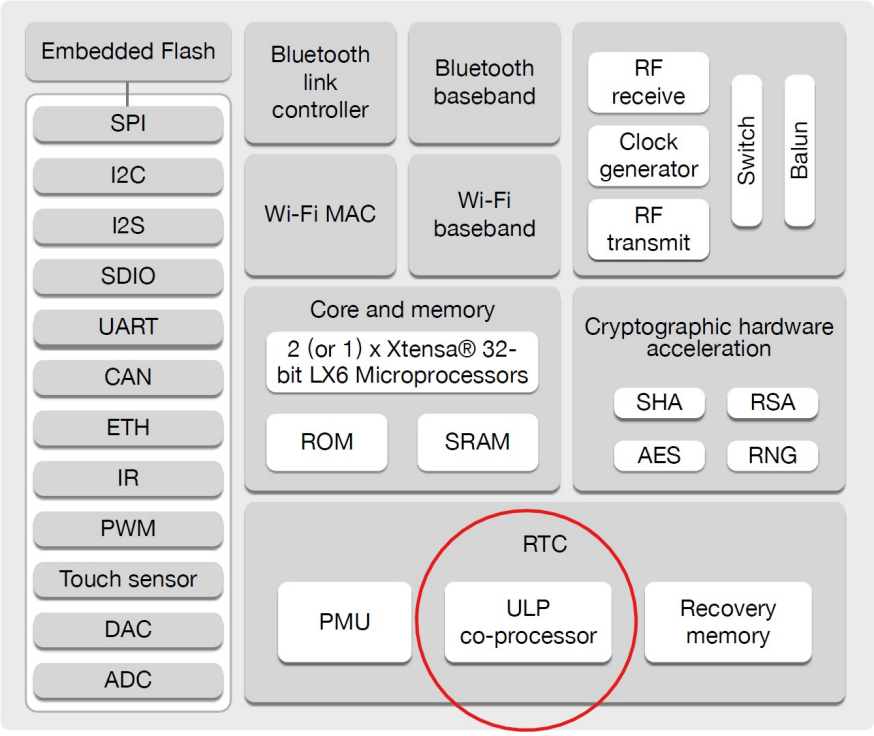

Figure 11: ESP32 functional block diagram [6]

The routine code in assembly is given on fig. 12. The threshold is an voltage value which is between 0 and 3.3V. The voltage is digitized using an 12-bit SAR ADC. The threshold which is suitable can be experimentally determined and set for each Hallway if necessary. The measuring period of 500ms also interchangeable.

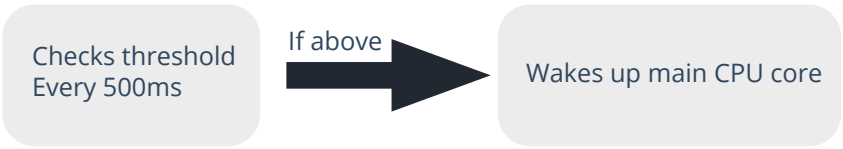

Figure 12: Co-processor routine

7.3 Monitoring test results

As an preliminary test of the routine, it was implemented on an WIPI 3.0 IoT development board [7]. On this board the power needed was around $20\mu A@3.3V$. It has to be noted that this is the power needed for only the co-processor. The audio sensor was not included in this measurement.

8 IR sensor

An IR sensor is used to detect how many people are present in a hallway. The IR sensor that is used is an Grid-EYE (AMG88) which is a high precision infrared array sensor based on advanced MEMS technology. It uses 64 (8x8) pixels to detect the temperature of its surroundings. The sensor makes it possible to detect people up to a maximum distance of 7 m. The figures in this section will mainly show one IR sensor because only one sensor was used during the programming and test phase. In order to be able to detect more of the hallway, multiple sensors can be chosen, for example two IR sensors displayed in the architecture of the design.
In this section the following topics will be discussed in detail:

- Reasons for using an IR sensor,
- The different packet options (for the wireless communication part), code and packet size,
- Power consumption,
- Future possibilities,
- Conclusion.

