G54MRT Coursework 2 Final Report

Title

Student ID: 4302467

Date: 8th May, 2019

Word count: 1992

Summary

My project is to design an automatic alarm system for shops, it can be activated automatically when all the staffs had left the shop and turned off when authorized staff sign in with a valid card.

Background and motivation

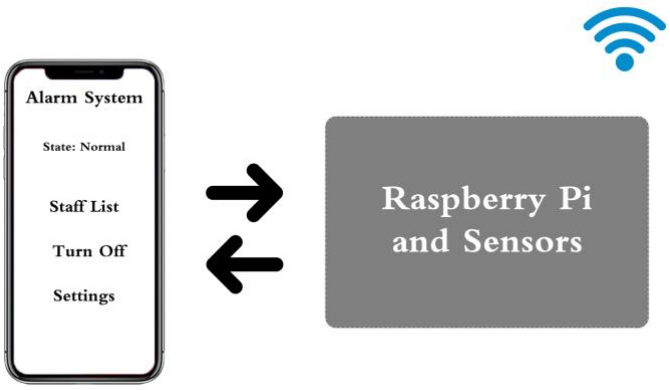
The alarm system is designed for shops, especially for those located near the street. Nowadays, shop burglary happens frequently in our daily life but many shops still haven't taken measures to prevent it. Some of the owners of the shops chose to install CCTV systems to record the video all the time, but CCTV cannot prevent or stop burglary happening. The only thing that CCTV can do is to provide video evidence for police. If the invader came with his (or her) face covered, then the video evidence will be useless. However, what if we can catch the thief on the spot or making a loud sound to force them to leave the moment they enter the store? There is no doubt that the loss will be reduced greatly.

Related work

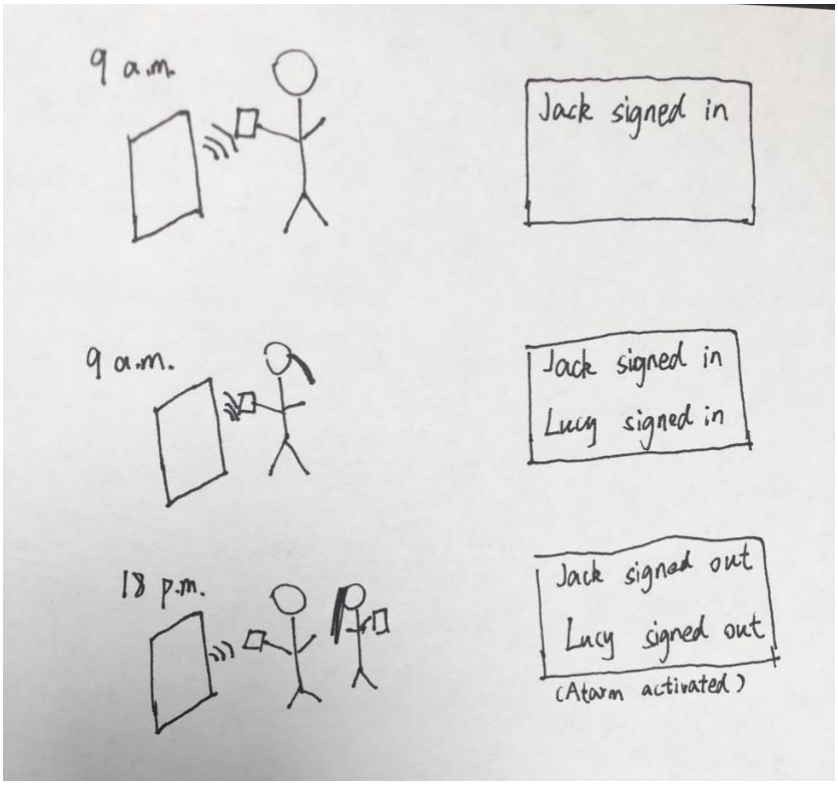
The idea of this project comes from a demo in the lecture called *Sneak Up* (ubicomp-sensor1, 2019), it enlightens me to think of how to build a system using various sensors to detect invaders. One ultrasonic ranger and one NFC tag are added in my design to realize more functions. Except that, I also learned several data filtering methods from (ubicomp-sensor2, 2019). Different filters are used to reducing noise for different sensors. Besides, I also learnt about ethical issues related to ubiquitous computing from Greenfield (2006).

