Faculty of Science and Technology

Middlesex University

Group Project Report Template

Andrei-Paul Țocu

Supervisor: prof. Raja Nagarajan

PDE2101 Engineering Software Development

BEng Computer System Engineering

February 2022

Table of Contents

1. Abstract

I believe that every individual on is trying their best to save up some money, get nicer things for themselves and just live a better life overall. There may always be a constant fear that there may be someone going through your personal belongings, especially when they live in a shared house. The motivation behind this device has to do with the drawbacks mentioned, as it is supposed to serve as an easy device to setup, that aims to at least keep track of abnormal movements in the absence of the user and hoping to deter the intruders when its alarm gets triggered.

2. Introduction

In this individual project of PDE2101 Engineering Software Development my aim is to create a motion sensor module that can be armed manually. Once armed and triggered, the device will be logging movements, duration of movements and the timestamp of these. After logging continuous movements for over 5 seconds the alarm will also be triggered. Alarm consists of a buzzer and a RGB LED.

In order to arm the device, there will be a button, that can act as a toggle for the on/off states of the device. Once armed, an LED will turn yellow/amber and there will be a short delay, up to 1 minute, till the device starts sensing and logging movements. This is to prevent false positive scenarios, like the user's movement after arming the device, before leaving the room.

3. Project objectives

Below is a list of the targets that this project is trying to achieve:

- Easy to be setup by the user
- Portable so its position can be changed easily
- Small form factor so that I can be placed out of sight
- Quick arming method at the push of a button
- Continuous tracking of the surrounding environment
- Reliable motion detection, minimizing false positives and true negatives
- Provide the user with logs of the movement detected since armed
- Deter intruders with the built-in alarm

4. Functional and Non-functional requirements

- Detection needs to happen fast
- It needs to be able to operate overnight
- It needs to be able to pick up motion from up to 5 meters
- It needs to be triggered by a silhouette of an average sized individual
- The system must perform with a minimum of false positive and true negative scenarios
- The device needs to be easy to operate by the end user
- Alarm must be triggered when motion is detected over a certain amount of time

5. Deliverables

- Project proposal
- Design drawings
- UML diagram
- Engineering report
- Research methodology
- Development methodology
- Testing results
- Device prototype
- End User's Manual
- Short presentation
- Demo

6. Ethical issues

The presence of Internet of Things (IoT) devices in developed countries makes it impossible for the individuals to have a healthy social life without being subject to tracking or monitoring in some sort of way. From government or other private and public companies, the streets are filled with CCTV, motion sensors, microphones and other beacons. Some would say that if you don't have anything to hide, there shouldn't be anything to be scared of. I believe that every person must have full access to their own civil and political rights as well as to economic, social and cultural rights. Rights to life, privacy, safety etc. and should never breach someone else's.

In the case of this project, its scope is to keep away these individuals that might attempt a burglary, theft or any kind of invasion of privacy. The device can be armed in the unattended private and personal space of the user, such as their dorm room, that others should not have access in the absence of the owner, being subject to trespassing.

As the rightful owner of that designated space, they should be able to make it as secure as possible, as long as it does not impede with other's rights. This device only detects movements in a certain timeframe when there shouldn't be any and can't be used to determine the identity committing the wrongdoing and may only provide a rough estimate if a breach of privacy might have happened.

7. Design and research

While working on this project I will be taking advantage of the Agile (Anon., 2022) development methodology. Agile development methodology allows for software to be released in multiple iterations, each iteration providing regular improved efficiency, ease of finding and fixing defects, develop and implement new features etc. Each of the incremental release must undergo the following phases: planning, designing, implementation and testing while each of these phases are being monitored.

When deciding the brains of this project, there were two choices: Keyestudio's V4.0 Board (Arduino UNO) or Adafruit's ItsyBitsy nRF52840 Express, both being ARDUINO capable.

	Arduino UNO (Anon., 2021)	**nRF52840 Express** (Anon., 2021)
Clock Speed	16MHz	64MHz
Operating voltage	5v	1.7v to 3.3v
Flash memory	32KB (0.5KB used by bootloader)	1MB
SRAM	2KB	256KB
Dimensions W/L/H	53/75/14mm	51/23/7.2mm
Other	PWM pins	Supports Floating-Point Unit (FPU), Bluetooth LE 5.3, NFC, ANT communication, PWM pins

Breaking down their specifications and comparing their SoCs, Arduino UNO has a 16MHz CPU, 5 times slower than nRF52840's, 64MHz. nRF52 offering roughly 30x more Flash memory, 10x more SRAM and many more communication options such as Low Energy Bluetooth, ANT and NFC.

Just to cut it short, the nRF52840 Express is better in every way than the Arduino UNO, therefore, I will be using the Adafruit ItsyBitsy nRF52840 Express as the main board of this project.

