# SI100B-Report

## Team members and Division of Work

- **Zhangzhi Xiong** (熊章智)

  Coding; Adjusting Parameters (Threshold and Model Related); Building Circuit

- **Tianni Yang** (杨天倪)

  Adjusting Parameters (Accuracy Related); Building Circuit

- **Yixuan Chen** (陈逸轩)

  Coding; Experimenting

- **Joint Work**

  Debugging; Recording ; Writing Report

## Text

### Main Content of the Project

#### Part 1: Some Preparations

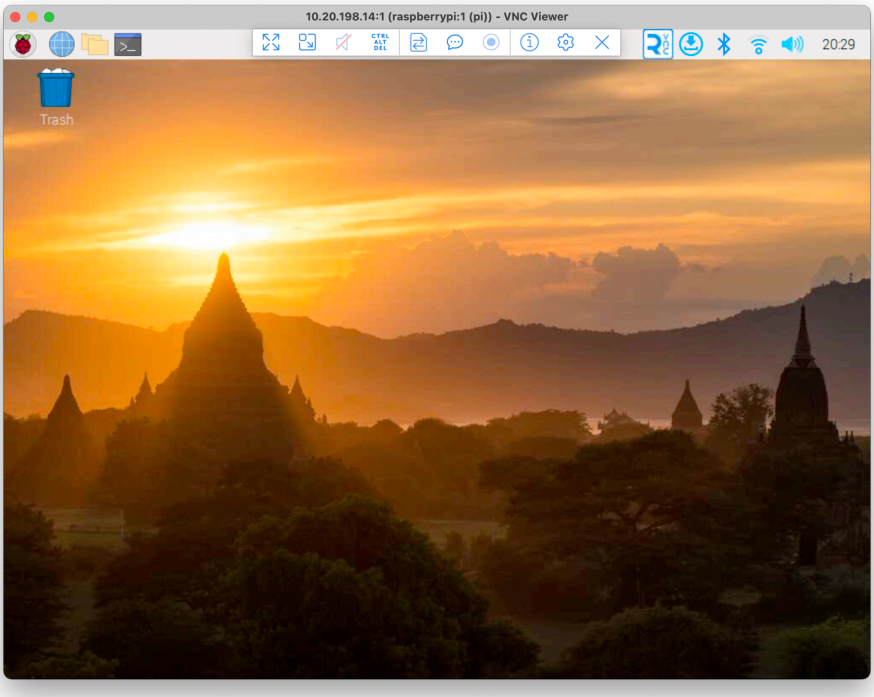
The Project "hand-written number recognition" is  realized through hardwares and program. As for the hardware, we use a Raspberry Pi 3B+  as the remote control and a Pi camera attached to it. As for the program, the language we use is Python, and we will use the KNN algorithm in OpenCV, which is a typical method in the field of Image Processing. Besides these two parts, to realize the interaction between the hardwares and the real world, we have to build a circuit with LED light and LED digital tube added, so as to better tell us the moment of taking photos and the result of the algorithm.

First of all, the most important thing is to understand how to operate the Raspberry Pi. We know that the Raspberry is a little computer with no screen or input device. As for the first stage, we will build a ***Remote Desktop*** through ***VNC***, in order to make it easy for us to write in programs and run our codes through our own computer. This may seem to be a little bit weird, but in fact, our computer only plays the role of displayer, and the running of codes is ac in the Raspberry Pi.

To realize this, we have to assign an IP to the usb interface in order to connect the Raspberry with the computer, since that this operation puts two devices into one LAN. After that, all we have to do is to open the VNC app and initiate the connection.

Through these operations, we can manipulate the Raspberry Pi through the remote desktop on our own computer.

Note that the Raspberry Pi Board and its camera is rather fragile. When assembling devices, we have to make sure the power is off. What's more, ***only after shutting down*** the board completely can we power the board off. If not so, it will do harm to the transmission of data.




## Part 2: The Establishment of the Training Set

KNN algorithm requires training data. But before that, we must know how to process our pictures. The photo taken is colorful, but this is not what we want. First, we have to change our colorful pictures into the gray one. Picture is actually a matrix, containing the RGB information of every pixel. But we don't want the RGB colors, all we want is to use a gray level value to represent the pixel. Using the algorithm in the library OpenCV, it is easy to realize this:

```
grayImg = cv2.cvtColor(src, code)
# code can be  `cv2.COLOR_BGR2GRAY`,
# `cv2.COLOR_BGR2RGB`,`cv2.COLOR_BGR2HSV`.
```

After that, the colorful picture is converted to a ***gray scale picture***. To use the converted picture to generate training data set, we have to provide samples which contain the picture of each digit and the attached label to tell the true answer.  To split the picture into samples like that, we use the ***numpy*** library to realize that. With labels attached, the data are finally set up, ready to be used in the KNN algorithm.

```
# number detected related
import cv2
import os
import numpy as np
import math
from lib import imshow
import random
# get the project path
PRJ_PATH = os.getcwd()
# OPENCV_data.npz
TRAIN_DATA_NAME = "OPENCV_data.npz"
img = cv2.imread(PRJ_PATH+"/DigitsLib/digits.png")
```

```
grayImg = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cells = [np.hsplit(row,100) for row in np.vsplit(grayImg,50)]
cells = np.array(cells)
# Training set
train = cells[:,:].reshape(-1,400).astype(np.float32)
# Testing set
test = cells[:,50:100].reshape(-1,400).astype(np.float32)
k = np.arange(10) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# Training set
train_labels = np.repeat(k, 5 * 100)[:,np.newaxis]
# Testing set
test_labels = np.repeat(k,5 * 50)[:,np.newaxis]
knn = cv2.ml.KNearest_create()
knn.train(train ,cv2.ml.ROW_SAMPLE , train_labels)
# Testing the training model
ret, result, neighbors, dist = knn.findNearest(test,k = 3)
matches = result==test_labels
correct = np.count_nonzero(matches)
accuracy = correct/result.size
print(f"{accuracy * 100: 0.02f}%")
# As for this case, when k=1, the accuracy would be 100%
# When k=3, the accuracy would be 97.56%

# Save your training model
fileName = os.path.join(PRJ_PATH, "TrainingData", TRAIN_DATA_NAME)
np.savez(fileName, train = train, train_labels = train_labels)
```

Pay special attention to the ***grammar related to Numpy*** when splitting the picture.

