GNG5140

Design Project User and Product Manual

Submitted by:

[Night Call Bell Team]

[Zizheng Fan, 300161358]

[Yacine Diagne, 7902246]

Date: 04/10/2021

University of Ottawa

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
NCB	Night Call Bell
UPM	User Product Manual
py	Python

Table 1: Acronyms

1 Introduction

This User and Product Manual (UPM) provides the information necessary for our potential clients and other groups that want to make improvements based on our products to effectively use the Night Call Bell (NCB) and for prototype documentation.

In the first part of this manual, we will explain the fundamental problems our client is facing, their basic requirements and how we take all the concepts into a solution. And in the next part, we will explain some professional and technical knowledge we have used in our design and how we realized our idea by using them.

2 Overview

Our client, Ms. Fran, is an elder lady with a physical disorder. Sitting in a wheelchair, she had difficulty moving her upper limbs and arms and could not pronounce words clearly. She and her caregiver, Ms. Fleur, hope that our team can design a night call bell, when the customer calls for help, the caregiver can receive the distress signal in time. Especially at night, the nurse sleeps soundly, so it is difficult to hear our client's asking for help. After three months of cooperation among team members, we completed the design and debugging of software and hardware, and finally produced a satisfactory product.

After group meetings and discussions, we classify the needs mentioned above into three categories according to their priority, from the most important to the least important. 1 means the most important and 3 means the least.

Needs	**Priority**
The device can quickly identify the voice of our client.	1
The device's ability to recognize sound will not be affected by background noise.	1
The device can quickly send alarm information to the staff.	1
It is best that the device can run without a network.	1
The operating device does not need to be borrowed through other devices such as mobile phones and computers.	1
The size of the transmitter is suitable for fixing on the table, and the size of the receiver is suitable for keeping in the pocket.	2

The device uses fixed power sources and sockets to provide power	2
The receiving end is prompted by optical signal and sound signal, and the transmitting end only uses optical signal.	2
The device has a good plastic package, preferably waterproof	3

Table 2: Client needs

As you can see, our design takes into account all the needs of our customers. Compared with other products, it is more portable, more stable, more friendly initialization process and faster response speed, and it has a better plastic shell.

Here we will show you how you could use our system. Remember the keyword is “Hey Yeah Hey Yeah”.