8.1 Why IR sensor

In this design, the IR sensor is used to detect people in the hallway. We use the IR sensor with the aim to know the intensity of the people, whether a hallway is blocked, or when there are a lot of people if there is still a way to pass through this group of people.

8.2 Packet

To send the wanted information via an wireless connection to the server we are using for the IR sensor certain data structure. This structure consists of two parts. The first part is the IR sensor address and the second a certain decision or the pixel values. This data structure is called a packet. In this project there are three options. Each option uses a different technique to give a representation of the amount of people that are present and/or to know if the hallway is blocked, partially blocked or free. Each option has his advantages and disadvantages and are explained below. fig. 13 shows the general structure of a packet.

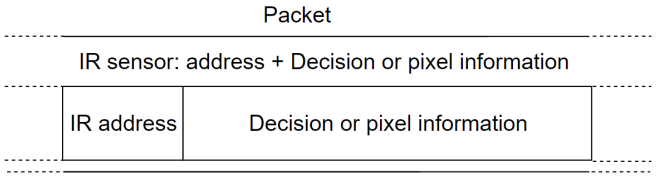

Figure 13: General structure of a packet

8.2.1 Option 1

Overview

With packet option 1 the packet is just going to contain all the pixel values of the IR sensor. When forming the packet we first add the IR sensor adres than all the pixel values. By sending all the pixel values to the server, all these values need to be processed by the server to make a decision. Figure 14 shows us an overview of the packet that will be send to the server when using option 1.

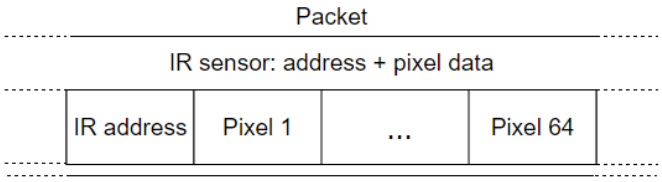

Figure 14: Overview of packet option 1

Code Here the code is displayed that is used for loading the packet with all the pixel values.

```
/**
 * Function: all pixel values
 * Load packet from [1-65] with all corresponding pixel values
 * Inputs: NONE
 * Outputs: NONE
 **/
void all_pixel_values(){
  // load the pixel array with all the measured pixels
  amg.readPixels(pixels);

  // load the packet array with all the pixel values from the pixel array
  for(int i=1; i<=AMG88xx_PIXEL_ARRAY_SIZE; i++){
    packet[i] = (uint16_t)pixels[i-1];      // uint16_t for the 8 byte low and
    8 byte high values per pixel
  }

  // Reset the Task Watchdog Timer (TWDT) to evade watchdog trigger
  esp_task_wdt_reset();
}
```

When running the needed code we get the following output:

```
================================
(Packet) Refresh IR pixel array:
done

(Packet) Show IR packet data:
Packet data: 69, 21, 21, 22, 23, 24, 25, 26, 26, 21,
22, 22, 23, 23, 25, 26, 27, 21, 22, 22,
21, 21, 24, 24, 25, 22, 22, 20, 20, 20,
20, 23, 26, 21, 22, 20, 19, 19, 19, 19,
23, 21, 21, 19, 20, 20, 20, 20, 21, 22,
20, 19, 19, 20, 20, 20, 20, 21, 19, 19,
20, 19, 20, 19, 20
Packet size: 130
done

================================
```

Figure 15: Output code packet option 1.

Packet size

Option 1 results in a large packet. The packet consists of the IR sensor address and all the pixel values. Since the packet consists of uint16_t values, the packet here will be 130 bytes in size.

Address = uint16_t = 2 byte
Pixel value = uint16_t = 2 byte
Amount of bytes = Address + (Pixel value . 64) = 2 byte + (2 byte . 64) = 130 bytes

8.2.2 Option 2

Overview

Packet option 2 uses levels that indicate the population present in the hallway. Five different levels are currently being used indicating a very low population to a very high population. The five levels that are now being used are very low, low, medium, high and very high population. This is optional and can be changed to the desired number of levels required for a given situation. When using this option, each time when packets are created an average value is calculated from all the pixel values. Next, the average value will be used to select a level. This can be easily achieved by assigning a certain value range to each level. When the mean value occurs in the value range of the level. That level will be selected. Each level has a corresponding number used in the packet. This number is a reference number that makes it possible to know the level of population when the packet is unpacked. Now the packet is formed by the IR sensor address and the reference number. Figure 16 gives an overview of the packet when choosing option 2.