## Part 3: Detecting the Numbers

At first, it is abstract for us to think how can we detect and extract the number out of the paper. But after knowing how and why, we will find it quite simple.

First we know that the white part of a gray scale picture is actually determined by the gray level of pixels. But how might we know where is the number? We have to grasp the information of the gray level. But the key problem is that the range of the gray level is from 0 to 255. We can't simply tell which pixel belongs to the number, for the values of gray level vary from each to each. So we have to use a procedure to 'force' that the gray level of each pixel can be only either 0 or 255. To realize this, we have to use a threshold to decide whether the gray level of a pixel is going to be changed into 0 or 255.

Through this process, we change the gray scale picture into a ***binary*** one.


OK. Now we have a matrix, whose elements can only be 0 or 255. What to do next is quite clear. We have to scan the matrix from left to the right, detecting where the 0 changes into 255 (type 1) and where the 255 change into 0 (type 2). Not all changes of this kind should be recorded. When the change of type 1 is detected, what we seek for should be a change of type 2. The area between the columns of the indexes recorded in pair is in fact the number! Record the indexes of the column required, and this will help us split the matrix vertically. As for the rows, we do the similar things like that, so as to correctly split the matrix horizontally. First split the whole matrix vertically, then split every unit horizontally, and we will get the number matrix we want!

Note that to further increase the accuracy, we make some adjustment to increase the width of  the black margin. All these functions are written in the file ***'my_function.py'*** .

```
# Note that we make some adjustment in the code so as to increase the margin
def image_split_column(img:np.ndarray)->list:
    # find out the number of columns in the original image
    # create a list to record the number of elements with a value of 255 in each column
    column = img.shape[1]
    columnHist = np.zeros(column)
    # initialize the variables
    flag = 0
    startList = []
    endList = []
    # step1:
    # count the number of elements with a value of 255 in each column and record it in columnHist
```

```
    # record the location where the the number of 255 changes in startList and endList
    # record the status with flag
    for i in (range(column-1)):
        if 255 not in img[:,i]:
            if 255 in img[:,i+1]:
                startList.append(i)
        if 255 in img[:,i]:
            if 255 not in img[:,i+1]:
                endList.append(i+1)
    # step 2:
    # following the startList and the endList, split the digits area from the original image.
    # there maybe several areas. recorder the areas in imgList and return imgList.
    imgList = [img[:,startList[i]-15:endList[i]+15] for i in range(len(startList))]

    ret = imgList
    return ret

def image_split_row(img:np.ndarray)->list:
    # find out the number of rows in the original image
    # create a list to record the number of elements with a value of 255 in each row
    row = img.shape[0]
    rowHist = np.zeros(row)
    # initialize the variables
    flag = 0
    startList = []
    endList = []
    # step1:
    # count the number of elements with a value of 255 in each row and record it in rowHist
    # record the location where the the number of 255 changes in startList and endList
    # record the status with flag
    for i in (range(row-1)):
        if 255 not in img[i,:]:
            if 255 in img[i+1,:]:
                startList.append(i)
        if 255 in img[i,:]:
            if 255 not in img[i+1,:]:
                endList.append(i+1)
    # step 2:
    # following the startList and the endList, split the digits area from the original image.
    # there maybe several areas. recorder the areas in imgList and return imgList.

    imgList = [img[startList[i]-15:endList[i]+15,:] for i in range(len(startList))]

    ret = imgList
    return ret
```

## Part 4: Detecting Multiple Line

We have known the procedure to secure the numbers in a single line. But what about the multiple lines scenario? Actually it is quite easy. We simply split the large matrix into two parts, each containing a single line. After that, we use the function to process each line.

```
rows = image_split_row(imgBin)
for p in range(len(rows)):
    imgCol = image_split_column(rows[p])
    print(f'The followings are the {p+1}th row')
    imshow(rows[p])
    imgMonos = []
    for col in range(0,len(imgCol)):
        imgMono = image_split_row(imgCol[col])
# The following code will be shown later in the report.
```

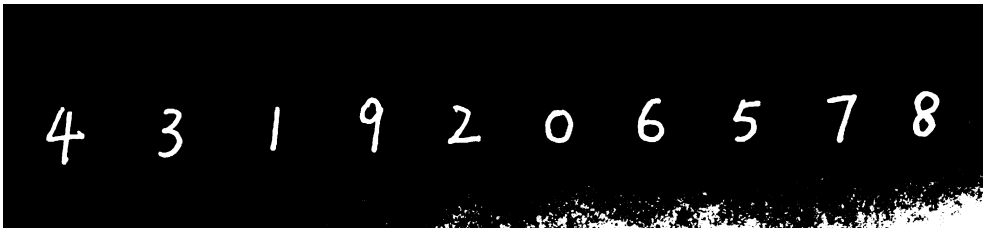
## Part 5:  Improving Accuracy

This is a harsh question: How can we improve the accuracy?  In the class, teacher gave us three hints: ***the segmentation range, threshold, and the parameter value of 'k' used in the KNN algorithm***.

As for the ***segmentation***, let's take a look back at the training data set. It's not hard to find that the samples have a black margin, which means that it is not going to be a single extraction. Yes, we have to add some black margin for the number extracted manually. Here we reach the goal through defining a ***function 'imgSqua()'***. The code is presented below.

```
# add black margin and change the matrix into a square one.
def imgSqua(img):
    (row,col) = img.shape
    m = max(row,col)
    new_matrix = np.zeros((m,m), dtype = np.uint8)
    row_start = (m-row)//2
    col_start = (m-col)//2
    new_matrix[row_start:row_start+row, col_start:col_start+col] = img
    return new_matrix
```

As for the ***threshold***, there is an important point: we have to avoid ***'noise white dots'*** (we create this term ourselves). If the threshold isn't set properly, some noise dots will turn white, bringing huge trouble to the segmentation. The situation is shown below with an example.