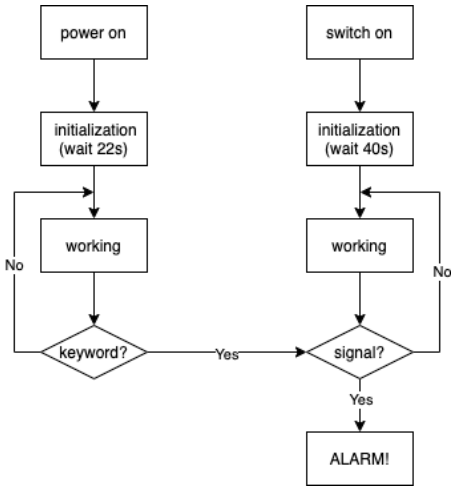

Figure 1: Simple flowchart

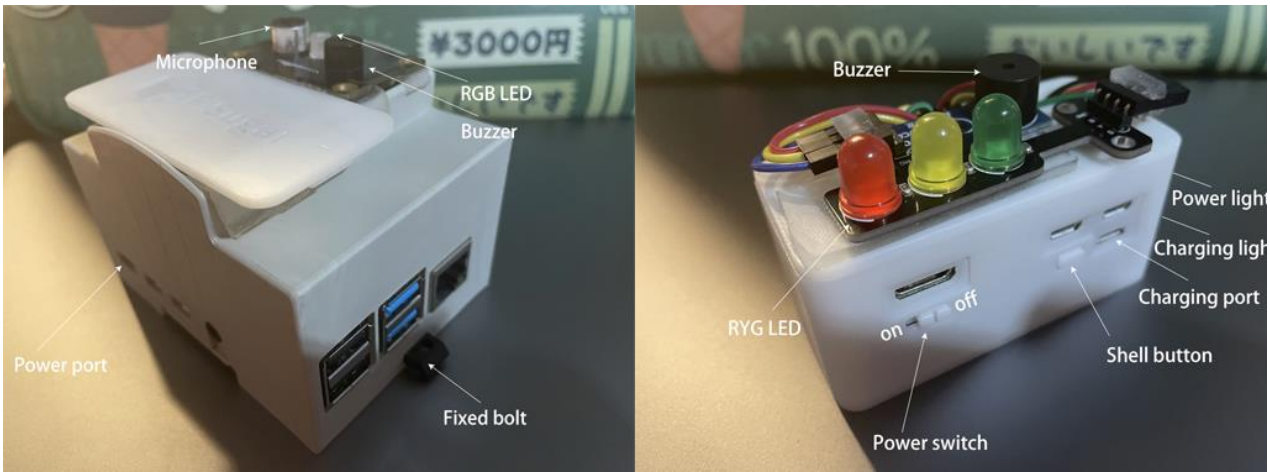

Figure 2: Prototype

2.1 Cautions & Warnings

We have minimized the difficulty of the user's operation, so if we encounter an unsolvable problem, rebooting the two machines can solve all the problems.

For engineers who want to modify and optimize on the basis of our products, you need to have a basic understanding of Linux system [1] and python language.

3 Getting started

3.1 Set-up Considerations

Here is the diagram of our design, it will explain the basic and draft constructure of our design.

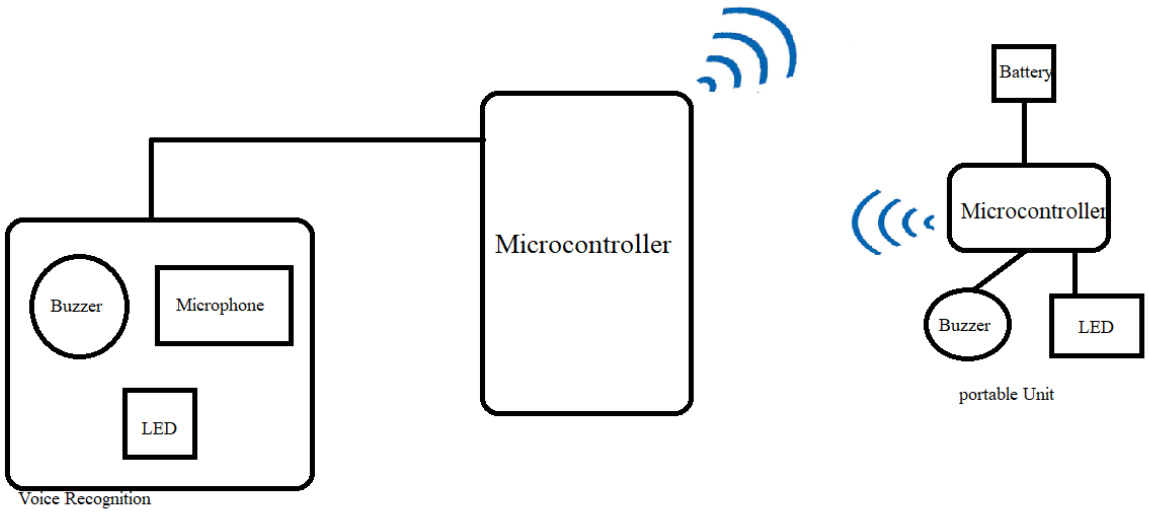

Figure 3: Block Diagram

3.2 User Access Considerations

Our target customers are the elderly and people who need help at night, especially those with physical disorders. They can manipulate the call bell by voice. Therefore, except for those who are completely unable to speak, you can use this product.