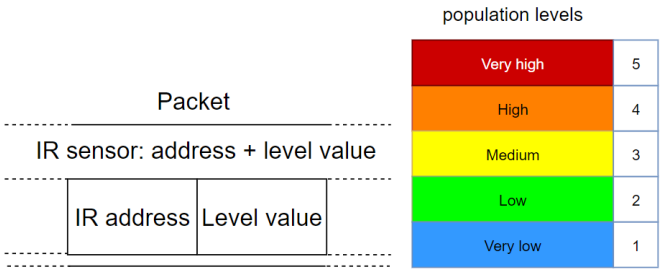

Figure 16: Overview of packet option 2 that is used for sending the wanted information from the IR sensor.

Code

Here you can see the code that is used to determine the average value and the reference number.

```
/**
 * Function: average pixels
 * Calculate the average value of all the pixels
 * Inputs: NONE
 * Outputs: NONE
 **/
void average_pixels(){
  // load the pixel array with all the measured pixels
  amg.readPixels(pixels);

  average = 0;
  // load the packet array with all the pixel values from the pixel array
  for(int i=1; i<=AMG88xx_PIXEL_ARRAY_SIZE; i++){
    average += (uint16_t)pixels[i-1];
  }

  // calculate the average value of all the pixels
  average = average / pixel_amount;

  // Reset the Task Watchdog Timer (TWDT) to evade watchdog trigger
  esp_task_wdt_reset();
}

/**
 * Function: population level
 * Select a population level that represent the amount of people available in
    the room
 * Inputs: NONE
 * Outputs: NONE
 **/
void population_level(){
  population = 0;

  if ((50 <= average) && (average <= 59)){
    population = 1;
  }
  else if  ((40 <= average) && (average <= 49)){
    population = 2;
  }
  else if ((30 <= average) && (average <= 39)){
    population = 3;
  }
  else if ((20 <= average) && (average <= 29)){
    population = 4;
  }
  else if ((10 <= average) && (average <= 19)){
    population = 5;
  }
  else{
    population = 0;
  }
  // load the second position of the packet
  packet[1] = population;
  // Reset the Task Watchdog Timer (TWDT) to evade watchdog trigger
  esp_task_wdt_reset();
}
```

When running the needed code we get the following output:

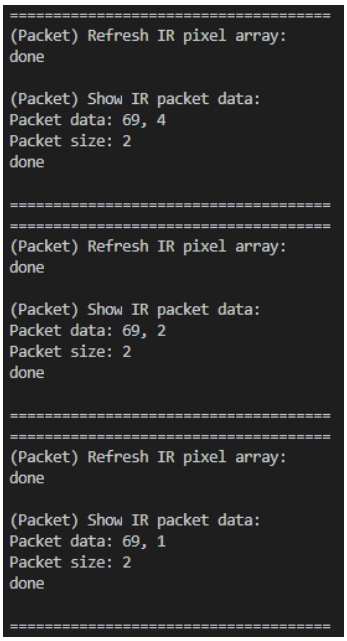

Figure 17: Output code packet option 2. We can see different reference values because of different mean values.

Packet size

When this option is used, a very compact packet is realized. The packet consists of the IR sensor address and the reference number (population). Since the packet consists of uint8_t values, the packet here will only be 2 bytes in size.

Address = uint8_t = 1 byte
Population = uint8_t = 1 byte
Amount of bytes = Address + Population = 1 byte + 1 byte = 2 bytes

8.2.3 Option 3

Overview

In option 3 we want to take it a step further. Here the location of the sensor is also taken into account. Because every hallway can be different (see later) we can tune the sensor so that it gives the best measurements to decide whether the hallway is blocked, partially blocked, or free. This gives us the opportunity to make the best interpretation of the hallway and is also the smartest and most accurate technique of all the options.

