Handwritten Number Recognition

蔡逸凡 魏奥力 朱哲霖

In this project, we use the Raspberry Pi to recognize the handwritten numbers.

We will first make a brief introduction about the main function. First of all, we build a circuit which contains a button, a digital tube and a LED light. with display device showing the live picture the camera captured. Once we press the button, the LED light flashes and a picture containing the handwritten numbers is taken. After running several computer codes, the digital tube will show the numbers in order, with each number flashing for 1 second.

The exact principle behind this function

1. Extract sample information and create the training dataset

During this part, we first import a training picture containing the numbers from 0 to 9 with 500 samples for each and change the colorful picture into a grayscale one. We then traverse the matrix and cut it into small matrices (20x20) with one number for each. By contrasting with the list [0,1,2,3….9] we successfully train the Pi to know how each number exactly looks like.

2. Recognize single line handwritten digits and explore the factors

affecting accuracy for improvement

During this, we use the sample picture with single line to test our codes. Since the picture is not exact white and black initially, we first change the matrix into all numbers of 0 or 255 with the comparation with 100, i.e., the threshold. (For we find that the black and white can be easily distinguished in pictures taken by smartphones.)

Some tries:

1.1 To find the best threshold with respect to different pictures in the later stage, we use the “OTSU” method to calculate different thresholds. It turns out to be effective in choosing the threshold. However, it may distinguish the dark shade in the middle of the picture as a number thus contain it, so we abandon it.

1.2 We turn to “adaptive thresholding” method to eliminate the shade. In comparison to global thresholding, adaptive thresholding will take light conditions in different areas on the picture into account. However, on the other hand, it will create a lot of noises, creating difficulty for recognition in later stage.

1.3 Thus, we turn back to the initial thought of a fixed threshold since the black number and the white background is relatively easy to be distinguished by an appropriate threshold.

Then we do the cutting, with the thought that when the grey level turns from 0 to 255 to 0 a number is completed, we cut the picture. (Noting: For there may be some small white noise that we cannot see with naked eyes (due to the threshold we choose), we let the computer know that small group of 255 (to be exact, <10) is noise not numberso we do not cut it, but in this way if the 1 is vertical or slanting it will be deleted as well----we made another try in the next class)

After splitting the matrix into exact one number a time, we compare it with the training data and output the value of the handwritten one.
(Noting: in this part we find that the training data didn't have an ideal result and the accuracy is floating around 70 percent, so we make the following tries:

1.1 After comparation, we find that the training data has a black frame while the testing ones don't. So, we try to add a frame artificially (to be exact, 10 rows/columns of black pixels on the four sides) to our testing one, however the result is still not 100 percent.

1.2 By observing the test data of MNIST dataset, we find that the width of the frame on the four sides is based on the barycenter (not the center) of the number. Therefore, we cut the image into two halves both vertically and horizontally, and then count the number of white pixels in each half in order to estimate the barycenter. Then we widen the frame, which means to add more columns/rows

of black pixels on the side, of the half with more white pixels. But the result turns out to be unsatisfactory, and we figure it out that it's because that the sum of white pixels does not precisely describe the barycenter of the number.

1.3 Then we change the thought and focus on the training data. We cut the training data again with the same way as we do to the testing data. To ensure that we get the grayscale (instead of binary one since our cutting functions are based on binary pictures) training dataset so as to maximize recognition accuracy, we binarize the copy of the matrix and build a starting point list and an ending point list. Then we use them to cut the original training data again and finally scale it back to a 20x20 square.

1.4 In our experiments, we find that the thickness of the number will affect accuracy. To define the thickness, we design a function to calculate the number of white pixels and divide it by the sum of pixels (both black and white). And if the 'thickness' we get is between 0.2 and 0.22, we dilate the picture by a 5x5 kernel; if it's under 0.2, we dilate it by a 10x10 kernel. (These values are acquired through experiments.) For numbers that are too thick, we erode them.

However, we didn't find an appropriate method to evaluate how much we improve by applying this function. Since if we write

numbers properly, the accuracy is already high (around 95%), it is hard to evaluate the improvement without a huge number of testing data. Meanwhile, we may accidentally dilate some noises. So, we decide not to put it in the main function.

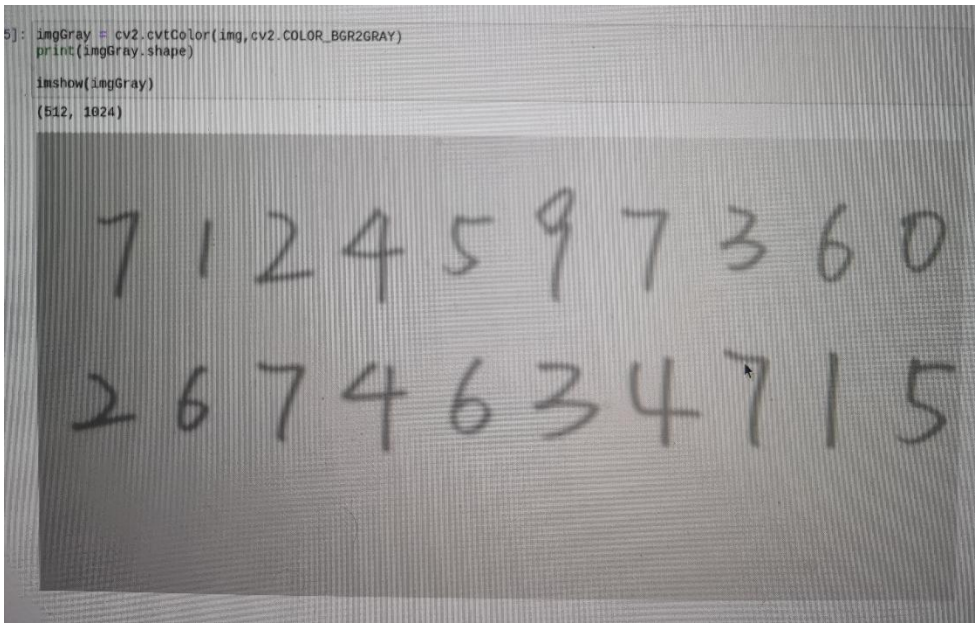
1.5 After changing the k value carefully, we finally use the value 7. (after several experiments we find that odds is better than evens) This time the result is accurate 100 percent !!!!