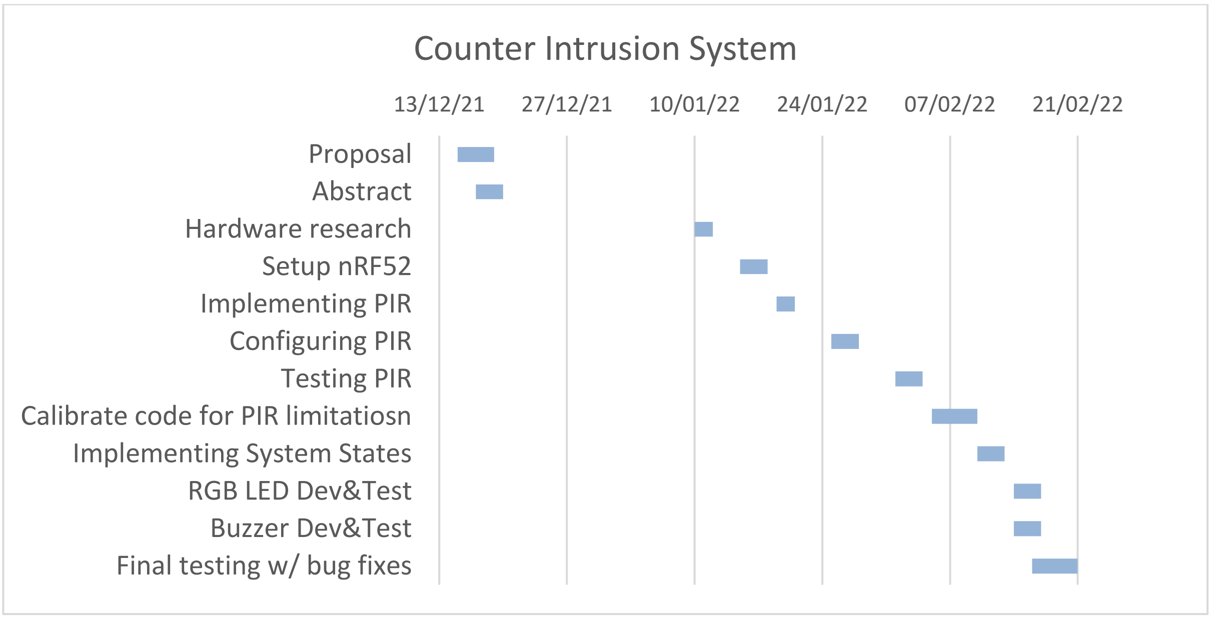
From the ELEGOO's 37 Sensor Kit I will be taking the HC-SR501 PIR (Passive infrared) motion sensor, one RGB LED, piezo speaker and a pull-up button. The motion sensor is working based on low-level radiation. As everything emits this kind of radiation, the hotter something is, the more radiation is emitted.

As for software, the mainboard and sensors can be programmed through the Arduino IDE, using "C/C++" as the programming language. In order to keep track of time, the device will also have a RTC (real time clock) module programmed, that will make it easier for logging using timestamps. The motion logs are stored onto the onboard memory of the nRF52, the file can then be downloaded and accessed from a computer through the micro-USB connection.

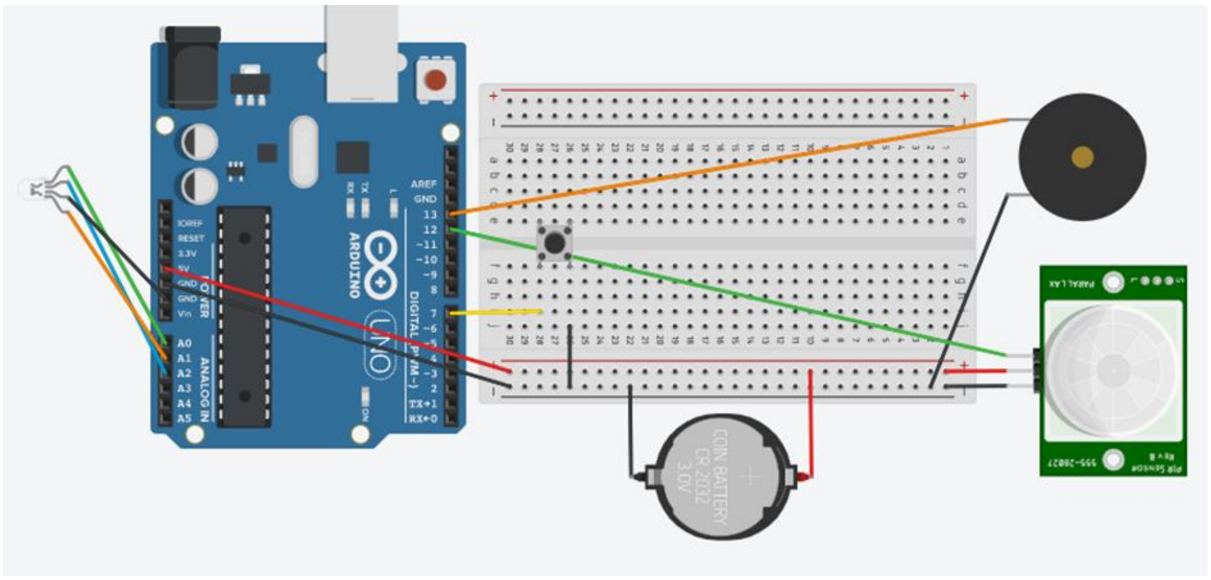
7.1. Timeline

Counter Intrusion System	**Start date**	**Duration**
Proposal	15/12/21	4
Abstract	17/12/21	3
Hardware research	10/01/22	2
Setup nRF52	15/01/22	3
Implementing PIR	19/01/22	2
Configuring PIR	25/01/22	3
Testing PIR	01/02/22	3
Calibrate code for PIR limitatiosn	05/02/22	5
Implementing System States	10/02/22	3
RGB LED Dev&Test	14/02/22	3
Buzzer Dev&Test	14/02/22	3
Final testing w/ bug fixes	16/02/22	5