For clarification we can take a look at fig. 18, fig. 19, fig. 20 and fig. 21. These images show different hallway situations. When we take a look at fig. 18 we can see that the full hallway is blocked by people. Here the user can see that there is no free passage and that the user needs to evade that hallway. fig. 19 shows a situation where all the people that are present in the hallway are standing at the right side of the hallway. When using option 2 the hallway would appear to be blocked. Then this would result in a decision to select a high population level. This would be an imperfect interpretation of the situation because there is clearly free space at the left side of the hallway. When using option 3 the pixel are scanned to decide if there is a passage available. fig. 20, fig. 21 gives us an example of a more complex situation where there are two hallways.

Here we can take it a step further and scan if there is one or two free passage available. When looking at fig. 20 we can see that all the hallways are blocked. But when taking a look at fig. 21 we see that one of the hallways are partially free and there is a possibility to pass the people that are standing at the side of the hallway.

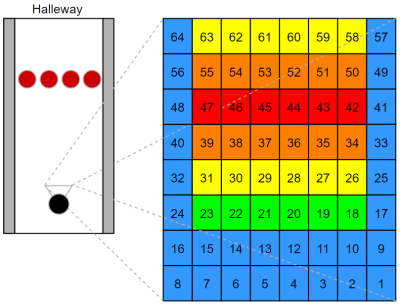

Figure 18: Situation where the hallway is fully blocked and where no passage is possible.

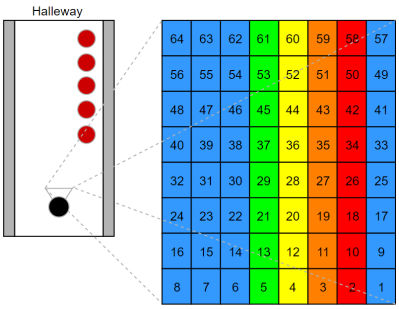

Figure 19: Situation where the hallway is at the left side free.

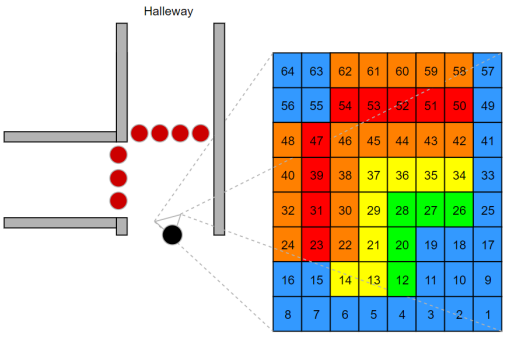

Figure 20: Situation where two hallway are blocked by people.

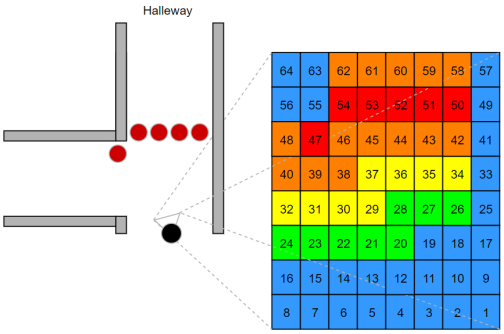

Figure 21: Situation where two hallways are shown and where one hallway is blocked and the other one is half blocked.

Code

This option is not programmed because the implementation depends on the hallway and is therefore better programmed on the field.

Packet size

The package size depends on the necessary calculations and the amount of information they bring to provide the wanted functionality.

8.3 Consumption

Because I (Pieter De Block) have no measuring equipment, I have not been able to measure the consumption of the sensor. Hence the results measured in the first semester by Jona Cappelle and Thomas Feys are used for further calculations. fig. 22 shows the consumtion of the IR sensor from the datasheet and fig. 23 shows the consumtion of the IR sensor measered by Jona Cappelle and Thomas Feys.

Energy Mode	Current
Normal	4.5 mA
Stand-by	0.8 mA
Sleep	0.2 mA

Figure 22: Consumtion of the IR sensor mentioned in the datasheet [8].

Power Mode	Verbruik [mA]	Tijd [s]	Energie [mJ]
Normal	4.8	0.105	1.6632
Stand-By	2.83	50	0.4669
Sleep	0.156	299.845	154.3602
		Totaal	156.4904

Figure 23: Consumtion of the IR sensor measered by Jona Cappelle and Thomas Feys, all credits to them [9].

fig. 24 show the diagram of operating mode.

