



**ELEC3542 Advanced Programming  
and application development**

**Final Report**

**Bus Safety System**

**GitHub Link:**

**<https://github.com/ms042087/elec3542Project>**

**Ho Yu Hin**

**UID: 3035276696**

**Major: ElecE**

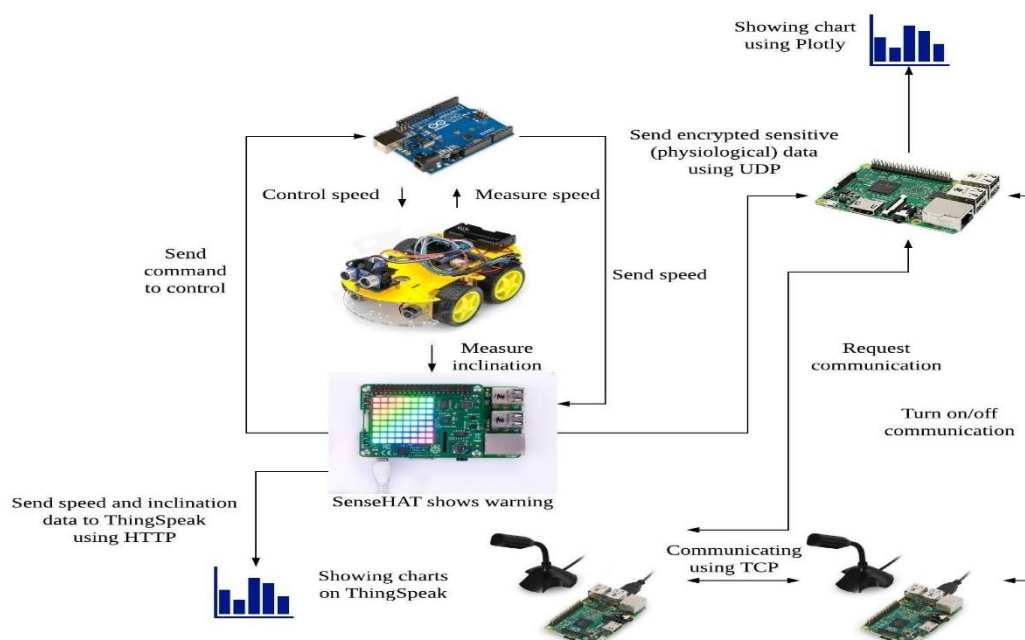
## 1. Introduction

Drivers need to wear a device that measures and monitors the physiological condition before driving. The system would compare the latest measured data with the data measured from the health test at the beginning of the year. The system then judges whether the driver is suitable for driving at that moment. The driver and the staff in the communication centre would be acknowledged if an abnormal condition occurs. A graph of physiological condition is generated in the server using Plotly. Also, the system would continuously measure the speed and inclination of the bus. It gives a warning if the speed or inclination exceeds a certain amount. These data would be uploaded to the ThingSpeak platform so that the staff can monitor it at any time.

Last but not least, if drivers encounter unexpected incidents and he/she wants to get some help, the driver can request the staff in the communication centre to help them through an intercom installed on the bus.

Buses company would like to purchase it since it makes the company become prestigious, as the system ensures the safety of both the driver and the passengers. More drivers would stay in the company and more passengers would take buses of this company. Also, the physiological data can be utilized to generate a health report, it serves as an additional fringe benefit to employees, such as the early detection of diseases.