7.2. Gantt Chart



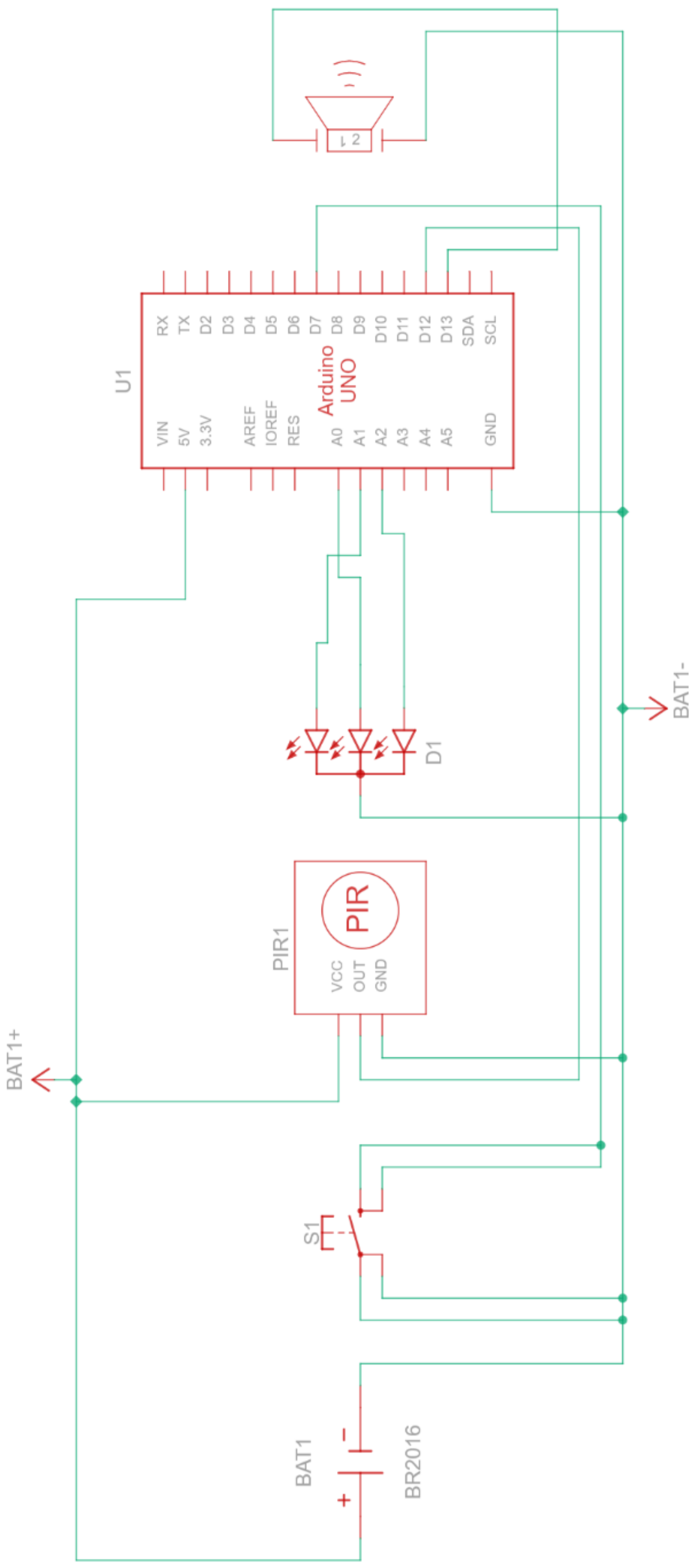
7.3. Wiring – TinkerCad Circuit (Tocu, 2022)



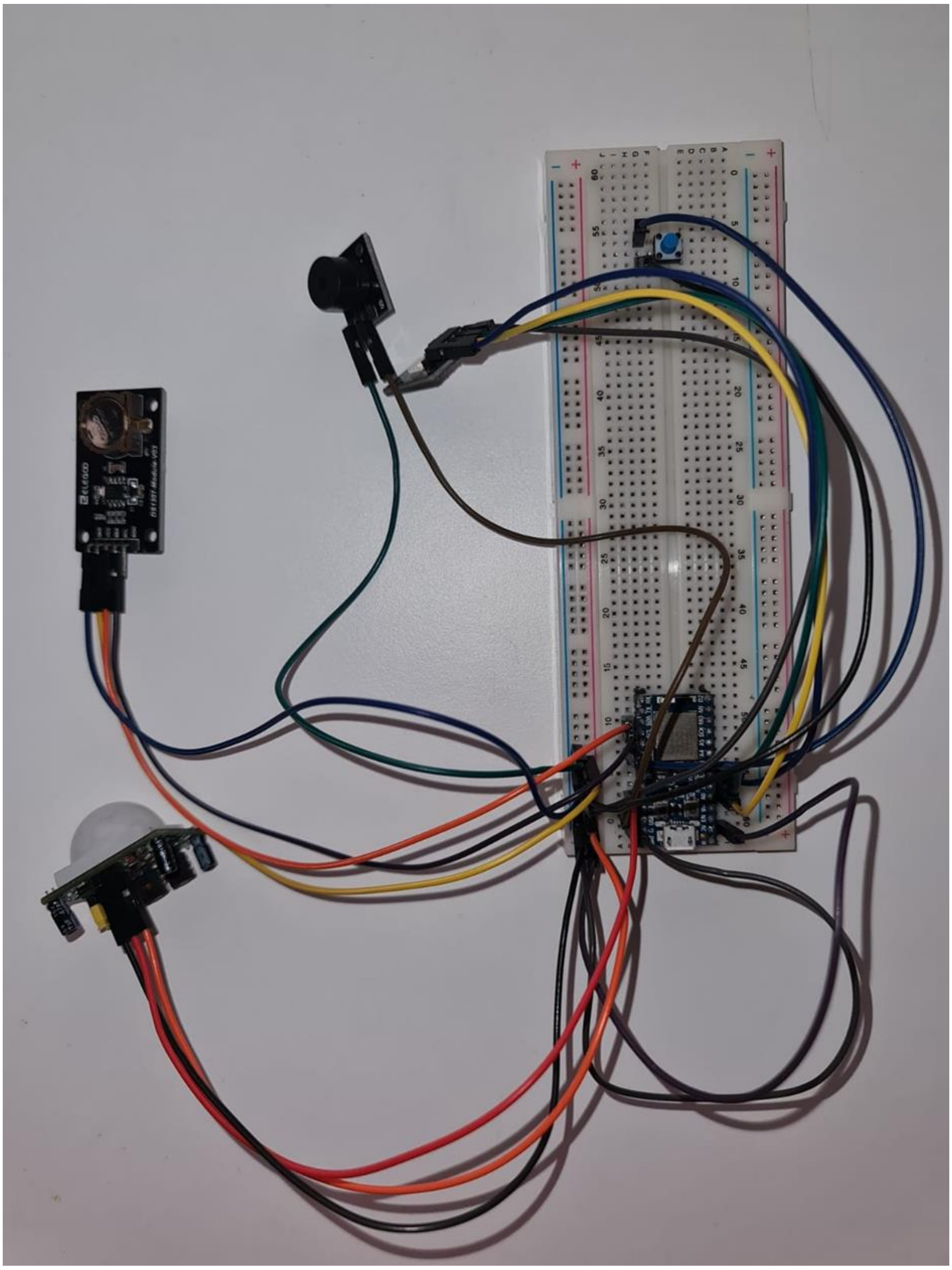
TinkerCad unfortunately doesn't have an nRF52 to represent the schematics more accurately, so I used an Arduino UNO. They differ in the physical dimensions but pin layout wise are quite similar.