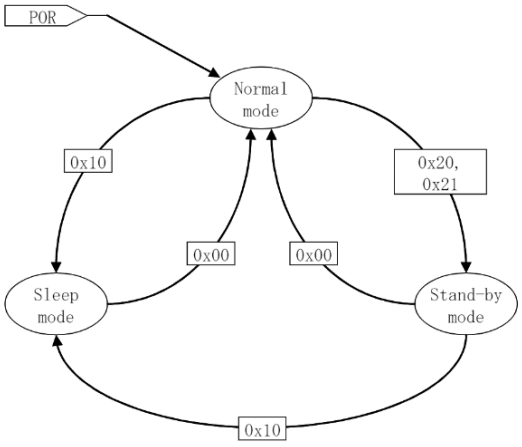

Figure 24: Diagram of Operating mode [10].

The setup time of the IR sensor on the other hand is equal to:

- Time to enable communication after setup is typical 50 ms
- Time to stabilize output after setup is typical 15 s

8.4 Conclusion

Some specifications were written down at the start of the design. The functional technical specifications stated for the IR sensor:

1. Monitoring presence of people (Are they blocking the hallway?)
2. The intensity of the crowd (How hard is it to pass through this hallway?)

When looking at the functional technical specifications, we can conclude that options 1 and 2 (partially) have been able to realize the wanted specifications. Option 1 is the simplest option because all pixel values must be read and forwarded to the server without having to perform calculations and decisions. In addition, sending all pixels is a drawback because many bytes are required per packet. But on the other hand it offers the possibility to provide the system with great functionality by performing calculations on the received pixel values in the powerful server. By sending these pixel values to the server, various calculations and decisions can be made via the server, which makes it possible to measure both the intensity of people present in the hallway and whether a hallway is blocked or not.// On the other hand, option 2 is a very simple and quick way to just know what the intensity is of the crowd in the hallway. In addition, option 2 makes it impossible to know whether a hallway is blocked or not. Why this option only offers the possibility to know the intensity is because option 2 calculates an average value of all pixel values and then selects a level to which a specific reference number belongs. This has the great advantage that the number of bytes per packet is small and less forwarding is required. Furthermore, it requires a little more processing power from our system, but it is far from being a problem.

In addition to options 1 and 2, option 3 was only theoretically discussed and not practical. The purpose of option 3 is to realize the functionality of option 1 on the system. Where the necessary decisions and calculations are performed on the system itself and not on the server. This ensures that we can meet the two requirements mentioned above. Here is the advantage that everything is done directly on the system and that the bytes per packet are lower than that of option 1. In addition, it will require more bytes per packet than with option 2. The disadvantage is that there are many more calculations here and decisions that need to be made and that there will also be more energy consumption.

So we can finally conclude that option 1 and 3 are the best options for realising the wanted functional technical specifications (presence, intensity, blocked, hard to pass, ...) and that option 2 is more an easy and fast way for only knowing the intensity and presence of people in the hallway. In addition to this, we can also conclude that in the future there are still many possibilities to optimize the system mentioned in Section 10.1.

9 Watchdog

There can be errors or problems in the software part of the design during runtime. When the design is not in hand range, it can be difficult to have to open the design during errors or problems and see what is wrong. That is why we use a watchdog. The watchdog will ensure that our design takes actions when these errors or problems occur and solves them if possible. For example, one action the system can take is to reset the system in the hope that it will work again after the reset.

The following will be discussed:

- How the watchdog works
- Code
- Conclusion

9.1 How the watchdog works

To make the watchdog work a task is used. The watchdog timer is activated when the task is created. The timer is given a specific time that can be set by the user and decide how long before the watchdog fire. After the task and the watchdog timer have been set, the timer will start running. When the timer reaches zero, the watchdog will take an action which is set by the programmer. Here the watchdog will reset the design. Another task may be to have the design run the stuck code again. There are also many more options. When using a watchdog there can be easy actions that need to be performed, and also advanced actions. In the future it is therefore possible to design an more advanced watchdog.