So we have to pick the right threshold value. But the question is how? A fixed value can't just work fine, for the reason that when it comes to the practical use, the pictures can be taken under different light environment. So we have to use a function to calculate the "appropriate" value of the threshold. Later, we found that this value is not actually the best. Normally, we have do an additional subtraction, like minus 25

or so, to reduce the noise white dots.

```
_threshold , imgBin1 = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
t = _threshold - 32
_threshold , imgBin = cv2.threshold(imgGray, t, 255, cv2.THRESH_BINARY_INV)
```

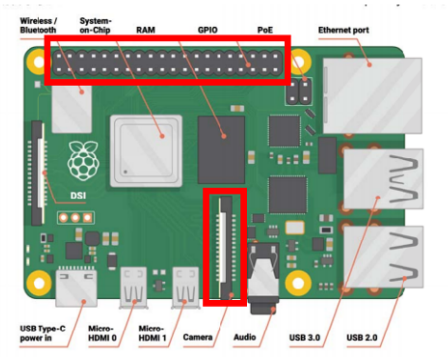
As for the 'k' value, after various experiment, we find that 3 is the most suitable one.

## Part 6: Building the Circuit

First, we have to understand how GPIO works in Raspberry Pi. We can set the GPIO port in the python code. We can choose ***the mode of IN or OUT***. After setting the GPIO port, we are free to set the GPIO port in ***high level voltage or low level voltage***, enabling the electric current to flow through the circuit.

To set and manipulate the GPIO ports through python, we have to ***import the PRI.GPIO library*** and choose a ***set mode***. The set mode you choose determine the coding of the ports you are going to use. Emphasize that the ***'5V' port is prohibited***, for the probability for damaging the whole Raspberry Pi board.

The picture of more information related to the GPIO is presented below.



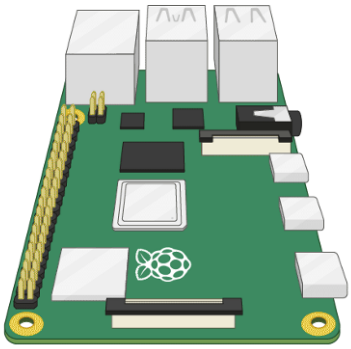

| wiringPi 编码 | BCM 编码 | 功能名 | 物理引脚 BOARD编码 | | 功能名 | BCM 编码 | wiringPi 编码 |
|---|---|---|---|---|---|---|---|
| | | 3.3V | 1 | | 5V | | |
| 8 | 2 | SDA.1 | 3 | | 5V | | |
| 9 | 3 | SCL.1 | 5 | 6 | GND | | |
| 7 | 4 | GPIO.7 | 7 | 8 | TXD | 14 | 15 |
| | | GND | 9 | 10 | RXD | 15 | 16 |
| 0 | 17 | GPIO.0 | 11 | 12 | GPIO.1 | 18 | 1 |
| 2 | 27 | GPIO.2 | 13 | 14 | GND | | |
| 3 | 22 | GPIO.3 | 15 | 16 | GPIO.4 | 23 | 4 |
| | | 3.3V | 17 | 18 | GPIO.5 | 24 | 5 |
| 12 | 10 | MOSI | 19 | 20 | GND | | |
| 13 | 9 | MISO | 21 | 22 | GPIO.6 | 25 | 6 |
| 14 | 11 | SCLK | 23 | 24 | CE0 | 8 | 10 |
| | | GND | 25 | 26 | CE1 | 7 | 11 |
| 30 | 0 | SDA.0 | 27 | 28 | SCL.0 | 1 | 31 |
| 21 | 5 | GPIO.21 | 29 | 30 | GND | | |
| 22 | 6 | GPIO.22 | 31 | 32 | GPIO.26 | 12 | 26 |
| 23 | 13 | GPIO.23 | 33 | 34 | GND | | |
| 24 | 19 | GPIO.24 | 35 | 36 | GPIO.27 | 16 | 27 |
| 25 | 26 | GPIO.25 | 37 | 38 | GPIO.28 | 20 | 28 |
| | | GND | 39 | 40 | GPIO.29 | 21 | 29 |

NXEZ.COM

## Raspberry Pi

After knowing what to do with the GPIO ports, we also have to learn more knowledge of the LED digital tube and the switch.  As for the switch, we have to know the inner structure of this device. The code of operation and the inner circuit is illustrated below.

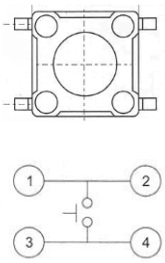
## Camera

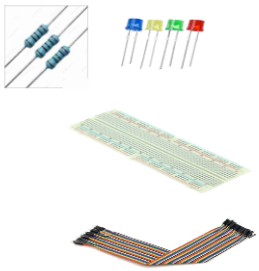


```
with PiCamera() as camera:
    # function:
    camera.start_preview()
    sleep(2)
    camera.capture(filename, use_video_port = False)
    camera.stop_preview()
    camera.close()

    setting:
    camera.brightness     # range:0~100, default:50
    camera.contrast       # range:-100:100, default:0
    camera.resolution     # set (width_32, height_16)
    camera.iso            # 100,200,320,400,500,640,800
    ......
```

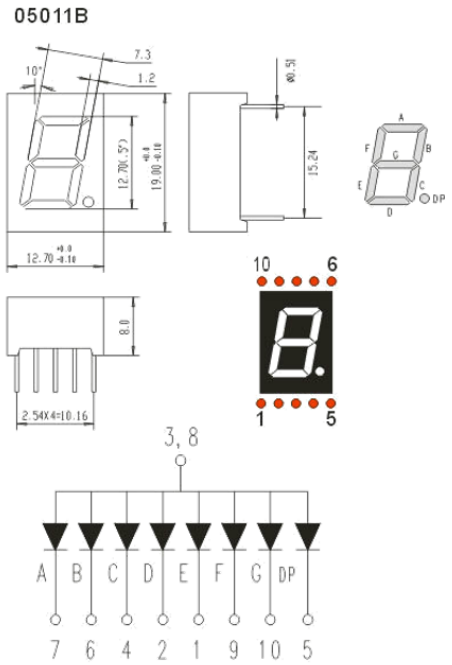
---

```
# Polling: detect the input status of GPIO in a loop

# GPIO edge detect:
# state: GPIO.RISING/GPIO.FAILING/GPIO.BOTH
GPIO.wait_for_edge(channel, state)
# event detect with callback function
GPIO.add_event_detect(channel,state)
GPIO.add_event_callback(16,callback function)
```



As for the digital tube, it's more complex in comparison. After knowing the control relationship between the LED and the port, we have to establish combinations of LEDs to represent different numbers. The relationship will be presented in the code of ***'my_function.py '*** and the circuit structure of the digital tube will be shown in the following picture.