It also doesn't have an RTC module to add to the circuit, so I added a 3v battery because it looks somewhat similar to the RTC. The battery only uses two terminals, positive and negative and doesn't have any other pins. The battery is missing SDA and SCL pins which would need to be connected to the SDA and SCL respectively on the board.

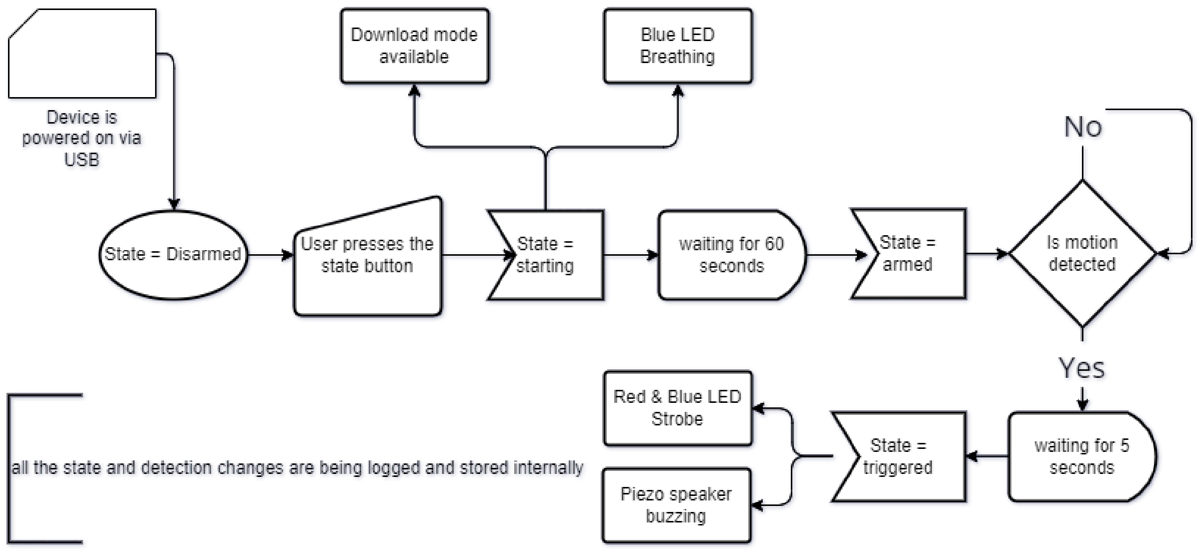
7.4. Wiring Schematic – TinkerCad Circuit (Tocu, 2022)



7.5. The finished device in real life



7.6. UML flowchart diagram



8. Testing, troubleshooting and development

8.1. Motion Sensor - HC-SR501 PIR (ada, 2014)

In terms of the physical connection, the motion sensor has three pins that need to be connected:

- 1 GND
- 1 Power (3.3v / 5v)
- 1 Data

The data pin is the one though which the module though which the sensor communicates with the board. When connected and properly initialized the motion sensor is outputting a digital signal, 0 and 1 respectively. Therefore, the board's pin to which the sensor needs to be connected to must be capable of digital communication, not analog. When it detects motion, it outputs 1 and 0 otherwise. This PIR module also has two trimmer potentiometers that control a timer and the sensitivity.

The sensitivity can be increased by turning the potentiometer clockwise. In our setting and testing, having the sensitivity set to maximum yielded best results. Based on the tests the sensor was able to pick out motion from a longer distance and finer movements.

The timer potentiometer is dictating for how long the output will stay high after motion was detected. Since in our use case the entire product is supposed to detect any kind of motion in a supposedly empty room, we want the timer to be as low as possible, this providing more accurate reading of the environment and helps minimizing false positives.

This PIR module also has special triggering headers, labelled L and H on the back of the board. Position L makes it so that the PIR will operate in a non-retriggering mode, which means that when it detects motion, it will output a high value for how long the timer trimpot has been set for, triggering being inhibited whilst it outputs high. Setting the header on H makes it so that retriggering is possible whilst still outputting HIGH. For this project the H header will be used as we need continuous monitoring and feedback of motion.

During the testing phase of the motion sensor, different configurations, physical setups, motions and distances were used. The final configuration is as follows:

- Timer set as low as possible
- Sensibility set to maximum
- Header set for retriggering (H)

In testing the motion sensor can be triggered from a wide range of distances: from 15 cm from the sensor all the way to 5 meters. Due to physical limitations more than 5 meters of distance were not possible for testing.

Before activating the motion sensor, it is required that there is no movement for the first minute, so that the sensor can properly calibrate. For this measure, the device has a state called “Arming” in which previous variable used in the detection loop are reset and also this state will have a timer of 1 minute before changing to the state “Armed” in which the detection is active.