In the ideal situation the timer never reaches zero. This is possible by resetting the timer every time a method is executed in the code. Here this is realized by the following line of code:

```
1         esp_task_wdt_reset();
```

We can also see how the watchdog works in fig. 25.

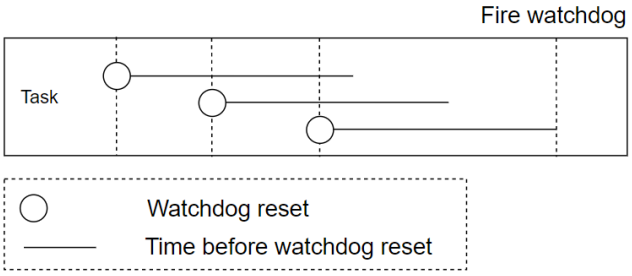

Figure 25: Overview of the watchdog task.

9.2 Code

Here you can see the code that is used to test the watchdog.

```
/**
 * Function: watchdog Task
 * Task for all the methodes that need to be checked by the watchdog function
 * Inputs:
 *        - void *myData Load Data in the watchdog Task
 * Outputs: NONE
 **/
void watchdog_Task(void *myData)
{
  printf("
    ##############################################################################\n");
  printf("Watchdog\n");
  printf("
    ##############################################################################\n");
  printf("# Running in myTask\n");
  printf("# Registering our new task with the task watchdog.\n");
  // Subscribe a task to the Task Watchdog Timer (TWDT)
  esp_task_wdt_add(nullptr);

  printf("# Resetting watchdog timer\n");

  // Reset the Task Watchdog Timer (TWDT)
  esp_task_wdt_reset();
  printf("# Packet\n");
  // Read pixel values, make the packet and print the packet data
  ir_packet();
  printf("# For loop\n");
  // Simulate the watchdog trigger
  for (int i = 0; i < 10; i++)
  {
      vTaskDelay(1000 / portTICK_PERIOD_MS);
      printf("Tick\n");
      printf("Resetting watchdog timer\n");
      esp_task_wdt_reset();
      if (i == 4)
      {
        printf("Test watchdog firing\n");
        vTaskDelay(2000 / portTICK_PERIOD_MS);
      }
  }

  // Test system reset
  // printf("# Test reset system\n");
  // resetModule();

  printf("# Removing our watchdog registration\n");
  // Unsubscribed a watched task from the Task Watchdog Timer (TWDT)
  esp_task_wdt_delete(nullptr);
  printf("# Ending Watchdog_Task\n");
  printf("
    ##############################################################################\n");
  // Delete task
  vTaskDelete(nullptr);
}
```

When running the needed code we get the following output:

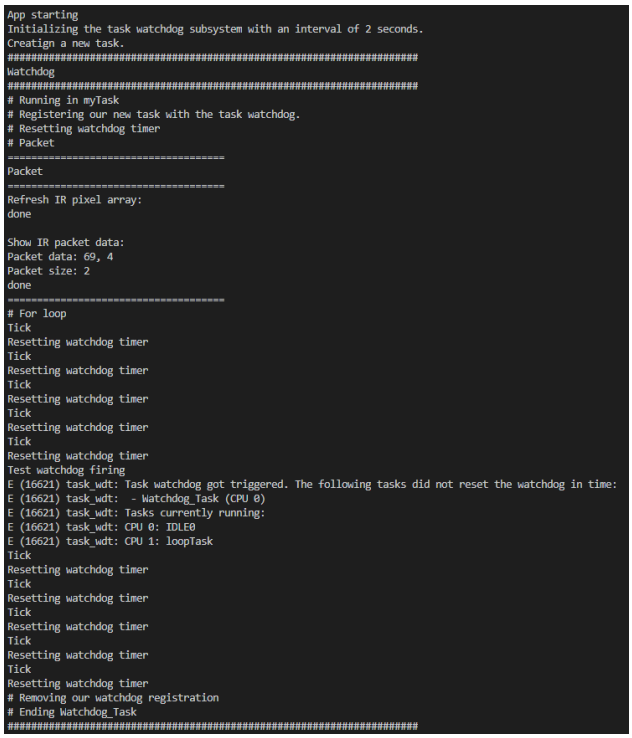

Figure 26: Output watchdog test with watchdog trigger.