## Digital Tube

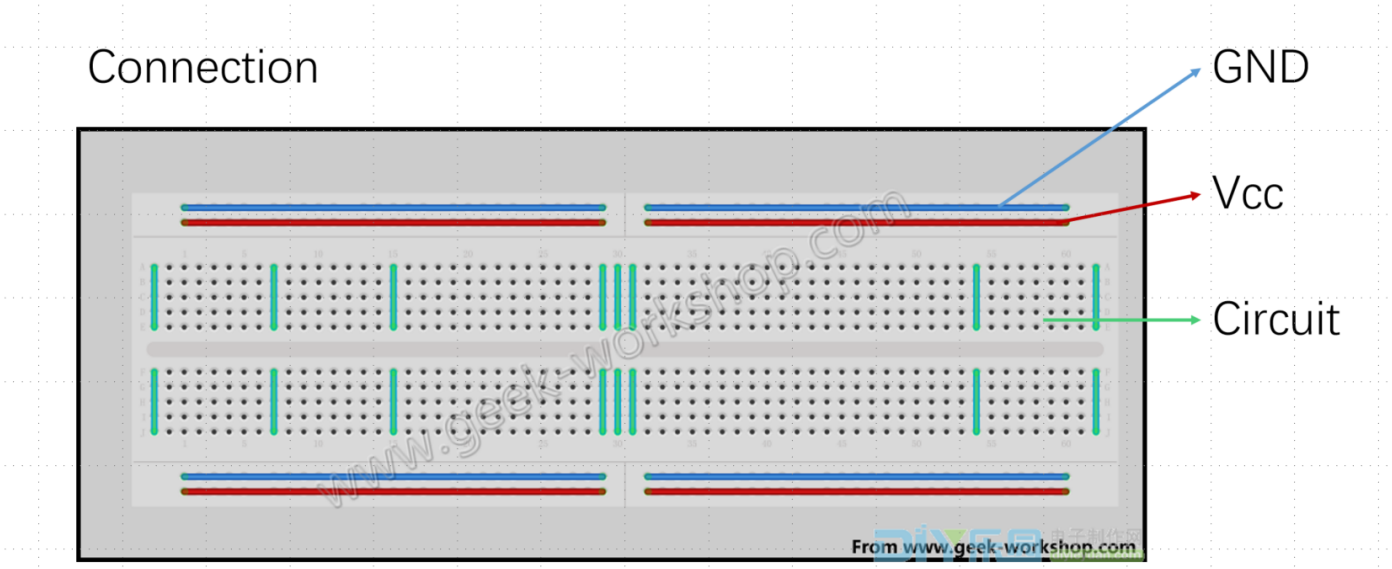
```python
def led_display(numList:list)->None:
    # step 1:
    # Clarify the relationship between led pins and GPIO pins
    # Set the GPIO pins to GPIO.OUT mode and give them the right output
    GPIO.setmode(GPIO.BOARD)
    out_para = GPIO.HIGH
    # step 2:
    # Clarify the led composition of each number
    a = 31
    b = 29
    c = 16
    d = 13
    e = 12
    f = 35
    g = 37
    dp = 18
    seg = [12,13,16,18,29,31,35,37]
    for segment in seg:
        GPIO.setup(segment,GPIO.OUT)
        GPIO.output(segment, True)
    num0 = [a,b,c,d,e,f]
    num1 = [b,c]
    num2 = [a,b,g,e,d]
    num3 = [a,b,g,c,d]
    num4 = [f,g,b,c]
    num5 = [a,f,g,c,d]
    num6 = [a,f,g,c,d,e]
    num7 = [a,b,c]
    num8 = [a,b,c,d,e,f,g]
    num9 = [a,b,c,d,f,g]
    n_list = [num0,num1,num2,num3,num4,num5,num6,num7,num8,num9]
    # step 3:
    # Display the numbers in the list one by one
    # Display every number for 1 second
    # Wait two seconds when displaying different lines
    for ele in numList:
        for dig in n_list[ele]:
            GPIO.output(dig, GPIO.LOW)
        time.sleep(1)
        for dig in n_list[ele]:
            GPIO.output(dig, GPIO.HIGH)
        time.sleep(1)
    # To present mutiple results, here wait for extra two seconds.
    time.sleep(2)
    GPIO.cleanup()
    ret = None
    return ret
```

Besides these two devices, we add an additional LED in combination with the switch and the camera to inform us the moment we take photos. The code ***'take_photo()'*** of taking photos, including the function of switch and LED illustration, is presented below:

Note that the returned result of this function is the path where the picture is saved.

```
def take_photo()->str:
    # step 1:
    #set a GPIO as an input channel for detecting
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(7,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(40,GPIO.OUT)
    # step 2:
    # create the camera obj and wait for a button to take a photo
    # recorder the saving path
    # clear the camera
    with PiCamera() as camera:
        camera.start_preview()
        while True:
            if GPIO.input(7) == 1:
                GPIO.output(40,GPIO.HIGH)
                sleep(1)
                GPIO.output(40,GPIO.LOW)
                sleep(1)
                camera.stop_preview()
                filename = 'photo.jpg'
                camera.capture(PRJ_PATH + filename)
                GPIO.cleanup()
                camera.close()
                break
    # step3:
    # return the saving path
    ret = PRJ_PATH + filename
    return ret
```

Finally, so as to build the circuit correctly, we have know how the bread board works. The illustration is presented below:




## Breadboard

We can see the GND line and the VCC line located on the top and the bottom (the real bread board used by us is a little different to the one in the picture). Each vertical line of fine holes is a current path. If we want the current to flow across each vertical line, we have to use a device to do the connection, like a switch or a resistor.

Note that resistors in the circuit are extremely important! They can protect the board from being jeopardized due to the high electric current.

## Presentation & Result of the Project

Finally, we have all the code and the circuit. Time for demonstration!

We will present the complete code and the circuit built for our project.