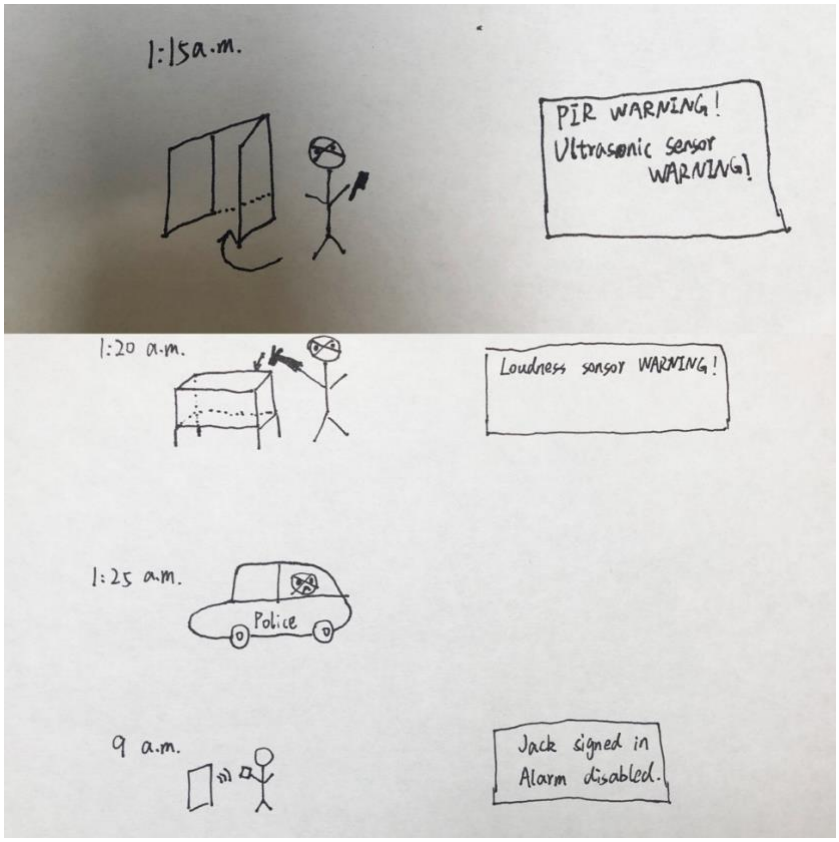
Design

The whole system consists of a Raspberry Pi with sensors and a mobile application which is used for user interaction.



The application has four main functions. First of all, the application allows the manager to manipulate with the employee information. Secondly, it also allows the administrators to change the settings of the system (if needed). Thirdly, it can do some data process of the allocated data. For example, the log of sign in / sign up time of each employee indicates their working hours. The last function is a warning. When the alarm system detects invaders, it will not only making a loud sound to force the invaders to leave, but also send a broadcast to the application to alert the user. The following diagrams indicate how this system works.

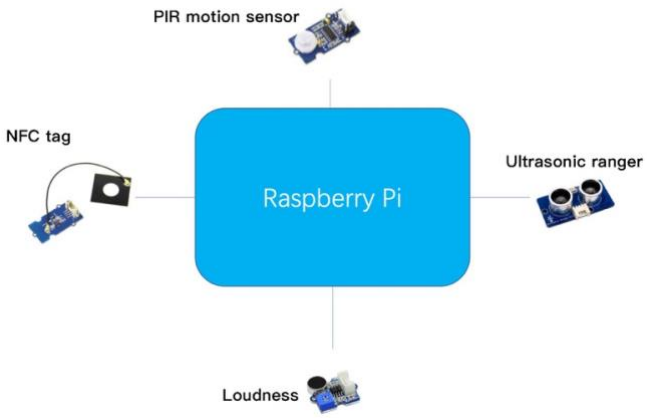




Considering the usage scenario, there are two factors that can trigger the alarm:

1. Someone makes a loud sound. (e.g. Break in by force, crashing of glass)
2. Someone moves in the area covered by the sensor

After looking up the description of each sensor provided by (Wiki.seeedstudio.com, 2019), I chose four sensors to implement the design: loudness sensor, PIR motion sensor, ultrasonic ranger and NFC tag. The record of signing in and signing out can be used to determine how many people are still in the shop and the alarm system will be activated automatically when everyone left the shop.

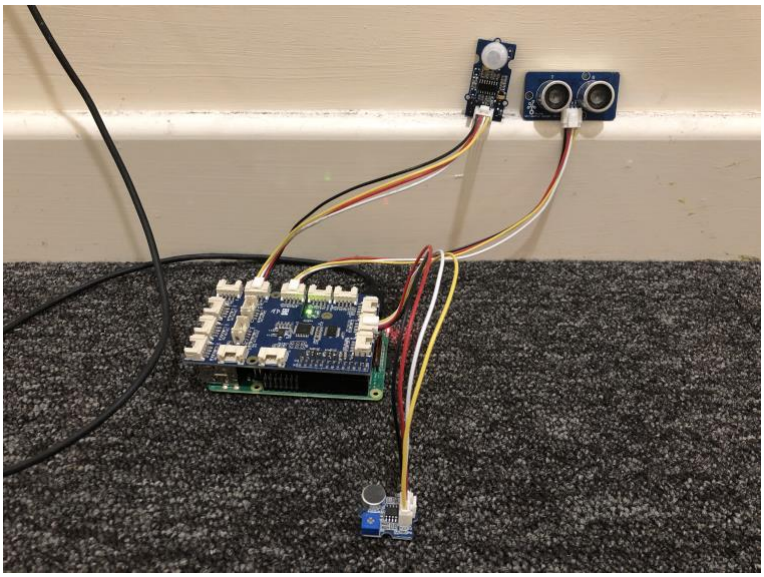


As for the mobile application, a cloud database can be used to store important data such as staff information and alert history. Those data will be firstly stored in local memory, and then synchronize them with the cloud database. The most obvious advantage of using cloud database is that the limit of local memory can be ignored and the safety of data can be guaranteed.

Implementation

Considering from personal interests and convenience of performing test, I bought a Raspberry Pi 3b+ and several sensors to build the system. Because of the limit of time, I didn't build the mobile application. Instead, an alternative alerting method is implemented using itchat (GitHub, n.d.). This is a python library which can send messages to WeChat account.

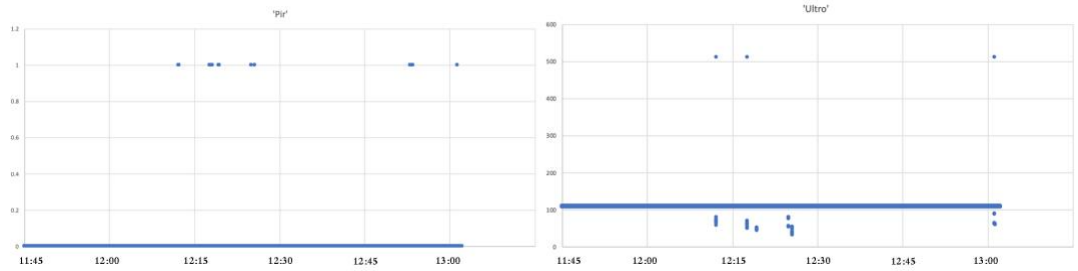
```
# Global variables
STATE = "ON"   # alarem state
LOUDNESS_SENSOR = "ON"
ULTRASONIC_SENSOR = "OFF"
PIR_SENSOR = "ON"
WECHAT_MESSAGE = "OFF"  # change to ON if you want to test it
```



The programming language used in this project is python. Generally, I created four threads to perform data collecting tasks and sensor state checking tasks simultaneously and a while loop in the main thread to check the state of the alarm system. The alarm will be a warning if any of those sensors' state turned into "WARNING".