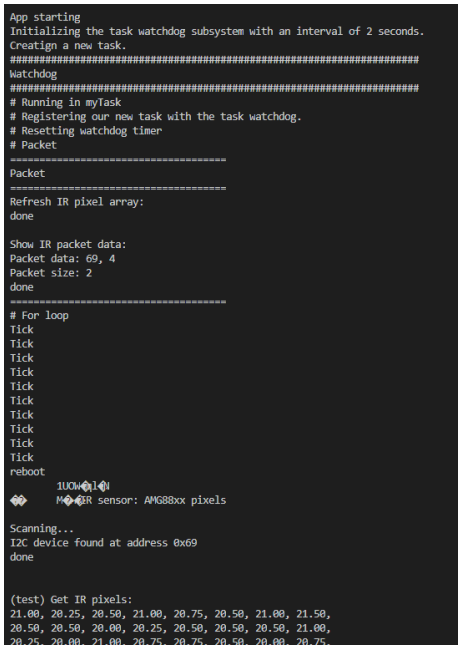

Figure 27: Output watchdog test with system reset.

9.3 Conclusion

We can conclude that this is a very basic implementation of the watchdog. There can still be made a lot of potential improvements in the future. The watchdog was tested to generate a trigger and an system reset. Both were tested separately to see that they work as expected. So that when we put the system together we can make an advanced system. Now there is no advanced watchdog system because the system has not yet been put together because we do not yet have a working system in hand. When we have a working system, we can adjust the watchdog step by step and improve it as desired.

10 General Conclusion

As show in this report, every part separately functions as expected. Unfortunately a whole system verification with the PCB hasn't been realized yet. In the following section some future possibilities are mentioned.

10.1 Future possibilities

- The system can be made more accurate using a IR sensor with a higher number of pixels. When the resolution is higher, smaller passages can be detected, even when there is a very high intensity of people in the hallway.
- More IR sensor can be used to cover more parts of the hallway. This has the disadvantage of increasing the power consumption.
- An app can be designed to guides the user undisturbed trough the campus. The app is used as an navigation system which shows the route. This can be achieved when multiple monitor devices are placed across the whole campus. An example of this idea can be seen in fig. 28. Here we can see that the green user wants to go to the green class. The setup makes it possible to realize this without problems.
- The system as it is implemented now is monitoring the whole time. This can be made more efficient measuring less during weekends and even classes. During classes the hallways are often not so crowded as between classes. This means that the system has a measuring pattern.
- The last step can be taken further by analysing the data to find patterns. This can't be done on the sensing device and needs to be done on the server because this requires a lot of processing power which would drain the battery of the sensing device. It would also require extra data storage. It also means that packet option 1 needs to be used which has the disadvantage of including more data and therefore needing more power for sending the data.