The ***my_function.py*** is presented below:

```
# General purpose
import os
import numpy as np
from time import sleep
import time
# GPIO related
import RPi.GPIO as GPIO
# camera related
from picamera import PiCamera, Color
# GPIO mode: GPIO.BOARD, GPIO.BCM
GPIO.setmode(GPIO.BOARD)
mode = GPIO.getmode()
# Close GPIO warning
GPIO.setwarnings(False)
# get the project path
PRJ_PATH = os.getcwd()
def image_split_column(img:np.ndarray)->list:
    # find out the number of columns in the original image
    # create a list to record the number of elements with a value of 255 in each column
    column = img.shape[1]
    columnHist = np.zeros(column)
    # initialize the variables
    flag = 0
    startList = []
    endList = []
    # step1:
    # count the number of elements with a value of 255 in each column and record it in columnHist
    # record the location where the the number of 255 changes in startList and endList
    # record the status with flag
    for i in (range(column-1)):
        if 255 not in img[:,i]:
            if 255 in img[:,i+1]:
                startList.append(i)
```

```
        if 255 in img[:,i]:
            if 255 not in img[:,i+1]:
                endList.append(i+1)
    # step 2:
    # following the startList and the endList, split the digits area from the original image.
    # there maybe several areas. recorder the areas in imgList and return imgList.
    imgList = [img[:,startList[i]-15:endList[i]+15] for i in range(len(startList))]

    ret = imgList
    return ret
def image_split_row(img:np.ndarray)->list:
    # find out the number of rows in the original image
    # create a list to record the number of elements with a value of 255 in each row
    row = img.shape[0]
    rowHist = np.zeros(row)
    # initialize the variables
    flag = 0
    startList = []
    endList = []
    # step1:
    # count the number of elements with a value of 255 in each row and record it in rowHist
    # record the location where the the number of 255 changes in startList and endList
    # record the status with flag
    for i in (range(row-1)):
        if 255 not in img[i,:]:
            if 255 in img[i+1,:]:
                startList.append(i)
        if 255 in img[i,:]:
            if 255 not in img[i+1,:]:
                endList.append(i+1)
    # step 2:
    # following the startList and the endList, split the digits area from the original image.
    # there maybe several areas. recorder the areas in imgList and return imgList.

    imgList = [img[startList[i]-15:endList[i]+15,:] for i in range(len(startList))]

    ret = imgList
    return ret
def led_display(numList:list)->None:
    # step 1:
    # Clarify the relationship between led pins and GPIO pins
    # Set the GPIO pins to GPIO.OUT mode and give them the right output
    GPIO.setmode(GPIO.BOARD)
    out_para = GPIO.HIGH
    # step 2:
    # Clarify the led composition of each number
    a = 31
    b = 29
```

```
    c = 16
    d = 13
    e = 12
    f = 35
    g = 37
    dp = 18
    seg = [12,13,16,18,29,31,35,37]
    for segment in seg:
        GPIO.setup(segment,GPIO.OUT)
        GPIO.output(segment, True)
    num0 = [a,b,c,d,e,f]
    num1 = [b,c]
    num2 = [a,b,g,e,d]
    num3 = [a,b,g,c,d]
    num4 = [f,g,b,c]
    num5 = [a,f,g,c,d]
    num6 = [a,f,g,c,d,e]
    num7 = [a,b,c]
    num8 = [a,b,c,d,e,f,g]
    num9 = [a,b,c,d,f,g]
    n_list = [num0,num1,num2,num3,num4,num5,num6,num7,num8,num9]
    # step 3:
    # Display the numbers in the list one by one
    # Display every number for 1 second
    # Wait two seconds when displaying different lines
    for ele in numList:
        for dig in n_list[ele]:
            GPIO.output(dig, GPIO.LOW)
        time.sleep(1)
        for dig in n_list[ele]:
            GPIO.output(dig, GPIO.HIGH)
        time.sleep(1)
    GPIO.cleanup()
    ret = None
    return ret
def take_photo()->str:
    # step 1:
    #set a GPIO as an input channel for detecting
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(7,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(40,GPIO.OUT)
    # step 2:
    # create the camera obj and wait for a button to take a photo
    # recorder the saving path
    # clear the camera
    with PiCamera() as camera:
        camera.start_preview()
        while True:
            if GPIO.input(7) == 1:
                GPIO.output(40,GPIO.HIGH)
                sleep(1)
                GPIO.output(40,GPIO.LOW)
```

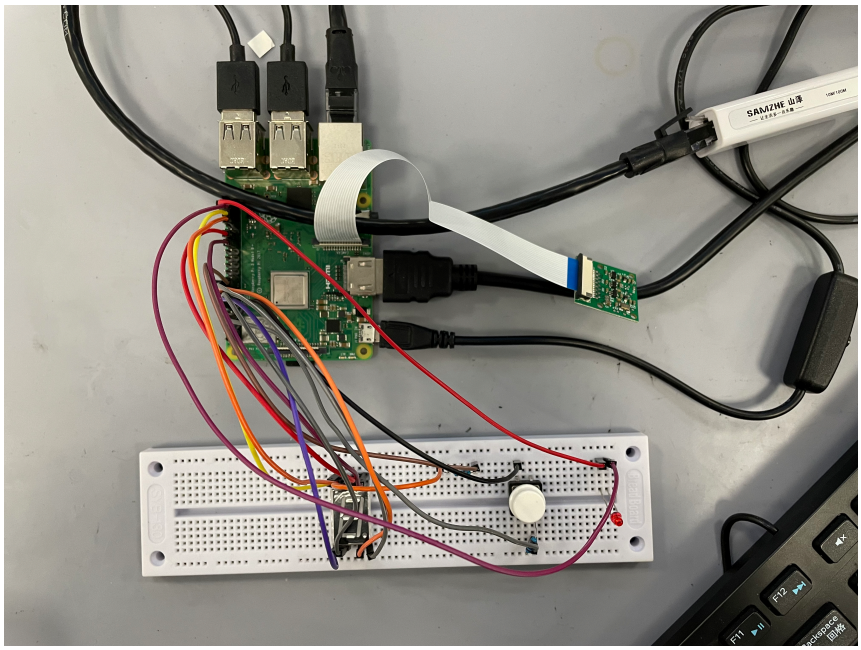
```
                sleep(1)
                camera.stop_preview()
                filename = 'photo.jpg'
                camera.capture(PRJ_PATH + filename)
                GPIO.cleanup()
                camera.close()
                break
    # step3:
    # return the saving path
    ret = PRJ_PATH + filename
    return ret
```