## 2. Finalized System Architecture



### Hardware Components:

1. Raspberry Pi \*4

Pi Number	IP (In my home)	Function
51	192.168.0.10	<ul style="list-style-type: none"> <li>- Communication Centre.</li> <li>- Receive and decrypt the physiological data.</li> <li>- Judge whether the driver is suitable for driving</li> <li>- Plot a graph to visualize the data using Plotly.</li> </ul>
108	192.168.0.100	<ul style="list-style-type: none"> <li>- Intercom in the communication centre</li> <li>- Wireless Access Point</li> </ul>
109	192.168.0.104	Intercom in the bus
110	192.168.0.105	<ol style="list-style-type: none"> <li>1. Getting speed and inclination information from Arduino and senseHAT respectively. Upload them to the ThingSpeak platform.</li> <li>2. Getting heartbeat data. Encrypt it and send to the Pi in communication centre.</li> <li>3. Display warning message on senseHAT.</li> </ol>

2. microphone\*2, speaker\*2

Function: Intercom for the communication between the driver and the staff in the communication centre.

3. Arduino car

Function: Model the bus.

4. Magnetic sensor and magnet

Function: Stick the magnet on the wheel. Use magnetic sensor to get the speed of the bus through Hall effect by measuring the time difference between two instances that the sensor sensed the presence of magnet.

#### Software components:

1. Raspbian OS

2. ThingSpeak platform

Function: Plot graphs of speed and inclination

3. Plotly library and platform

Function: Plot a graph of heartbeat

#### Information exchange between components:

There are two types of data – sensitive and insensitive data.

Sensitive data includes the physiological data of the driver and it would be encrypted. In the simulation, the client sends the data to the server using UDP every second.

Insensitive data includes the speed and inclination of the bus. The data would be

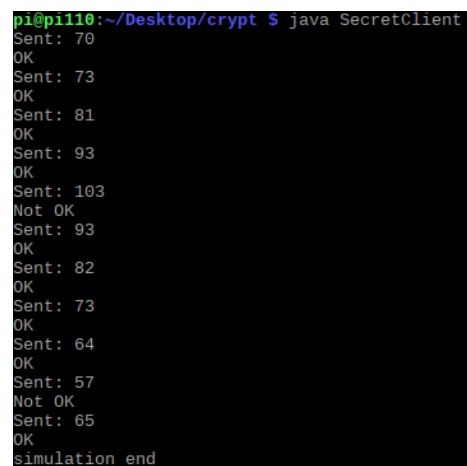
uploaded to the ThingSpeak platform using HTTP every two seconds. The senseHAT sends the inclination data to the Raspberry Pi every 0.2 seconds. For the intercom, the sampling rate of the voice is 44100 frames per second, size of each sample is 2 bytes, each frame is 24 bytes. Here we need to chunk the data so the stream flow would be smoother and prevent memory leaks.

The rate of sending the data is given by  $\text{RATE}(\text{number of frames per second})/\text{CHUNK}(\text{number of frame in each packet}) = 44100/8192 = 5.38$  per second. Hence, the voice data is sent every 0.186s.

### 3. Work Accomplished and Performance Analysis

Sensitive data would be encrypted as follows to protect the privacy of the driver.

1. Use **Diffie-Hellman key exchange** method for the client and server to exchange cryptographic keys securely.
2. Client gets/simulates the physiological data
3. Client encrypts the data through **AES encryption** and send to server
4. Server receives and then decrypts it
5. Server determines whether the body condition is good
6. Server sends the decision to client, tell the driver to stop driving if necessary.



```
pi@pi110:~/Desktop/crypt $ java SecretClient
Sent: 70
OK
Sent: 73
OK
Sent: 81
OK
Sent: 93
OK
Sent: 103
Not OK
Sent: 93
OK
Sent: 82
OK
Sent: 73
OK
Sent: 64
OK
Sent: 57
Not OK
Sent: 65
OK
simulation end
```

The simulation is shown in the screen capture above. The client sends data and receives the response from the server. If the heartbeat is smaller than 60 or larger than 100, the result will be Not OK, otherwise, the result is OK.

In the server side, the decrypted message will be stored in a csv file, as shown in the picture on the left. There is also a python code as shown on the right that creates a graph using **Plotly**.