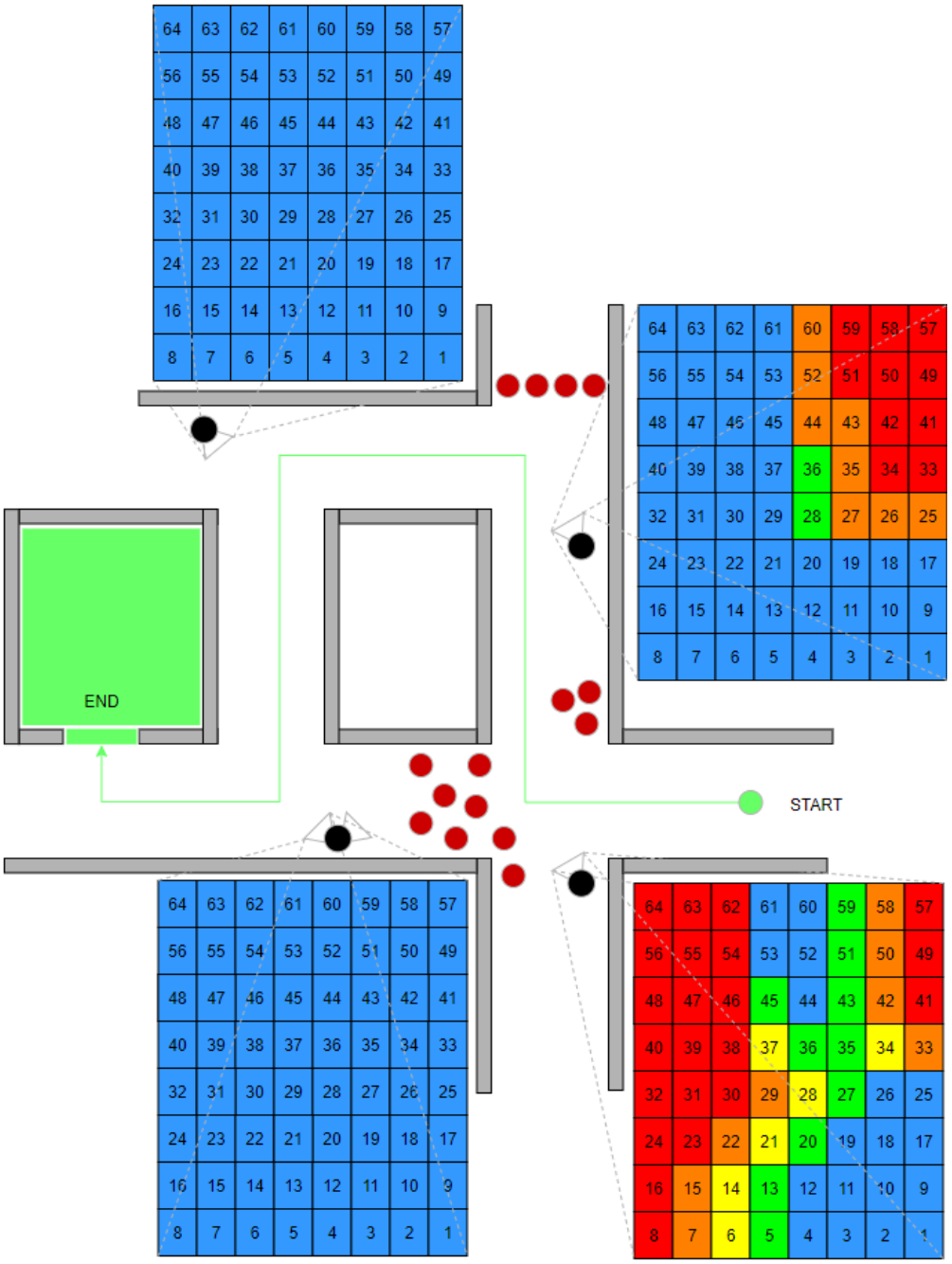

Figure 28: An advanced system design with navication.

References

[1] Espressif, "Esp32 frequently asked questions." [Online]. Available: https://www.espressif.com/sites/default/files/documentation/ESP32_FAQs__EN.pdf

[2] ——, "Esp32-solo-1 datasheet." [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-solo-1_datasheet_en.pdf

[3] L. M. Engineers, "Insight into esp32 sleep modes their power consumption." [Online]. Available: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

[4] "datasheet spw2430hr5h-b.pdf," https://www.knowles.com/docs/default-source/model-downloads/spw2430hr5h-b.pdf.

[5] "Ncs333 - operational amplifier, zero-drift, 10 uv offset, 0.07 uv/c dsatasheet," https://www.onsemi.com/pub/Collateral/NCS333-D.PDF.

[6] "esp32_datasheet_en.pdf," https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[7] "Wipy 3.0 - pycom," https://pycom.io/product/wipy-3-0/, (Accessed on 05/05/2020).

[8] P. Corporation, "Infrared array sensor grid-eye (amg88)," April 2017. [Online]. Available: https://cdn.sparkfun.com/assets/4/1/c/0/1/Grid-EYE_Datasheet.pdf

[9] J. Cappelle and T. Feys, "Embedded project." [Online]. Available: https://github.com/jonacappelle/Embedded_Project

[10] P. Corporation, "Infrared array sensor "grid-eye"." [Online]. Available: https://www.robot-electronics.co.uk/files/grideyeappnote.pdf