3.3 Accessing the System

In order to facilitate the use of users, we have simplified the process of opening and initializing the device to the greatest extent. All you have to do is turn on the switches of the two devices (in no order), and then wait about 40 seconds for the device to be fully initialized. Here on the left is the power button of the bell unit, you could only click it once to turn on the device. And what in the middle is the power switch of our portable unit, you could switch it to the left to turn on and switch it to right for turning off. And the button on the right hand is called shell button which is for re-initialize the device and manually connect Bluetooth.

Figure 4: Power switch and shell button

The two devices are automatically connected to Bluetooth after initialization. At this point, the user can try to say the keyword trigger to verify if the devices are working properly. Normally, bell unit's led should turn red, and after a few seconds, portable unit's led will change from green to red until the user presses the button.

If the keyword is recognized (bell unit turns red) but the portable unit does not respond for many attempts, it may be that the two devices have lost their Bluetooth pairing for some reason. You can match and connect manually by long pressing the button on the shell. Don't forget to restart both machines after manual pairing and connection.

If the device still doesn't work after the above steps, you can go directly into the raspberry pi and debug it. For specific operations for going into the raspberry pi system, you can refer to its official website [2]. The passwords of two motherboards are both "fanzizheng666".

3.4 System Organization & Navigation

For normal user, you could easily use our products by only interacting with the two devices. However, if you want to know how it works or improve it, maybe you have to go into the system of raspberry pi [3].

```
pi@rasp4b:~ $ ls -a
.                .cache               .gnupg       net_test.py  Templates     .wget-hsts
..               client_test.py       hello.py     Pictures     .thumbnails   .Xauthority
.bash_history    .client_test.py.swp  .local       .pki         Videos        .xsession-errors
.bash_logout     .config              .minecraft   .pp_backup   .vim          .xsession-errors.old
.bashrc          .dbus                mu_code      .presage     vim
bluedot_server.py Desktop             Music        .profile     .viminfo
bluetooth_test.py Documents           .mypy_cache  Public       .vimrc
Bookshelf        Downloads            Ncb          regularused  .vnc
```

Figure 5: Root dictionary of bell unit

Codes of Bell Unit	Path
Voice Recognition Code	/home/pi/Ncb/circle_asr.py
Bluetooth Client Code	/home/pi/client_test.py

Table 3: Paths of source code of bell unit

```
pi@rasp0w:~ $ ls -a
.                  button_buzzerstop.py  .gnupg        Pictures         Videos
..                 button_shell.py       hellow.py     .profile         .vnc
.bash_history      .cache                .local        Public           .Xauthority
.bash_logout       .config               MagPi         .python_history  .xsession-errors
.bashrc            Desktop               Music         reunion.py       .xsession-errors.old
bluedot_client0.py Documents             Ncbw          server_test.py
bluetooth_test0.py Downloads             net_test0.py  Templates
```

Figure 6: Root dictionary of portable unit

Code of Portable Unit	Path
Bluetooth Server Code	/home/pi/server_test.py
Buzzer Stop Code	/home/pi/ button_buzzerstop.py
Manual Connection Code	/home/pi/ reunion.py

Table 4: Paths of source code of portable unit

These two figures and two tables have shown you where you could find our python codes. If you wish to know how to deal with it, you have to learn some basic command in Linux system. And

if you want to take a look at them, just add a “vim” ahead of their path. If you want to run the codes, you could add a “python3” ahead of their path. Don’ forget that there is a space between the “vim” or “python3” with the path.

3.5 Exiting the System

If you want to exit the system, you could just tap “sudo halt” in its command, then turn off its power. Don’t forget to save your documents before halt the system.

4 Using the System

The following sub-sections provide detailed, step-by-step instructions on how to use the various functions or features of the bell unit and portable unit.

4.1 Voice Recognition Module

In order to realize the function of voice recognition, we decide to use Speech Recognition Module of YAHBOOM [4] which is a Chinese company but you actually could buy its products in North America.