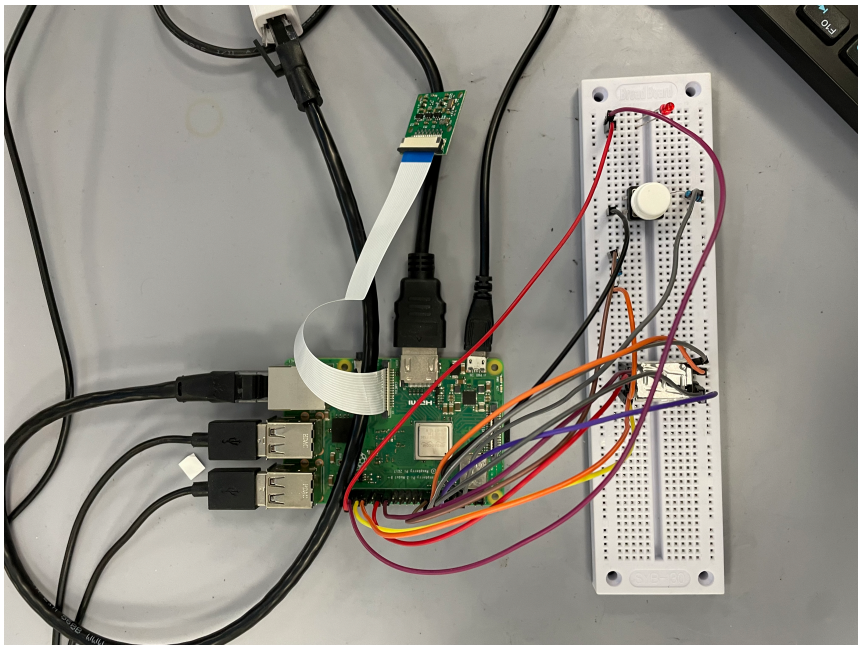
The ***ultimate_code.ipynb*** is presented below:

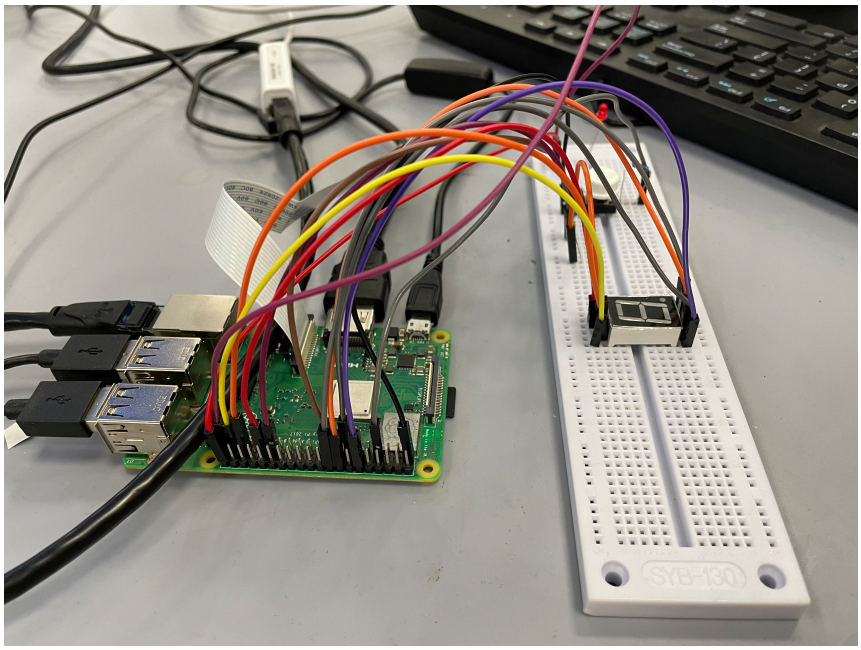
```
%load_ext autoreload
%autoreload 1
# number detected related
import cv2
import os
import numpy as np
import math
from lib import imshow
import random

# get the project path
PRJ_PATH = os.getcwd()
# OPENCV_data.npz
TRAIN_DATA_NAME = "OPENCV_data_Beta.npz"

%aimport my_function
from my_function import image_split_row, image_split_column, led_display, take_photo
# load the knn training data
# load the knn training data
with np.load(PRJ_PATH + '/TrainingData/' + TRAIN_DATA_NAME) as data:
    train = data["train"]
    train_labels = data["train_labels"]
train = train.astype(np.float32)
train_labels = train_labels.astype(np.float32)

# create KNN obj
knn = cv2.ml.KNearest_create()
knn.train(train,cv2.ml.ROW_SAMPLE,train_labels)

image = take_photo()
img = cv2.imread(image)
imshow(img)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imshow(imgGray)
_threshold , imgBin1 = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
t = _threshold - 32
```

```
_threshold , imgBin = cv2.threshold(imgGray, t, 255, cv2.THRESH_BINARY_INV)
imshow(imgBin)

def imgSqua(img):
    (row,col) = img.shape
    m = max(row,col)
    new_matrix = np.zeros((m,m), dtype = np.uint8)
    row_start = (m-row)//2
    col_start = (m-col)//2
    new_matrix[row_start:row_start+row, col_start:col_start+col] = img
    return new_matrix


imgCol = my.image_split_column_new(imgBin)
imgMonos = []
for col in range(0,len(imgCol)):
    imgMono = my.image_split_row_new(imgCol[col])
    print(f"{(col)}:")
    imshow(imgMono[0])
    imshow(imgSqua(imgMono[0]))
    imgMonos.append(imgSqua(imgMono[0]))

resizeSize = (20 , 20)
reShapeSize = (1, 400)
# resize and reshape the image with single number
# then recognize the number with knn.findNearest(imgReshape,k=?)
numberList = []
for i in range(0,len(imgMonos)):
    imgResize = cv2.resize(imgMonos[i], resizeSize,interpolation=cv2.INTER_AREA)
    _,imgResize = cv2.threshold(imgResize,0,255,cv2.THRESH_BINARY)
    imgResize = cv2.blur(imgResize, (2,2))
    _,imgResize = cv2.threshold(imgResize,127,255,cv2.THRESH_BINARY)
    imshow(imgResize)
    imgReshape = imgResize.reshape(reShapeSize).astype(np.float32)

    _,result,_,_ = knn.findNearest(imgReshape,k=3)
    numberList.append(int(result))
print("The "+str(1)+"th row has:" + str(numberList))
show_list = []
show_list.append(numberList)
# do the image splite and reshape
# then recognize the number with knn.findNearest(imgReshape,k=?)
led_display(show_list)
```