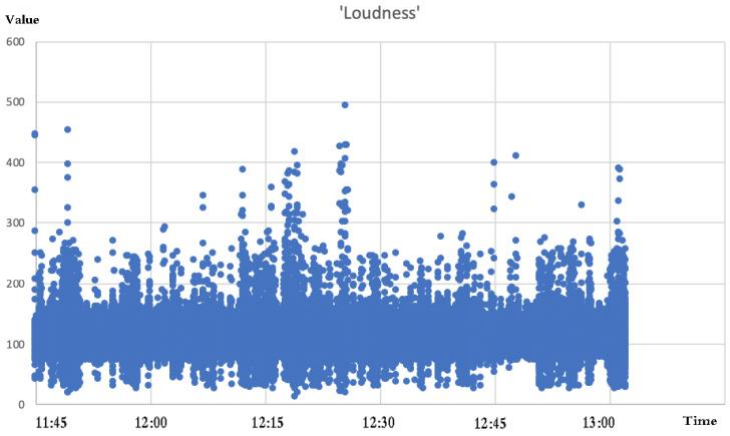
```
try:
  thread.start_new_thread(activeNFCReader,())
  thread.start_new_thread(validateID,())
  thread.start_new_thread(checkState,())
  thread.start_new_thread(collectSensorData,())
except Exception as e:
  print(e)

while True:
  if STATE == "ON":
    print(warning_message)
  elif STATE == "OFF":
    if myshop.getNumberOfStaffInShop() == 0:
      loudnessFilter.reset()
      movementFilter.reset()
      distanceFilter.reset()
      for x in range(10,0,-1):
        print("Alarm activate in " + str(x) + " second")
        time.sleep(1)
      STATE = "ON"
      print("Alarm on")
  elif STATE == "ALERT":
    print(warning_message + " Scan valid ID card to turn off!")
```

To reduce the false positive rate of the system, different filters are used to reduce the noise of the data. Both PIR motion sensor and ultrasonic ranger have stable performance, hence only one filter for each of them is good enough.



The median filter is used for PIR motion sensor because the output of the sensor is non-linear (0 or 1), while the low pass filter is used for an ultrasonic ranger to smooth the distance value.

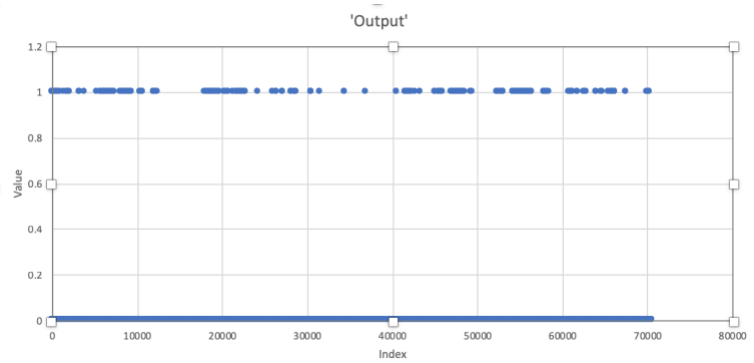
It takes more effort to figure out a way to process the data of the loudness sensor. As shown in the figure (x), the distribution of values collected in a quiet room covers a wide range and sometimes it is indistinguishable from human-made big noise.