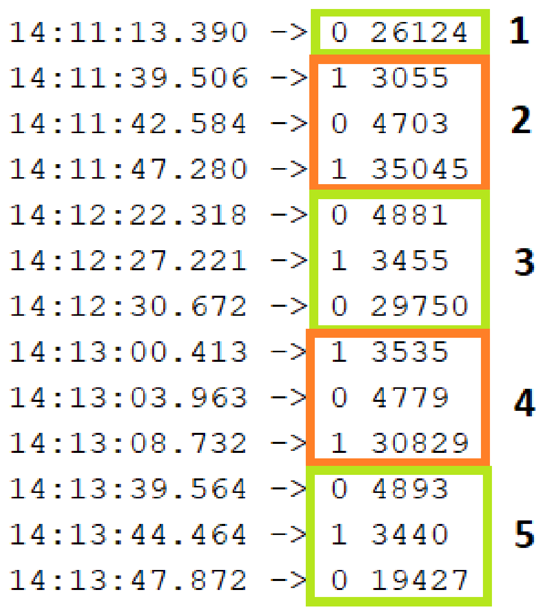
The following piece of code and many more were developed to determine the states which the motion sensor outputs, and what are the timings of these; how small of a movement can trigger it, how true to real-time the detection is, determining the calibration factors and trying to figure out its base rate bias or classification (true vs. false vs. positive vs. negative) etc.

```
int motionPin = 12, laststate; // digital PIN 12 used for PIR to board communication
bool curr=1, prev=!curr; // previous state and current state of PIR (0, 1)
                        //setting previous state as inverted of current state
void setup() {
  Serial.begin(115200); // starting Serial communication on baud rade 115200
  while(!Serial) {} // Infinite empty loop, waiting for the Serial to be ready
                      before proceeding
  pinMode(motionPin, INPUT); // Setting pin 12 for INPUT communication
}

void loop() { // Start of the infinite loop
  curr = digitalRead(motionPin); // constantly getting current sensor's state (0 –
                                            no motion, 1 otherwise)
  if(prev != curr){ // when a change is detected
    Serial.print(" "); // display a spacing charactecd, ' ', for easier reading
    Serial.println(millis()-laststate); // display the amount of time for current
                                 state, println demands newline return after output
    Serial.print(curr); // output on a new line current state of PIR
    prev = curr; // updating previous state to current state
    laststate = millis(); // snapshot of current time for when PIR states changed
  }
}
```

For testing we need a controlled environment, so we setup the board and the sensor in a room, facing the wall. To be sure there are no movements to be picked up we make sure the lighting will not create shadows that the PIR will pick up as these can change its states. With the sensor facing the wall we can trigger it by moving our fingers in front of it for a known amount of time.

After powering up the board we can open the Serial Monitor while the code is running on the board, and we have an output. For the first 10 seconds after powering everything up, it is best to not trigger the sensor as it goes to some calibration.



```
14:11:13.390 -> 0 26124    1
14:11:39.506 -> 1 3055
14:11:42.584 -> 0 4703     2
14:11:47.280 -> 1 35045
14:12:22.318 -> 0 4881
14:12:27.221 -> 1 3455     3
14:12:30.672 -> 0 29750
14:13:00.413 -> 1 3535
14:13:03.963 -> 0 4779     4
14:13:08.732 -> 1 30829
14:13:39.564 -> 0 4893
14:13:44.464 -> 1 3440     5
14:13:47.872 -> 0 19427
```

1. In our output we can see in the first green area, labelled with a 1, that the PIR output 0 for about 26 seconds. The timer of 26 seconds was output only when we triggered the sensor, changing states from 0 to 1.
2. By moving our fingers in a circular motion in front of the sensor for about 35-40 seconds, the state got changed to 1. Unfortunately, due to how the sensor was designed we can see that after 3055 milliseconds of it being triggered, the sensor outputs state 0 for 4703, although the rest of 35045ms of motion were being picked up.
3. Green square labelled 3 represents the lack of any detection, as I intended to have no triggering movements for the PIR. This lasted for about 35 seconds which were again divided, biggest chunk being 29750ms, but before this we have another switch of states which can be seen in the following cases as well.
4. After the ~35sec pause, the PIR got triggered again for another 35 seconds followed by a 25sec pause.

From the above test we can see that there is anomaly for the output of the PIR:

- When there is motion being picked up, the sensor will output HIGH for 3-4 seconds, then will output LOW for 4-5 seconds, and only after that will output the rest of the state correctly
- Same goes for when there is a stop in movements; it will output LOW for 4-5 seconds, then HIGH for 3-4 seconds and then the rest of the state correctly.

8.2. RTC Real Time Clock module – DS1307 (lastminuteengineers, 2021)

The RTC has 4 pins that need to be connected to the board. To make it work properly the following connection was used (listing connections as RTC to nRF52):

- GND – GND, ground
- VCC – USB pin, 5v connection
- SDA – SDA, serial clock sync
- SCL – SCL, serial data

Through the SCL pin the RTC communicated with the board to synchronize data movement via I2C serial interface, SDA pin is used to the data input and output from I2C interface.

Once everything is connected to the nRF52 it is advisable that for the first test to reset the clock. In order to do that we must have the battery removed for 5 seconds and put it back in. This way we can setup the clock on the first power on.