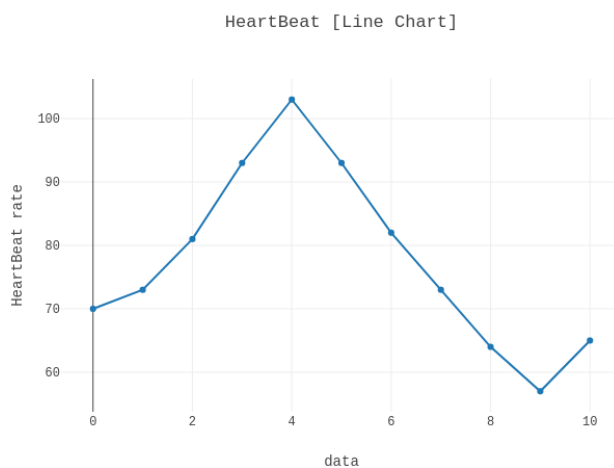
	A
1	y
2	50
3	60
4	70
5	80
6	81
7	82
8	83
9	84
10	85
11	86
12	93
13	75
14	78
15	83

```

7 import plotly.plotly as py
8 import pandas as pd
9 from plotly.graph_objs import *
10
11 py.sign_in('ms042087', 'Ytkty7FUBCOGMoFSX0w')
12
13 hb=pd.read_csv("test.csv")
14
15 data = Data([
16     Scatter(
17         y=hb["y"],
18         mode='lines+markers',
19         name="linear",
20         hoverinfo='name',
21         line=dict(
22             shape='linear'
23         )
24     )
25 ])
26
27 layout = Layout(
28     title='HeartBeat [Line Chart]',
29     font=Font(
30         family='Courier'
31     )
32 )
33
34 fig = Figure(data=data, layout=layout)
35 plot_url = py.plot(data,filename='HeartBeat [Line]')
36 py.image.save_as(fig, 'HeartBeat.png')

```

The graph generated by Plotly:



Insensitive data would be sent to the **ThingSpeak** platform. ThingSpeak is a cloud service designed for collecting, analyzing and visualizing data from IoT devices. Staff in the server side can use the link to view the graph of speed and inclination data of the bus. Here we will use the REST API based on HTTP requests that developed by ThingSpeak to update the information.

Code:

```

# update the data to ThingSpeak platform
def send(speed, inclination):
    api_key = "OQNDMGC09ZYVIYSF"
    data = {"api_key": api_key, "field1": speed, "field2": inclination}
    req = requests.post("https://api.thingspeak.com/update", data=data)
    return req.text

```

```

while(1):
    speed = getSpeed();
    r = getInclination();
    timeStampNew= time.time();
    if(timeStampNew - timeStampOld) > 2:
        send(speed,r);
        timeStampOld = timeStampNew
    if(exceedInclination(r) == True):
        alarm()
    time.sleep(0.2)

```

```

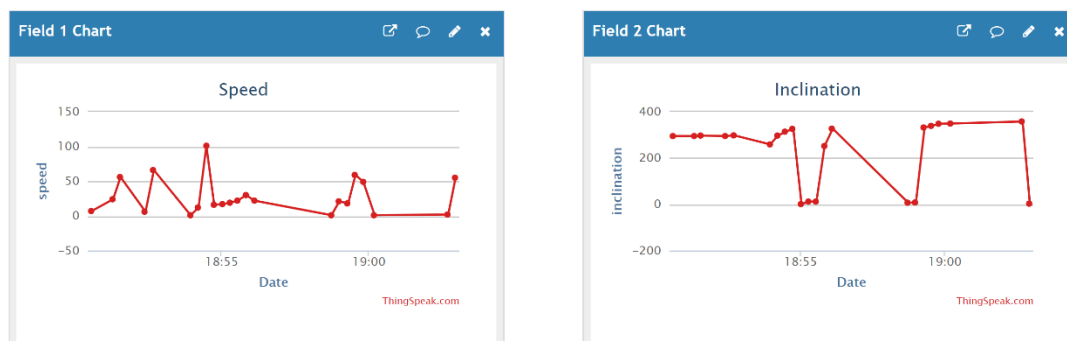
speed: 55
inclination: 3.0
inclination: 4.0
speed: 56
inclination: 4.0
inclination: 3.0
speed: 56
inclination: 4.0
inclination: 3.0
speed: 44
inclination: 4.0
inclination: 3.0

```

The picture on the left shows the main logic and the picture on the right shows the result.