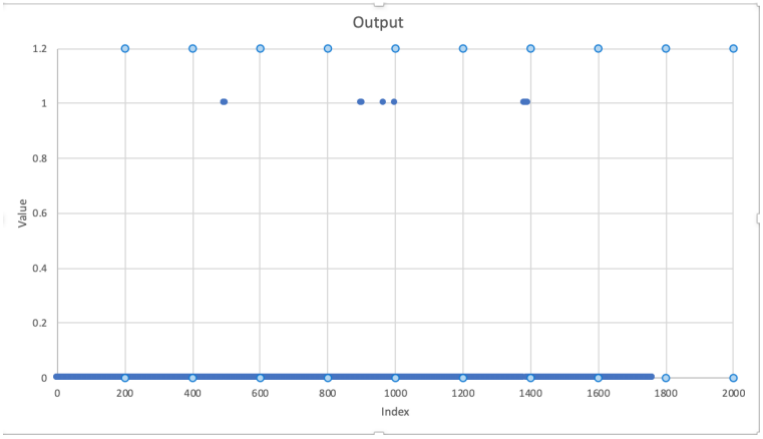
After trying different combinations of filters, I finally chose one solution which balances the false negative rate and true positive rate.

```
class LoudnessFilter(object):
  state = "NORMAL"
   def __init__(self, constant, highPassed, threshold, hhm_delay, chm_delay, ob_size,
ob_th):
     self.constant = constant
     self.highPassed = highPassed
     self.medianOfHighPassed = highPassed
     self.threshold = threshold              # threshold for high pass filter
     self.hhm_delay = hhm_delay              # delay of median filter for history high pass
     self.hhm = MedianFilter(self.hhm_delay) # history high pass median filter
     self.chm_delay = chm_delay              # delay of median filter for current high pass
     self.chm = MedianFilter(self.chm_delay) # current high pass median filter
     self.highPassFilter = HighPassFilter(self.constant, self.highPassed, self.threshold)
     self.ob_size = ob_size                  # output buffer size
     self.ob_th = ob_th                      # output buffer threshold
     self.outputBuffer = collections.deque(maxlen = self.ob_size) # mean filter
     for i in range(0, self.ob_size):
      self.outputBuffer.append(0)

   def addData(self, value):
    if self.num < 60:
      self.num += 1
      self.highPassed = self.highPassFilter.addData(value)
      self.cur_highPass = self.chm.addData(self.highPassed)
      self.diff = abs(self.cur_highPass - self.medianOfHighPassed)
      self.medianOfHighPassed = self.hm.addData(self.highPassed)

      if self.diff > self.threshold:
        self.outputBuffer.append(1)
        if sum(self.outputBuffer) >= self.ob_th and self.num > 55:
          self.state = "WARNING"
        else:
          self.outputBuffer.append(0)
```

Firstly, the input data will be processed by a high pass filter that turns the data into short term variations. After that, a median filter is used to store the history of high-passed value. This filter is designed to ignore the recent noisy value and the output acts as a benchmark. Then, the other median filter with lower delay is used to filtering the current high-passed value, which ideally could avoid the mutative values. By now, both current high-passed value and recent high-passed value are supposed to be immune from noisy values and the difference between them will be the change of ambient sound. If the absolute value of the difference is greater than the threshold, someone must have made a sound. However, the false positive rate of alert is still high after applying three filters.



Soon I realized that there is an underlying pattern of the data. When I made a big noise by closing the door, the ones tend to be sequential while the negative ones are dispersive. As a result, I added a mean filter to process the output. If the mean of a series output is larger than the threshold, it would be considered a loud sound. According to the scatter diagram (figure x), the result is much better than before.



The NFC tag performs good so there is no need to add filters. I tested NFC tag with several student ID cards (the chip is in the left bottom of the card), the output will be a list of numbers if it succeeds in reading the information and "None" otherwise. As I didn't buy NFC tag because of the high price, I also wrote some code to replace NFC reading with typing numbers. You can set the global variable NFC to “OFF” to use the typing function. As multi-thread is used in the system, some message may keep prompting while you are typing numbers and you can just ignore them.

The warning history is stored in warning_history.csv and the attendance history is stored in attendance.csv.

Testing

The following measurements are referenced from (Classifier evaluation with imbalanced datasets, 2019).