As for code, in the heading of our script we must include the RTC library created by Adafruit (installing the library can be done from the Library Manager of Arduino IDE).

```
 #include "RTClib.h"
 RTC_DS1307 rtc;
```

After this, in the *setup()* function it can be initialized and check if the RTC is properly set. If the clock wasn't set since the battery removal, then it can be done inside the *if(! rtc.isrunning())* by using the function *rtc.adjust(DateTime(2022, 2, 21, 15, 45, 12))* – this sets clock to 21st February 2022, 15:45:12.

```
Serial.begin(115200);
while (!Serial); // wait for serial port to connect. Needed for native USB
if (! rtc.begin()) {
  Serial.println("Couldn't find RTC");
  Serial.flush();
  while (1) delay(10);
}
if (! rtc.isrunning()) {
  Serial.println("RTC is NOT running, let's set the time!");
  // When time needs to be set on a new device, or after a power loss, the
  // following line sets the RTC to the date & time this sketch was
compiled
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  // This line sets the RTC with an explicit date & time, for example to
set
  // January 21, 2014 at 3am you would call:
  // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
}
```

In order to make use of the values inside the RTC we need to create a *DateTime* data type into variable now. This way we can get values such as the year, month, minutes etc. using *now.year()*, *now.month()* and *now.minutes()* respectively. The following code snipped creates a string variable in which we make or own time format, "D/M/Y H:M:S" and returns its value as a string.

```
String getTimeStamp(){
    DateTime now = rtc.now();
    String datestr, timestr, datetime;
    datestr += now.day(); datestr += '/';
    datestr += now.month(); datestr += '/';
    datestr += now.year();
    timestr += now.hour(); timestr += ':';
    timestr += now.minute(); timestr += ':';
    timestr += now.second();
    datetime = datestr + ' ' + timestr;
    return datetime; //returns date and time using D/M/Y H:M:S format
}
```

From here we can just use *Serial.println(getTimeStamp())* in order to print the time from RTC to the serial monitor. And the same to store data in the log file.

8.3. RGB LED and buzzer, piezo speaker and system states

Our RGB LED has 4 pins that must be connected in order to take advantage of the red, green and blue LEDs. The LEDs will be used to represent the different states of the system, as there is no other way of knowing when we do not have a serial monitor open.

```
int R = A0, G = A1, B = A2; // R G B led pins
void setRGB(int r, int g, int b) {
  analogWrite(R, r);
  analogWrite(G, g);
  analogWrite(B, b);
}
void breatheLED(bool r, bool g, bool b) {
  int val = 1, asc = 1;
  while (val += asc) {
    setRGB(val * r, val * g, val * b);
    if (val == 255) asc = asc * (-1);
    delay(3);
  }
}
```

The speaker only has two pins that we will use: GND and signal pin, through which the board will send data. We are going to use built-in functions to send a 1kHz signal to the speaker, *tone(buzzer, 1000)*. To stop it from beeping we will use *noTone(buzzer)*.

The system has four states:

1. Disarmed – nothing is being logged, nothing is being monitored, only the green LED is breathing off and on.
2. Starting – the variables used to determine detection are being reset, the blue LED is breathing, and it waits for 1 minute before switching to armed state.
3. Armed – constantly looking for movement and keeps track of how long the movements were happening for, if there was movement for over five continuous seconds the state changes to triggered; LED is off to not have false positives due to shadows / light changes.
4. Triggered – alarm is triggered; red and blue LED flashing concurrently and speaker does an on and off beeping noise.

```
void stateChange() {
    ++state %= 4; // increment state, then assign state its value % 4 =
[0,1,2,3]
    String logging = "\nSystem State changed to " +
(String)systemStates[state];
    writeToFlash(logging);
    stateTimer = millis();
}
void stateChangeManual() {
  if (millis() - stateTimer > 1000) // manually change state no earlier
than once/sec
    switch (state) {
    case 0:
      stateChange();
      break;
    default:
      state = 0;
      String logging = "\nSystem State changed manually to " +
(String)systemStates[state];
      writeToFlash(logging);
      stateTimer = millis();
      break;
    }
}

void loop() {
    switch (state) {
    case 0: //disarmed
      break;
    case 1: //starting
      break;
    case 2: //armed
      break;
    case 3: //alarm triggered
      break;
    default: //other unknown states
      break;
  }
}
```

8.4. Writing data to internal Flash

Thanks to the Adafruit's development team, there is a library tailored for the nRF52 based on the LittleFS (Anon., 2021). This becomes available in the Examples section of the IDE after installing the Adafruit's nRF52 board (Adafruit, 2021), in the board manager. It comes with a script for formatting the flash, one for listing the contents of the flash, another one to read/write to flash and another to stress test the flash.

In order to do anything regarding the internal flash we need to include the following libraries in our header:

```
#include <Adafruit_LittleFS.h>
#include <InternalFileSystem.h>
using namespace Adafruit_LittleFS_Namespace;
File file(InternalFS);
InternalFS.begin();
```

For the purpose of this code and our example, the variable ‘file’ will be used to describe the object to which the system writes/reads. The logs will be stored into the file names “motionlogs.txt”.

In order to check if the file exists, we use *file.open(filename, FILE_O_READ)*. If this returns a 1, it means the file exists and we can read from it. Otherwise, a file with the name “motionlogs.txt” needs to be created in order to read or write to it.

To create a file called “motionlogs.txt” we need to call *file.open(filename, FILE_O_WRITE)* which returns 1 if it was successful – meaning we have writing access to it, anything else means it failed.

Writing into the file is being done with *file.write(contents, strlen(contents))* – variable ‘contents’ is the data that is being written to file, *strlen(contents)* returns an integer which is the size of the variable ‘contents’, that tells the function when to stop writing. If the file already exists, the function will just append to it, it will not be overwritten.