It has already integrated LD3320 chip which provides a speech recognition module based on its internal MCU. We can directly enter the recognition term and the corresponding serial number of the entry through I2C, set the mode of the module (loop detection, password trigger, key trigger), and obtain the identified result, which can be used directly without knowing the internal processing. And the module is also integrated with a buzzer and a RGB lamp which we can control its color and set the switch to identify the prompt sound through I2C.

Here, we will show you how our codes function in detail.

```
#! /usr/bin/python3


import os #command line in system shell
import smbus #transfer data through data bus
```

```
import time #control time


bus = smbus.SMBus(1) #set the format of databus

i2c_addr = 0x0f   #This is the address of voice_recognition module

asr_add_word_addr  = 0x01   #address where to add keywords

asr_mode_addr  = 0x02   #address where to set recognition mode, value from 0 to 2, default=0:circle recognition

asr_rgb_addr = 0x03   #address to set RGB LED,must be 2 bits, the first bit is 1: blue 2: red 3: green, the second bit is brightness from 0 to 255

asr_rec_gain_addr  = 0x04    #address where to set sensitivity，value from 0x40 to 0x55，default=0x40

asr_clear_addr = 0x05   #address where to clear cahce, before input new keywords you must clear the cache

asr_key_flag = 0x06  #address of the button, only used in button triggering mode.

asr_voice_flag = 0x07   #address where to set if we need an alarm when voice is recognized

asr_result = 0x08  #address where to store our results

asr_buzzer = 0x09  #address to trigger the buzzer, 1: open, 0:close

asr_num_cleck = 0x0a #address where to check the input keyword
```

we import three modules. “os” is for write command in shell. “smbus” is for controlling data bus. “time” is for counting time in this code. After finishing this, we can step into the address

definition part. We refer to the official user manual and define the address of each register on the chip hardware to facilitate the next use. There are a lot of addresses that we didn't use later, but we wrote them down anyway.

```
def AsrAddWords(idnum,str):
    global i2c_addr
    global asr_add_word_addr
    words = []
    words.append(idnum)
    for  alond_word in str:
        words.append(ord(alond_word)) #convert the chip's voice-converted string into Unicode

    print(words)
    bus.write_i2c_block_data (i2c_addr,asr_add_word_addr,words)
    time.sleep(0.08)

def RGBSet(R,G,B):
    global i2c_addr
    global asr_rgb_addr
    date = []
    date.append(R)
    date.append(G)
    date.append(B)

    print(date)
    bus.write_i2c_block_data (i2c_addr,asr_rgb_addr,date)

def I2CReadByte(reg):
```

```
    global i2c_addr
    bus.write_byte (i2c_addr, reg)
    time.sleep(0.05)
    Read_result = bus.read_byte (i2c_addr)
    return Read_result
```

The first one is keyword_adding_function. It adds the entry sequence number and the keyword of the entry, this function writes the entry register address to be operated, and then write the phrase sequence number and the keyword string that identifies the phrase one byte after another where the append functions. And the ord function is to convert string format to Unicode to facilitate our comparison

The second one is to control the RGB color of LED

The last function is data_reading_function. This function firstly writes the register value to be read to the module, that is, what is read here is the detection result, so what is written is the address value of the result storage register, and then it reads the module to obtain the identified value. So that we could get a result to check if we detected the keyword. If the keyword has not been detected, the returned result will be default 255, otherwise, it will be the value you set.

```
if 0: #only set it as "1" when you input new or change keywords
    bus.write_byte_data(i2c_addr, asr_clear_addr, 0x40)#clear cache
    time.sleep(12) #it will cost at least 10s to clear the cache, so we just wait for finishing
    bus.write_byte_data(i2c_addr, asr_mode_addr, 0x00)
    time.sleep(0.1)
    #this is where you set your keywords
    AsrAddWords(1, "hey yeah hey yeah")
```