Where Accuracy = (TP+TN) / (P+N), Error Rate = (FP+FN)/(P+N), Sensitive = TP/P.

1. Unit Test

PIR motion sensor

I randomly walked around the sensor to generate motion data and then compared it with the ground truth.

Delay	Accuracy	FP	FN	Sensitive
5	0.98	5	0	1
10	0.98	2	0	1
15	0.99	1	0	0.99
20	1	0	1	0.99
25	0.98	0	3	0.90

When the delay increases, the false positive rate is getting lower and the false negative rate is getting higher. In other words, the system becomes less sensitive to the fast move if the delay is too large. In this case, the delay is set to 20.

Ultrasonic ranger

This sensor was tested by creating slow movement and then compared the result with the ground truth.

Constant	Accuracy	Error Rate	Sensitive
0.1	0.995	0.005	0.9
0.3	0.997	0.003	0.93
0.5	0.998	0.002	0.95
0.7	0.999	0.001	0.97
0.9	0.997	0.003	0.98

Similar to the PIR motion sensor, the constant value of ultrasonic ranger decides how sensitive the sensor is. When the constant is set to 0.7, the error rate is low and the sensitivity is not bad.

Loudness sensor

The selection of parameters for loudness sensor is quite complicated as I used four filters to process the data. The definitions of the parameters are given in the

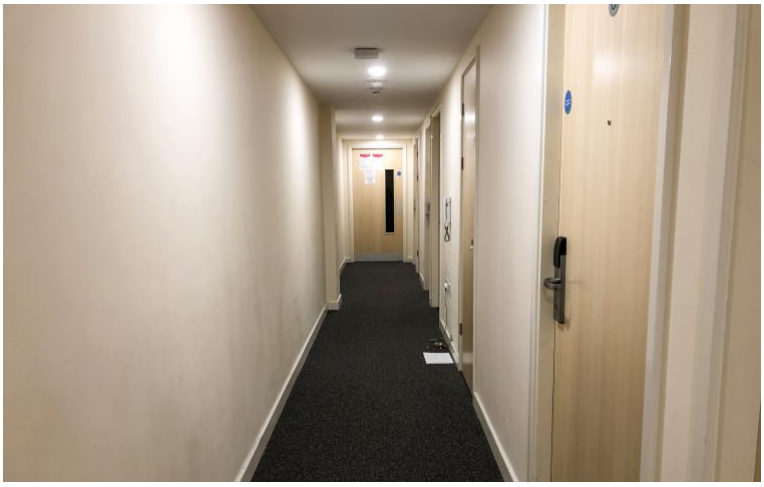
implementation of LoudnessFilter. To make it easier to find the relationship between each parameter and the effect on the result, the following table (partial) keeps true positive numbers and false positive numbers. The total number of positive samples is 50 and the total number of false positive samples is 950.

Threshold	HHM_Delay	CHM Delay	OB Size	OB Threshold	TP	FP	Sensitive
10	10	3	3	2	50	139	1
20	10	5	3	2	50	83	0.95
20	15	6	3	2	50	61	0.95
30	10	3	5	3	42	38	0.8
30	10	5	5	3	40	20	0.8
30	15	6	3	2	45	40	0.85
30	15	6	8	5	38	11	0.75

It is obvious that the lower the threshold is, the more sensitive the sensor is. However, when the threshold is below 30, the sensor has a non-acceptable error rate. To balance the true positive rate and the false positive rate, the last set of parameters is chosen.