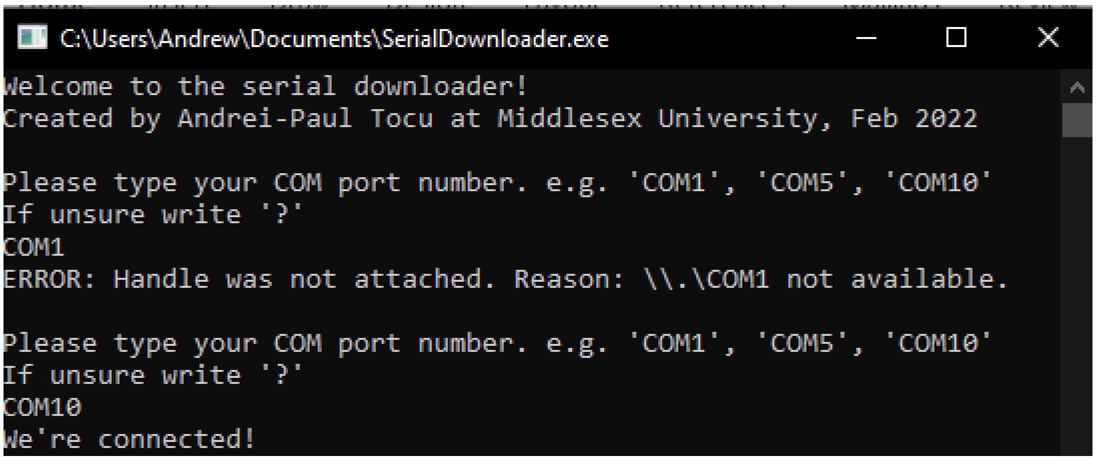
In our case, we created the function *writeToFlash()* which takes care of everything that we want written to file.

```
#define filename "motionlogs.txt"
void writeToFlash(String logging){ //variable logging is the content to
write
  char contents[64];
  int i;
  if (file.open(filename, FILE_O_WRITE)) { //making sure file exists/cre-
ated and we have write access to file
    strcpy(contents,""); //copy an empty string to contents
    for(i=0; i<=logging.length(); i++) contents[i] = logging[i]; //put each
characters of logging into contents
    contents[i] = '\r'; //last character of contents is carrage return to
mark the end
    file.write(contents, strlen(contents)); //writing to file contents
    file.close(); //closing file
  } else Serial.println("FAILED!");
}
```

8.5. Downloading data from flash

My first intention was to have the device uploaded with an algorithm that will do the output if its internal memory to a file on the PC. Unfortunately, this is not possible as the board only communicates with a PC through serial port. Thus, the device doesn’t have access to anything like writing/reading data directly to the host device.

Being limited to just the serial port, I managed to find on the internet the necessary library (Arduino, n.d.) to make a windows executable that is capable of send and receive the data via serial. The source code for these files (*SerialClass.cpp*, *SerialClass.h*) and the main.cpp of my application are included in the .zip file. When launching the executable is mandatory to specify the COM port that you want to access .

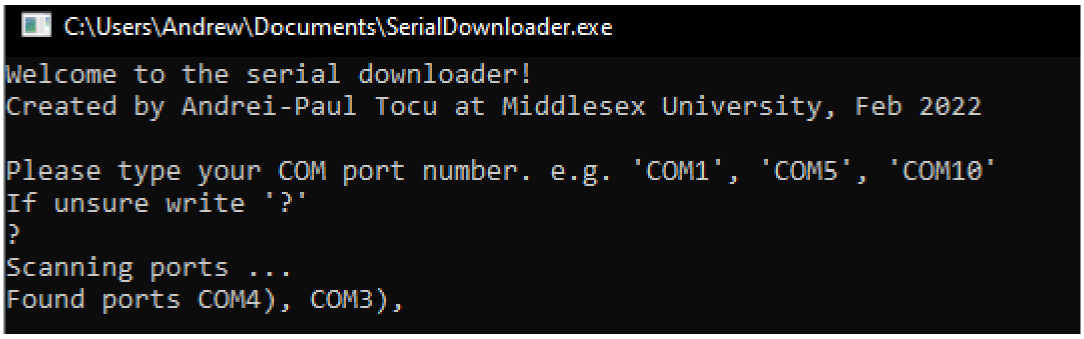


```
C:\Users\Andrew\Documents\SerialDownloader.exe
Welcome to the serial downloader!
Created by Andrei-Paul Tocu at Middlesex University, Feb 2022

Please type your COM port number. e.g. 'COM1', 'COM5', 'COM10'
If unsure write '?'
COM1
ERROR: Handle was not attached. Reason: \\.\COM1 not available.

Please type your COM port number. e.g. 'COM1', 'COM5', 'COM10'
If unsure write '?'
COM10
We're connected!
```

If the user is unsure about which COM port to access, there is a helpful feature that will scan and print all the ports that are available – only USB/COM ports can be accesses this way.



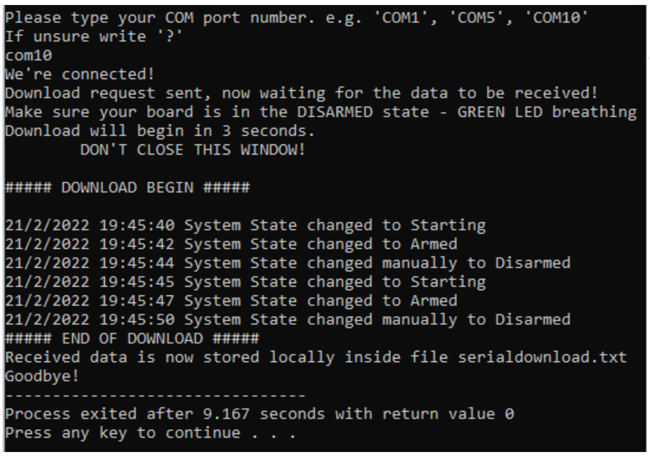
```
C:\Users\Andrew\Documents\SerialDownloader.exe
Welcome to the serial downloader!
Created by Andrei-Paul Tocu at Middlesex University, Feb 2022

Please type your COM port number. e.g. 'COM1', 'COM5', 'COM10'
If unsure write '?'
?
Scanning ports ...
Found ports COM4), COM3),
```

This will launch a command *wmic path Win32_SerialPort* through the opened *command prompt* window. This is a safe Windows built-in command that scans for all the communication ports the system has running. My script takes that report and puts it into a temporary file named *tempfile*, then is being processed line by line, then prints the found ports, making it easier for the user to choose. When picking the right COM port of the board, the communication will begin and any output from the board can be seen in this window.