Now, it's time to go into if_check. This part is to check if there are new keywords needed to be input in registers or old keywords changed. If there are, we should set the condition to 1, otherwise, set it to 0, so that we actually could skip this part in daily use.

```
bus.write_byte_data(i2c_addr, asr_rec_gain_addr, 0x45) #set sensitivity
time.sleep(0.1)
bus.write_byte_data(i2c_addr, asr_voice_flag, 1) #set alarm
time.sleep(0.1)
RGBSet(100,100,100) #set RGB
time.sleep(2)
RGBSet(10,10,10)


while True: #this is the main loop of the code, constantly detecting and judging the voice string.
    result = I2CReadByte(asr_result)
    if(result != 255): #result != 255 means keywords are recognized
        print('triggered!')
        os.system("python3 client_test.py") #from system shell we call another py document to establish bluetooth connection
        time.sleep(1)
        RGBSet(10,10,10)
    time.sleep(0.5) #which means we detect the voice per 0.5s, this value would affect the accuracy of voice recognition because of the talking speed of speaker
```

Finally, this is the main loop which means it is the actually continuously running part of our code. While True means it will forever be running itself till the end of this world.

When the if notice the gotten result is not default 255 which means the chip just detected our keyword, it will call another py document which is used to establish Bluetooth connection and send alarm to portable unit.

4.2 Bell Unit Bluetooth Client Module

After multi-comparison, we finally choose bluedot [5] as the module for Bluetooth data transmission.

```
#! /usr/bin/python3

from bluedot.btcomm import BluetoothClient
from time import sleep
import os


def data_getit(data):
    print(data)


flag1 = True
while flag1:
    try:
        c = BluetoothClient("B8:27:EB:72:1E:32", data_getit)
        c.send("trigger")
        flag1 = False
    except:
        print("miss!")
```

As you can see, when establishing Bluetooth communication, we use a while-try loop with flag to ensure the success of data transmission.

4.3 Portable Unit Bluetooth Server Module

Here, we determine to use RPi.GPIO module to control the GPIO of raspberry because they the best compatibility. Also, Bluetooth communication is realized by bluedot.

```
#! /usr/bin/python3

import RPi.GPIO as GPIO
from bluedot.btcomm import BluetoothServer
from signal import pause
from time import sleep


GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(16, GPIO.OUT) #buzzer
GPIO.setup(18, GPIO.OUT) #LED_green
GPIO.setup(22, GPIO.OUT) #LED_Yellow
GPIO.setup(36, GPIO.OUT) #LED_red
```

This part is to initialize GPIO pins, and tell the raspberry pi which pin is to be used.

```
def data_received(data):
    if(data == "trigger"):
        print("led on buzzer on")
        GPIO.output(18, 0)
        GPIO.output(36, 1)
```

```
    GPIO.output(16, 1)
    GPIO.output(22, 0)
```

Here, we defined a function which would be run when Bluetooth server has gotten the already set trigger word “trigger”. In this function, when portable has received the trigger signal from bell unit, it will turn off green led, and have the buzzer keep beeping, the red led keep shining until the button on shell is pressed.

4.4 Shell Button Functions

Now, I think it’s the time to introduce the two functions controlled by the button on shell.

```
import RPi.GPIO as GPIO


GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(16, GPIO.OUT) #buzzer
GPIO.setup(18, GPIO.OUT) #green
GPIO.setup(22, GPIO.OUT) #yellow
GPIO.setup(36, GPIO.OUT) #red


GPIO.output(16, 0)
GPIO.output(18, 1)
GPIO.output(22, 0)
GPIO.output(36, 0)
```

This how we realize the function that the portable unit will keep making noise until someone press the button one time, just like your morning alarm clock. This mechanism can significantly improve the basic functions of alarming of our products.

As you can see, we did not set any functions here, but just called the GPIO interface of the motherboard to adjust the potential of each port.

```
#! /usr/bin/python3

import RPi.GPIO as GPIO
import os
from time import sleep


GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

GPIO.setup(22, GPIO.OUT) #yellow

GPIO.output(22, 1)
sleep(1)
GPIO.output(22, 0)
os.system("bluetoothctl")
os.system("pair DC:A6:32:F1:89:72")
os.system("connect DC:A6:32:F1:89:72")
```

This function is that when you press the button for more than a second, the machine will try to pair and connect Bluetooth again.