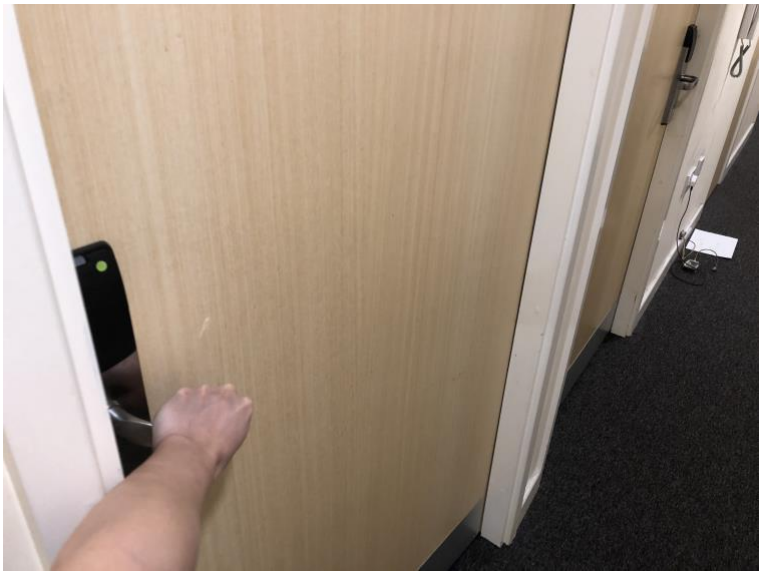
2. Integration Test

In this test, I integrated all the sensors to simulate the real use scenario. I set up the system in the lobby of my dormitory and invited my roommate to act as a thief who was trying to get into the kitchen.

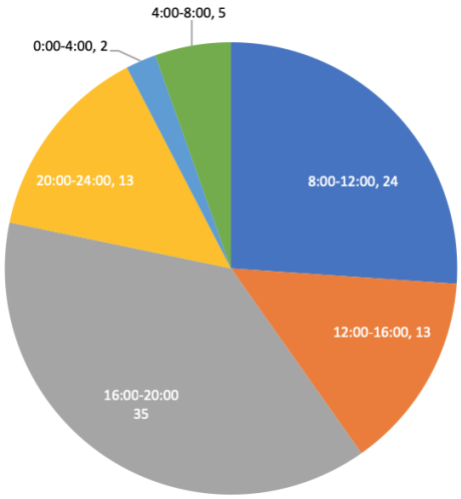


Not surprisingly, every time my roommate passed by the Pi, they were caught by my PIR motion sensor. One possible way to disable the PIR motion sensor is to cover it by a paper before the system is turned on (not likely to happen in the real scenario). In this case, the slow movement can be still detected by an ultrasonic ranger and I haven't come up with a good method to cheat the ultrasonic ranger. Assume that the invaders find a way to bypass two sensors mentioned above (e.g. break the window), the loudness sensor can also detect them. We tried to open the door which

behind those sensors, the noise can be detected by the loudness sensor with 80 percent chance.



I also collected the history of warning in two days and separated them into six time periods. The following pie chart shows the result, which is the number of detected move or sound in each time period. The overall result is quite close to the real scenario of the activities in our dormitory, except that warnings between 0 o'clock and 8 o'clock are definitely false positives because all of us were sleeping at that time.



Critical reflection

According to Greenfield (2006), there are five key ethical principles: do no harm, save time, disclosure, save face and plausible deniability. As my design aims at protecting people's property and simplify the sign in process, we shouldn't worry about the first two principles. Data allocating and processing of this system only

happens in off hours and inside the shop, so there is no need to concern about plausible deniability.

The first issue is that all the information about the alert system should be clearly disclosed to employees, for example, what kind of data will be collected. The second issue is the confidentiality of the history of attendance. If everyone has access to the database, those staff with unsatisfied attendance rate will be embarrassed. Moreover, if the data is leaked to malevolent people, the staff will be in danger because history can be used to analyze the behaviour pattern of them. Hence, access to the database should be limited to the manager of the shop.

Except for the ethical issues, there also exists a weakness of the system. As I mentioned above, the system can only rely on the loudness sensor if the PIR motion sensor were disabled and the invader found a path to bypass the ultrasonic sensor. In my implementation, I used many filters to process the loudness data to reduce the false positive rate. As a result, the small sound like hand clap cannot be detected by the system. One possible solution is using machine learning methods to determine the warning level. In this case, Artificial Neural Network (ANN) and Decision Tree might help to increase the sensitivity of the loudness sensor and reduce the false positive rate at the same time. This problem can be considered as a classification problem which has two labels, 0 for normal and 1 for warning. Both ANN and Decision Tree will require long term training to reduce the error.