In order to put the board into a download mode, it needs to be in the disarmed state (green LED breathing), as that is the only state which listens for the sequence that will start streaming contents of internal flash.

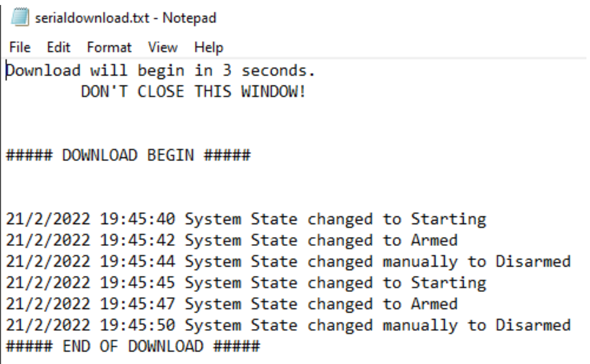
In my case will use COM10 and the board will have the file created and filled with some previous logs. This is how the output will look side by side with the content stored inside the file on PC:



```
Please type your COM port number. e.g. 'COM1', 'COM5', 'COM10'
If unsure write '?'
com10
We're connected!
Download request sent, now waiting for the data to be received!
Make sure your board is in the DISARMED state - GREEN LED breathing
Download will begin in 3 seconds.
        DON'T CLOSE THIS WINDOW!

##### DOWNLOAD BEGIN #####

21/2/2022 19:45:40 System State changed to Starting
21/2/2022 19:45:42 System State changed to Armed
21/2/2022 19:45:44 System State changed manually to Disarmed
21/2/2022 19:45:45 System State changed to Starting
21/2/2022 19:45:47 System State changed to Armed
21/2/2022 19:45:50 System State changed manually to Disarmed
##### END OF DOWNLOAD #####
Received data is now stored locally inside file serialdownload.txt
Goodbye!
--------------------------------
Process exited after 9.167 seconds with return value 0
Press any key to continue . . .
```



```
Download will begin in 3 seconds.
        DON'T CLOSE THIS WINDOW!


##### DOWNLOAD BEGIN #####


21/2/2022 19:45:40 System State changed to Starting
21/2/2022 19:45:42 System State changed to Armed
21/2/2022 19:45:44 System State changed manually to Disarmed
21/2/2022 19:45:45 System State changed to Starting
21/2/2022 19:45:47 System State changed to Armed
21/2/2022 19:45:50 System State changed manually to Disarmed
##### END OF DOWNLOAD #####
```

The program will read the lines received via serial, and if it will contain the substring "##### END OF DOWNLOAD #####" the program will close the file on the host machine and print the successful message shown above.

At the current stage of this program, it is necessary to have a Windows machine in order to download via serial.

9. Evaluation

For testing I placed the device next the entrance of a room, on a desk, about 1 meter off the ground and was pointing the door. Made sure it has power and that every module is connected. Right before leaving the room and closing the door I pressed the state switch button. The LED changed from green to blue meaning it was in the starting state, so I left the room.

I waited outside for almost 2 minutes and then got back in. I performed this test about 5-6 times with the device in the final release. When entering the room, it usually took about 10 seconds at most before the alarm was triggered. As long as there is no movement enough time for it to calibrate at first in the first 10-20 seconds when powered up, it should work as planned – after 5 seconds of continuous movement the alarm gets triggered, buzzing and blue-red LED flashes. The file also contains all the logs as expected.

10. Conclusion

The device is serving the purpose it's been built for, with the hardware provided by the Middlesex University and works as intended. There is always room for improvement, such as adding other sensors that detect the door opening, add a louder alarm, over-the-air notifications to phone or email or even a SMS system.

Of course, as the device is setup for now it can easily be shut down by an intruder or even stolen, which is quite a big drawback when its purpose is to deter such threats from happening. The list of improvements can go on and on, but what was expected from the device in this stage was already met.

11. References

Adafruit, 2021. *Adafruit_nRF52_Arduino/libraries/InternalFileSytem/examples at master · adafruit/Adafruit_nRF52_Arduino · GitHub.* [Online]
Available at:
https://github.com/adafruit/Adafruit_nRF52_Arduino/tree/master/libraries/InternalFileSytem/examples
[Accessed 18 February 2022].

ada, l., 2014. *PIR Motion Sensor.* [Online]
Available at: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor
[Accessed 11 February 2022].

Anon., 2021. *GitHub - littlefs-project/littlefs: A little fail-safe filesystem designed for microcontrollers.* [Online]
Available at: https://github.com/littlefs-project/littlefs
[Accessed 18 February 2022].

Anon., 2021. *keyestudio.* [Online]
Available at:
https://wiki.keyestudio.com/KS0497_Keyestudio_V4.0_Development_Board(Compatible_With_Arduino_Uno)
[Accessed 18 February 2022].

Anon., 2021. *nRF52840 Product Specification.* [Online]
Available at:
https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fkeyfeatures_html5.html
[Accessed 18 February 2022].

Anon., 2022. *What is Agile?.* [Online]
Available at:
https://www.atlassian.com/agile#:~:text=Agile%20is%20an%20iterative%20approach,small%2C%20but%20consumable%2C%20increments.
[Accessed 16 February 2022].

Arduino, n.d. *arduino.cc.* [Online]
Available at: https://playground.arduino.cc/Interfacing/CPPWindows/
[Accessed 20 February 2022].

lastminuteengineers, 2021. *Interface DS1307 RTC Module with Arduino.* [Online]
Available at: https://lastminuteengineers.com/ds1307-rtc-arduino-tutorial/
[Accessed 18 February 2022].

Tocu, A.-P., 2022. *Counter Intrusion System - TinkerCar.* [Online]
Available at: https://www.tinkercad.com/things/3FDIab3O7Km-counter-intrusion-system/editel?sharecode=fm--9JtociD9EFQl9Gc_iVZUcNxuebtRCHzp1h10aV4
[Accessed 18 February 2022].