During the test, we found that the Bluetooth pairing may be lost after the device is not turned on or connected to each other for a period of time. So, we have added the function of manually connecting Bluetooth here.

You can see that we still choose to use the “os” method here, invoke the instructions on the command line to open the Bluetooth configuration software, named bluetoothctl, that comes with the system for Bluetooth pairing and connection.

5 Troubleshooting & Support

5.1 Error Messages or Behaviors

We have already described the errors you might face in Chapter 3.3. If you still have any problems, you could firstly go to the official website of raspberry pi for help, or search for the solution from StackOverflow [6] which is a website for answering questions of computer science.

5.2 Special Considerations

In most cases, there will be no problems with our equipment. If the device has solder joint falling off or jumper fracture caused by physical damage, you can repair it according to the device structure diagram of Chapter 6. And if it is a software problem, you can restore the software according to the source code on the Makerpo website. The paths and names of the software has been marked in Chapter 3. As for how to modify it, you need a little basic knowledge of Linux and Python.

5.3 Maintenance

Our products can be used without special maintenance. However, despite the dustproof and waterproof design, please still try to avoid dirt and water. In addition, in order to ensure the normal operation of the software, it is best to reboot the two devices after 24 hours of continuous operation.

5.4 Support

If you encounter any problems that are difficult to solve, please contact the following email:

zfan063@uottawa.ca

6 Product Documentation

6.1 BOM (Bill of Materials)

	Item	Number	Cost	Weblink
1	raspberry pi 4b	1	46.67	https://m.tb.cn/h.4PxixGI?sm=4134d5
2	AC/DC Adaptor	1		
3	SD card(16g) for 4b	1		
4	heat sink	1		
5	voice recognition chip	1	23.24	https://m.tb.cn/h.4lrLQmp?sm=d89a7b
6	plastic package for 4b	1	3.12	https://m.tb.cn/h.4P1xnlX?sm=f86b9e
7	raspberry pi zerow	1	18.74	https://m.tb.cn/h.4lrKgmj?sm=a5fb5a
8	SD card(16g) for zerow	1		
9	tricolor LED	1	1.56	https://m.tb.cn/h.4O90jHk?sm=a769c4
10	buzzer	1	0.41	https://m.tb.cn/h.4lroQ63?sm=ddd0d9
11	PiSugar rechargeable battery	1	25.19	https://m.tb.cn/h.4PxRFSo?sm=ee7021
12	PiSugar 3D printing package	1		
13	jumper wire	40	0.37	https://m.tb.cn/h.4lrpgWO?sm=d08092

	TOTAL		119	

Table 5: Bill of project

6.2 Equipment list

	Raspberry pi	**Arduino**	**OPI**
Money Cost	2	3	4
Learning Cost	4	2	3
Functions	5	5	3
Community Support	5	4	2
TOTAL	**16** >	**14** >	**12**

Table 6: Motherboard decision

From this table you could see, we have set up a criteria method for choosing the hardware we should use. Obviously, the raspberry pi gets the highest score and we chose it as our motherboard. If you would like to duplicate our design, you don't have to use the 4b model, because it has a little bit of overperformance. We recognized that the model of 3B is the most proper.

Among the many choices in the market, we chose PiSugar to provide shell and battery solutions. Because it provides the best quality products, although the price may be a little high. But it provides excellent button function in the battery module, which effectively solves a lot of our problems. You can read how to buy and use it in its official websites.