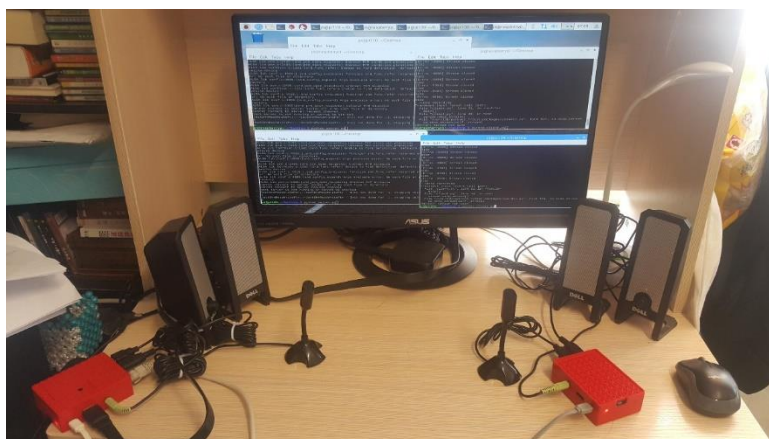
#### Main Logic Explanation:

We first get the speed from the Arduino via serial USB and the inclination from the senseHAT. Then using time stamp to **update the data to the ThingSpeak platform every two seconds**. If the inclination exceeds 20 degrees, there will be a warning appears on the LCD matrix of senseHAT. We measure the inclination every 0.2s. The picture in the right shows the output from the monitor.



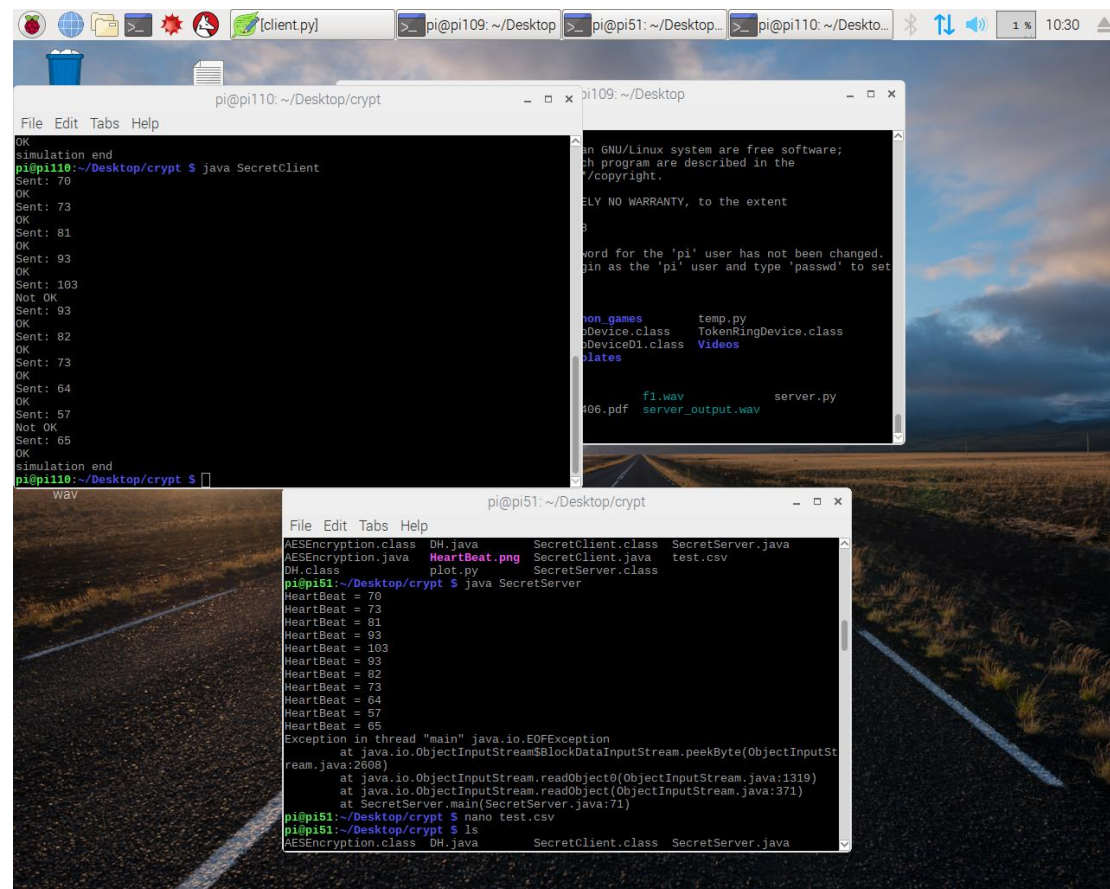
For the intercom, we connect a USB microphone and USB speaker to the pi. Here we use pyaudio library to record and replay the voice, and use socket library to create a TCP socket to transmit the data. The following link is a demonstration of this function.