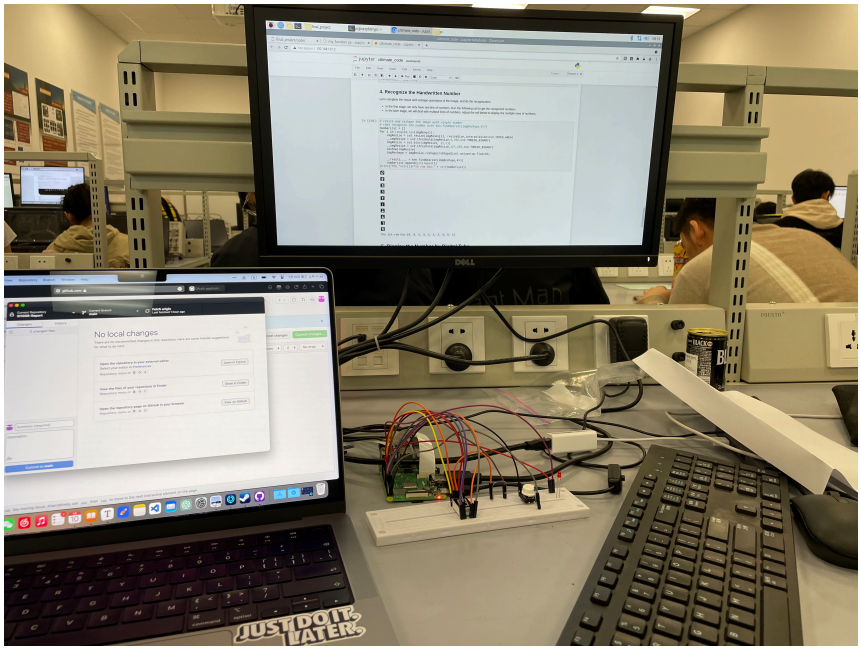
The pictures of the circuit are presented below:


SAMZHE 山泽



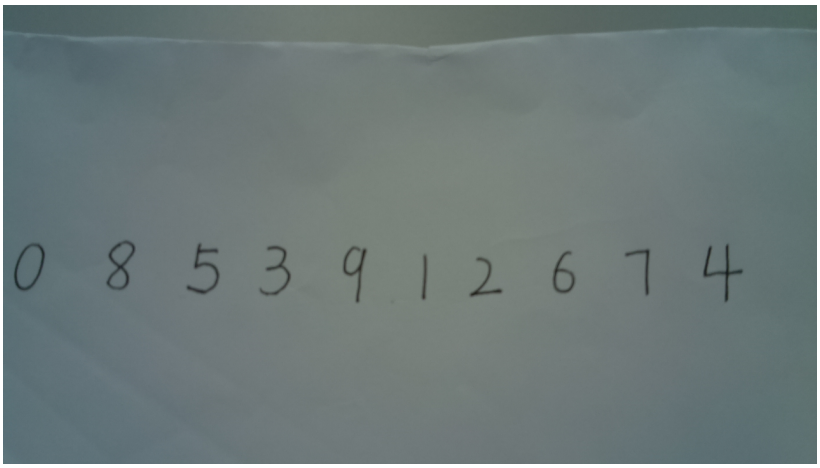


The scene of demonstration:

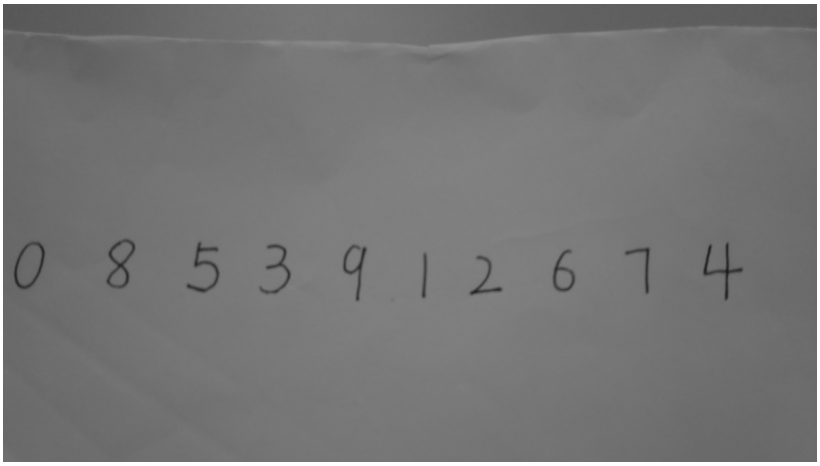



Finally, we completed our mission successfully, with the ***accuracy of 80%***, i.e, eight correct out of ten. Though during the demonstration we have to make one adjustment to the ***threshold*** due to the ***noise white dot***, in a whole, the demonstration was a total success. The accuracy was 80% and the digital tube presented the number correctly. We will show more details in the following pictures:

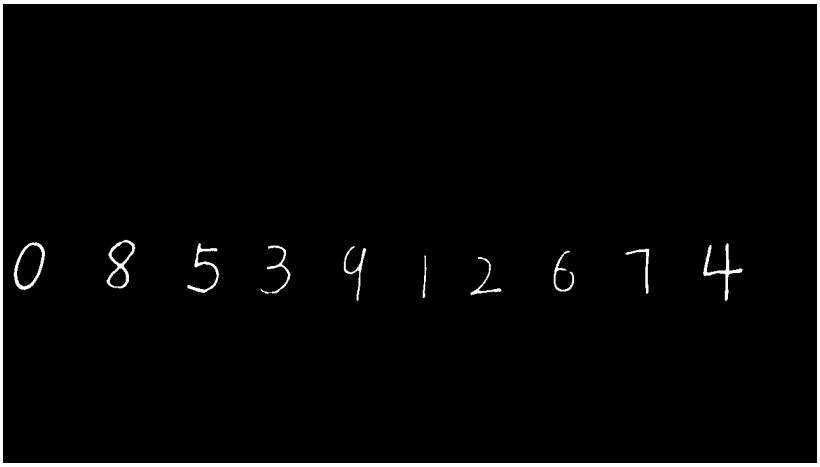
The picture taken is shown below:




The gray scale image is shown below:



The binary image is shown below:



The recognition result is shown below:

```
...
```

```
...
```

```
...
```

```
...
```

```
...
```

```
...
```

```
...
```

```
...
```

```
...
```

```
...  The 1th row has:[0, 8, 5, 3, 4, 1, 2, 6, 9, 4]
```

So far, we have finished the whole project. Well Done!