References

1. ubicomp-sensors1. (2019) [ebook] Available at: https://moodle.nottingham.ac.uk/pluginfile.php/4579877/mod_resource/content/5/ubicomp-sensors1.pdf-.pdf [Accessed 7 Apr. 2019].

2. ubicomp-sensors2. (2019) [ebook] Available at: https://moodle.nottingham.ac.uk/pluginfile.php/4579878/mod_resource/content/7/ubicomp-sensors2.pdf-.pdf [Accessed 7 Apr. 2019].

3. Wiki.seeedstudio.com. (2019). *Seeed Wiki*. [online] Available at: http://wiki.seeedstudio.com [Accessed 5 Apr. 2019].

4. Classifier evaluation with imbalanced datasets. (2019). *Basic evaluation measures from the confusion matrix*. [online] Available at: https://classeval.wordpress.com/introduction/basic-evaluation-measures/ [Accessed 6 Apr. 2019].

5. Greenfield, A. (2006). *Everyware*. [ebook] Available at: https://moodle.nottingham.ac.uk/pluginfile.php/4579882/mod_resource/content/1/Pages%20from%20Everyware-The-Dawning-Age-of-Ubiquitous-Computing-.pdf [Accessed 7 Apr. 2019].

6. GitHub. (n.d.). *littlecodersh/ItChat*. [online] Available at: https://github.com/littlecodersh/ItChat [Accessed 7 Apr. 2019].

Appendix A - instructions

[Put instructions related to running / operating your coursework's code etc. here]

First of all, all sensors should be connected to the correct port. You can also change the code in main.py to match your own choice.

```
def collectSensorData():
      while True:
            if LOUDNESS_SENSOR == "ON" and STATE != "OFF":
                  loudnessFilter.addData(grovepi.analogRead(1))
            if ULTRASONIC_SENSOR == "ON" and STATE != "OFF":
                  distanceFilter.addData(grovepi.ultrasonicRead(2))
            if PIR_SENSOR == "ON" and STATE != "OFF":
                  movementFilter.addData(grovepi.digitalRead(3))

            time.sleep(0.01)
```

Secondly, you can choose the sensor you want to test by changing the state of sensors in main.py. If you want to test sending WeChat message functions, you will need to register a WeChat account and install itchat library by typing *pip3 install itchat* in the terminal.

```
STATE = "ON"    # alarm state
LOUDNESS_SENSOR = "ON"
```

```
ULTRASONIC_SENSOR = "ON"
PIR_SENSOR = "ON"
NFC = "ON"
WECHAT_MESSAGE = "OFF"
```

The staff information is defined in initialiseHumanResource() function. Please enter the ID of your card manually before you test it. The type of ID should be string.

```
def initialiseHumanResource():
      global staff, myshop
      if NFC == "ON":
            yanke = Employee("Yan Ke", "[115, 13, 157, 221]")
      else:
            yanke = Employee("Yan Ke", "4302467")
      jack = Employee("Jack Ma", "666666")
      tom = Employee("Tom Zhang", "333211")
      staff = []
      staff.append(yanke)
      staff.append(jack)
      staff.append(tom)
      myshop = Shop(staff)
      return
```

The last step is adjusting the parameters in main.py. You can change each of the parameter filters used to process the data (movementFilter doesn't need change). You can look up the definition in dataprocessfunction.py to figure out the meaning of each parameter.

```
def initialiseDataProcessor():
      global loudnessFilter, distanceFilter, movementFilter
      loudnessFilter = LoudnessFilter(0.5,0,30,15,6,8,5) # please refer
 to definition
      distanceFilter = LowPassFilter(0.7, 500, 80)  # change the third
one, cloest distance
      movementFilter = MedianFilter(20) # delay
```

To run the system, please type *python3 main.py* in the terminal.