<https://youtu.be/RTe1t6HdHxU>



## Pi Control

In this project, Pi108 would be used to control all other Pi using Secure Shell (ssh). Just like the following.



```
pi@pi110: ~/Desktop/crypt
File Edit Tabs Help
OK
simulation end
pi@pi110:~/Desktop/crypt $ java SecretClient
Sent: 70
OK
Sent: 73
OK
Sent: 81
OK
Sent: 93
OK
Sent: 103
Not OK
Sent: 93
OK
Sent: 82
OK
Sent: 73
OK
Sent: 64
OK
Sent: 57
Not OK
Sent: 65
OK
simulation end
pi@pi110:~/Desktop/crypt $

pi@pi109: ~/Desktop
File Edit Tabs Help
an GNU/Linux system are free software;
ch program are described in the
/copyright.
ELY NO WARRANTY, to the extent
3
word for the 'pi' user has not been changed..
pin as the 'pi' user and type 'passwd' to set

non_games      temp.py
oDevice.class   TokenRingDevice.class
oDeviceD1.class Videos
plates
f1.wav          server.py
006.pdf server_output.wav

pi@pi151: ~/Desktop/crypt
File Edit Tabs Help
AesEncryption.class DH.java SecretClient.class SecretServer.java
AesEncryption.java HeartBeat.png SecretClient.java test.csv
DH.class plot.py SecretServer.class
pi@pi151:~/Desktop/crypt $ java SecretServer
HeartBeat = 70
HeartBeat = 73
HeartBeat = 81
HeartBeat = 93
HeartBeat = 103
HeartBeat = 93
HeartBeat = 82
HeartBeat = 73
HeartBeat = 64
HeartBeat = 57
HeartBeat = 65
Exception in thread "main" java.io.EOFException
    at java.io.ObjectInputStream$BlockDataInputStream.peekByte(ObjectInputSt
ream.java:2688)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1319)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
    at SecretServer.main(SecretServer.java:71)
pi@pi151:~/Desktop/crypt $ nano test.csv
pi@pi151:~/Desktop/crypt $ ls
AesEncryption.class DH.java SecretClient.class SecretServer.java
```

## 4. Challenges in Implementation

One of the challenges is the implementation of intercom. The first challenge is finding a microphone and speaker that provide satisfactory performance. Not many high-end devices provide driver in the Linux environment, especially on raspberry pi. I would like to thank Macro, the TA, providing me a tested speaker. I also bought two USB microphones from TaoBao, but one of the microphone did not work well. There was a large variation in the volume of the recording. However, I can still use them to demonstrate the main idea successfully. Another challenge is to implement the voice communication in raspberry pi. I have tried to use TCP to send text data before, but not voice. Luckily, I found a library that is very useful to sample the speech. I also learned the basic principle of recording and replaying voice message in Pi.