6.3 Instructions

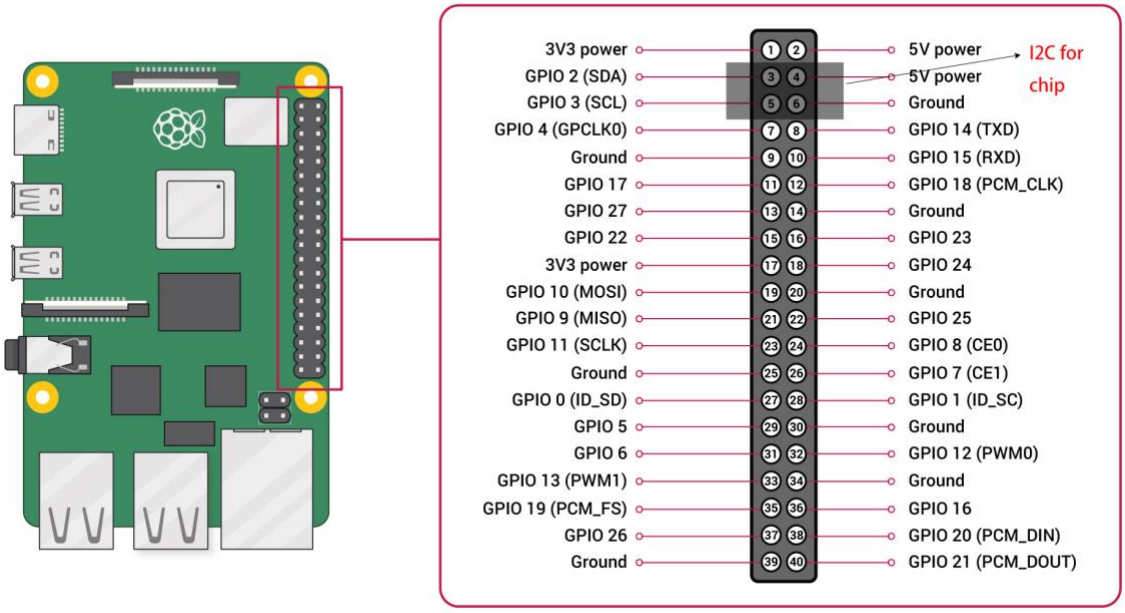

Figure 7: Raspberry Pi 4b used for the bell unit

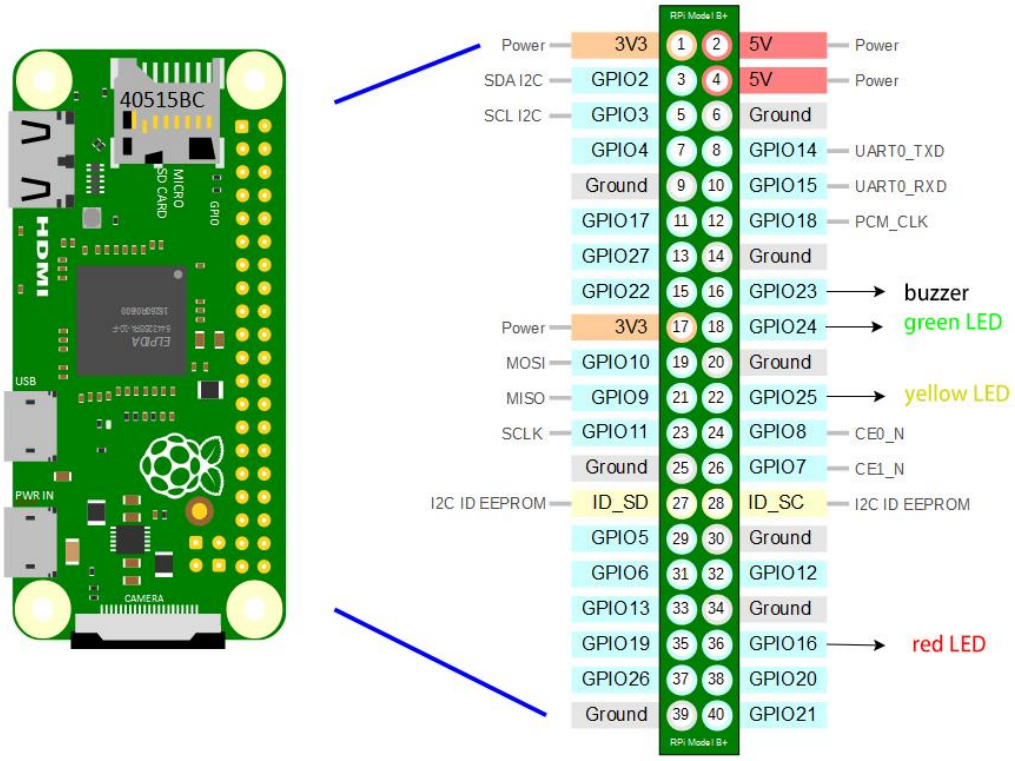

Figure 8: Raspberry Pi zerow used for the portable unit

6.4 Testing & Validation

	Conditions	**Testing results**
1	Can the signal strength be maintained within ten meters?	Yes. Even if there is a wall between and ten meters apart, the signal strength can be maintained well.
2	If one unit shuts down suddenly, will this unit automatically connect itself to another unit via Bluetooth when it is rebooted?	Yes. Raspberry pie itself has the function of Bluetooth automatic search and

		automatic connection. At the same time, we added codes to check the connection status and maintain the connection. The automatic connection after switching on and off can be guaranteed.
3	What happens when the bell unit sends a signal when the Bluetooth connection is disconnected?	The LED light of the bell unit will remain red, indicating that the signal still fails to reach the portable unit, and the bell unit will continue to search and connect with portable unit. As soon as the connection is restored, the signal will be sent out immediately.
4	How long does the battery capacity last for the portable unit?	After two tests, we could say that it can be maintained for at least 5 hours after being fully charged. And its full-charging time is about 40 minutes.

5	How is the accuracy of voice recognition of this prototype?	Nine of ten tests are successful and relatively standard pronunciation is required. We also need to make adjustments and optimizations in this respect.
6	Invite different people (another group idiom, my parents) to say key words 5 times each person and observe the success rate of speech recognition	The success rate is still maintained in a high number which is about 93.3%