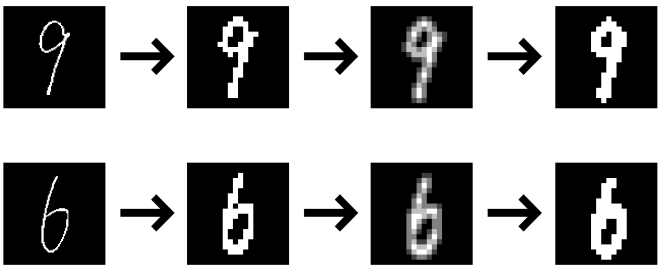
# Problems Encountered and Solutions

## Problem 1: Recognition rate

- **Description:** The initial challenge we faced was improving the recognition rate. During the first few attempts, we observed a low recognition rate of approximately **30~40%**, which fell far short of our target of around 90%.
- **Solution:** Notice that the numbers in given images are too thin to be reshaped, it leaves us to process them to match the train set. We used several methods, including cropping to retain suitable margins, adjusting binary thresholds, and applying blur filters, which are shown in the following code:

```
# Applying blur filters
imgResize = cv2.blur(imgResize, (2,2))
_,imgResize = cv2.threshold(imgResize,127,255,cv2.THRESH_BINARY)
```

This process can be shown as follows:



In addition, we make a little adjustment to the training data as well. Mentioned that the left half of the picture used for training is actually ***a little bit too illegible***, we only choose the right half as the training samples for ***avoiding the left half samples contaminating the training model***. This adjustment works well, with the increase of roughly 20% in accuracy.  That's why the training model used in the ***ultimate_code.ipynb*** is ***"OPENCV_data_Beta.npz"***.

## Problem 2: Camera initialization

- **Description:** When attempting to initialize the newly installed camera on the Raspberry Pi, it consistently presents an ***'Failed to enable connection: Out of resources'*** error. Despite trying several solutions, the issue persists.
- **Solution:** We eventually diagnosed the issue by running code `vcgencmd get_camera` in the terminal, which revealed `supported=1 detected=0`, meaning that the problem stemmed from ***poor camera connections*** which can hold back the ***initialization process***. Once we powered down and reconnected the camera, the issue was resolved.

## Problem 3: Camera preview

- **Description:** After writing the `camera.start_preview()` code, the Raspberry Pi, which should have initiated the camera preview, remained unresponsive without any error messages. Nevertheless, it executed the subsequent code without issues.
- **Solution:** Unfortunately, despite our best efforts, including ***disabling the VNCServer***, the problem persisted. Consequently, we had to rely on blind navigation during the code testing phase.

## Thoughts and Inspirations

During the program, we have gained a lot of thoughts and inspirations. Here we are going to summarize them, make some conclusions, sum up some experience and significance.

First of all, we learned how to deal with problems. That's a quite important skill, for the reason that in the future it is inevitable for us to encounter countless bugs and errors. But the key is: How can we find out the solutions to the bugs? We can ***surf the Internet***, we can ask ***ChatGPT*** for help, we can search for information on the ***online forum***, we can search for the solutions through various ways! At the same time, we have to be alert about whether the information provided on the Internet is ***valid or not***. What's more, it is also not a bad way to ***view what you've done***, like your code, again and again to ***debug***, or just to simply ***abort what you've finished and start from the beginning***. The key in common is ***patience***.

Secondly, we have a deeper understanding of the connection ***between the hardware and the program***. As a group consists of three students majoring in CS, it is not a big challenge in the coding part. But in the ***circuit and the Raspberry Pi Board part***, stagnant is frequent and sometimes it takes a long time for us to understand how this function is realized or how this circuit works. Especially when it comes to the ***GPIO*** part, we were deeply amazed by the interaction between the code and the hardware. The moment that we finally ignite the digital tube, we were much delighted, clapping hands together. Computer Science not just deals with the algorithm or programs, it also has to do with the devices, the vessels of the program.

What's more, we have a deeper understanding of the Machine Learning (I don't know if this classification is appropriate or not, but this thought is intuitive). The procedure of transforming the colorful picture into a ***binary picture***, the ***segmentation of the matrix***, and the ***reshape of the sample matrices,*** all these operations broadened our horizon and refresh our understanding of the ***Image Processing***. If we were asked: how to realize hand-written number recognition, we may have no idea. But after the project, we are quite confident to introduce the fundamental rationale of this project to others! See? This ***project's principles*** are in fact not that complex to understand!  Being aware of that, we can even generate our own idea, like the theory that the bad samples can contaminate the training model and the black margin matters a lot ( yes, we were aware of these points before teacher pointed them out in the class! ).

At last, we learn how to ***better cooperate***. Team work is rather important. The power of individual is limited, but the power of a team will be infinite if the team is organized in order. Besides, everyone's advantages and shortcomings are different, which makes it possible to the team members to learn from other's strong points to offset one's weakness, so as to further form a more ***cohesive and resilient team***. In the future, team work is inevitable in the field of scientific research, the skill to better collaborate with other teammates or organize the team in a good order will be rather beneficial. This unique experience won't be forgotten by each member of our team!

Finally, always try to ***learn new things***, and ***learn to learn new things*** fast and clear. During the project, to fix the problem encountered, we have to learn lots of new knowledge and apply these knowledge to practical use. For example, the structure and working principle of the Raspberry Pi is highly related to ***Computer System and Structure***, the circuit is related to the knowledge and the usage of the bread board and the electrical devices, even this pdf document is edited in ***Markdown***, which means that we have to learn the grammar of the markdown fast. But the joy of exploration is exactly the process of meeting new stuff and learn them. Thus, it is quite a serious point that whether we can grasp the essence of the knowledge, make some connections and put them into application or not.

In a nut shell, we have learned so much from the project, haven't we? Sum them up and absorb the wisdom, and we will gain more experience and make more steady steps on the scientific road we have embarked on.

## Credits

Hereby, we would like to ***express our sincere gratitude*** to ***the teacher and the TAs***. Without the guidance of teacher and TAs, it is impossible for us to complete the project. We really learned lots of stuff from the course, and we are grateful for the help provided by the teacher and TAs.

Thank you for your reading!

Best regards,

- @ZhangZhi Xiong xiongzhzh2023@shanghaitech.edu.cn
- @TianNi Yang yangtn2023@shanghaitech.edu.cn
- @YiXuan Chen chenyx2023@shanghaitech.edu.cn