Also, I realized that the inclination data that gets from the senseHAT is not accurate if we measure it with an interval  $>0.5s$ . After researching the reasons behind this, I found that the senseHAT combine a lot of data from the gyroscope and

accelerometer etc. to compute the inclination. Hence, the measuring interval must be small to get the rate of change in the motion and position of the pi accurately. However, the program also needs to update the data to ThingSpeak platform periodically and warns the driver if a dangerous condition occurs. These two functions cost some time and thus interrupt the measurement. Therefore, multi-threading must be used to perform these job parallelly. Timestamp is used to control the frequency of uploading the information to the ThingSpeak platform. For every two seconds, a new thread would be started. When the senseHAT needs to alert the driver, a new thread would also be started.

## **5. Possible Future Development**

I would analyze the possible future development from several perspectives, including the functionalities, power consumption, security and privacy, latency, and robustness.

### **Functionalities:**

#### **1. More accurate detection of abnormal conditions**

Since there are a huge amount of data, we can apply big data technique to analyze the data and make wiser decision automatically. To increase the speed of searching abnormality, we can apply machine learning algorithm.

#### **2. Smarter control on the bus**

Due to the advancement of technology in the field of computer vision and machine learning, we can also apply automatic driving on buses. It can take control of buses if needed.

### **Power consumption:**

Since this device is installed on the bus, we can apply energy harvesting technique to provide energy to it. For example, installing a solar panel on the roof of the bus would be a good choice since buses expose to sunlight at most of the time.

### **Security and privacy:**

The physiological data needs to be protected. DH key exchange is subject to the man-in-the-middle attack. Hence, we need authentication such as involving a certificate authority to verify the identity of each devices. Also, session key should be used. Since the key generation process is expensive, it is better to protect messages using light-weight session keys for a session. It also makes the attacker harder to crack the key.



**Latency and limitation in bandwidth:**

There is obvious delay in the ThingSpeak platform and the intercom. We can develop our own platform to visualize and analyze the data. We can use UDP for the intercom like Skype, but we need signal processing technique to reduce the size of data. For example, applying a high pass filter to filter the noise and using Huffman coding or lossy signal compression techniques involving wavelet could be a solution, it really depends on how much computational power we have for sampling, compression and transmission.

Also, huge amount of data is generated when the system scales up. Instead of each bus sending information to the server individually, we can apply data aggregation technique. Especially for some city having high buses density like Hong Kong, this technique can be easily applied to reduce the bandwidth requirement. Nearby buses can form a network, and then elect a leader with the strongest connectivity to gather the data, and finally report the data to the server. Besides, the data can be represented with sparse property since some data is changing slowly, that is, difference in measurement can be reported instead. For example, 1<sup>st</sup> measurement: 15 degrees, 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup>, 5<sup>th</sup> and 6<sup>th</sup> measurements also are: 15 degrees. Instead of sending 15,15,15,15,15,15, we can send 15,0,0,0,0,0 if appropriate scheme is developed. We can compress this data or use appropriate data structure to optimize the space complexity, and thus the amount of data needed to be sent is reduced.

**Robustness:**

Restarting system and self-healing system should be installed on the device to ensure resilience. If devices fail to work, for example measure the inclination, the system should restart and connect to the network again automatically.

Since data is transmitted to and analyzed in a cloud platform, we can design the platform to be loosely coupled. If one of the components fail to work, including die, fail and busy, the whole system would keep functioning.

## References

1. Lecture notes and lab exercises

2. PyAudio library

<http://people.csail.mit.edu/hubert/pyaudio/docs/>

3. ThingSpeak tutorial

<https://www.mathworks.com/help/thingspeak/getting-started-with-thingspeak.html>

4. Plotly tutorial

<https://plot.ly/python/>