7	Test of signal penetrating ability against the wall	When separated by a wall, the signal strength is still excellent. When two walls are separated, the signal strength is seriously affected. We don't expect the signal to pass through three walls, but this kind of use is also rare.

Table 7: Series of tests

7 Conclusions and Recommendations for Future Work

During this semester, the night call bell team worked very hard to make this project happen. Each deliverable allowed us to know where we were in relation to the project, the modifications, the plan established for the success of the latter. So, this project allowed us to develop skills such as troubleshooting, teamwork, respect for others. We also learned in a little time a new programming language.

This project was designed for people who have difficulty projecting their voice to ask for help. We would like to improve the product in the future so that other people who cannot use their voice can benefit from it. Adding a button might be ideal.

8 Bibliography

[1] Shotts, W. (2019). The Linux command line: a complete introduction. No Starch Press.

[2] https://www.raspberrypi.org/

[3] Upton, E., & Halfacree, G. (2014). Raspberry Pi user guide. John Wiley & Sons.

[4] https://category.yahboom.net/products/micarray?_pos=3&_sid=63a7b084a&_ss=r

[5] https://bluedot.readthedocs.io/en/latest/

[6] https://stackoverflow.com/

APPENDICES: Design Files

Here is the link of our project in MakerRepo: https://makerepo.com/ZizhengFan/882.gng5140-nightcallbell-project. You could find all the project report, source code and the demo presentation here.

Document Name	Document Location and/or URL	Issuance Date
Final Presentation.pdf	https://makerepo.com/Afurl/nica-bell-team-a2	11/19/2020
VoiceModuleUserManual.pdf	https://makerepo.com/spate214/711.gng2101a11smarttech1minpitch	11/19/2020
GNG2101 User Manual.docx.pdf	https://makerepo.com/jchen525/b12callforcare	11/19/2020

Table 8: Referenced Documents