1.6 We change the way of distinguishing the white noise, this time we calculate the of the difference values of starting point and the ending point, deleting the too small ones.

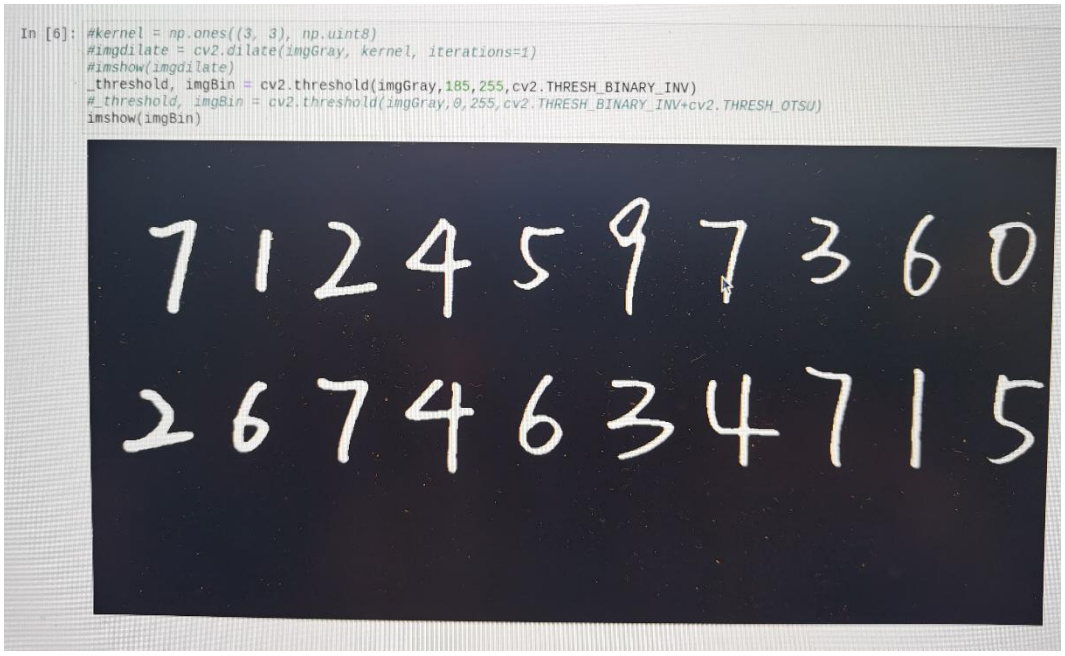
3. Recognition of multiple lines of handwritten numerals

This is broadly the same to part 2, we cut the row in the same way. and first cut the row then the column. Here are some photos：

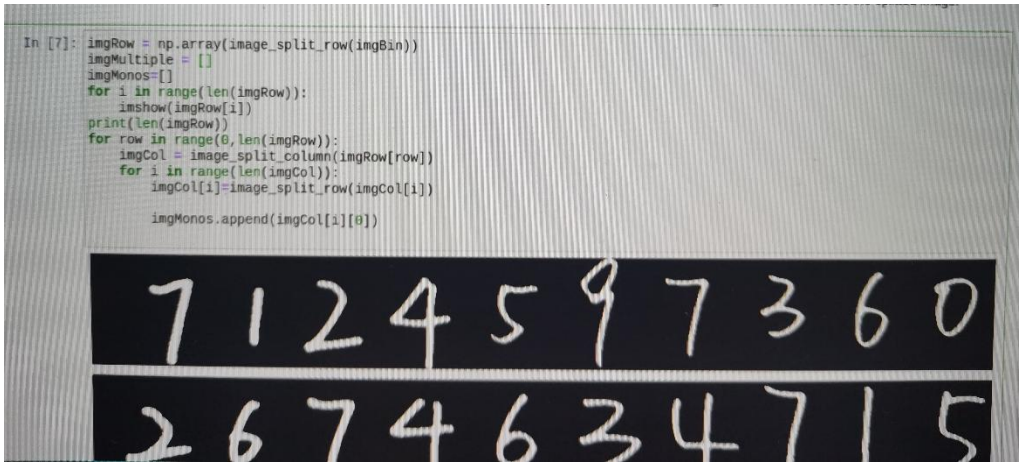
1.The grayscale one



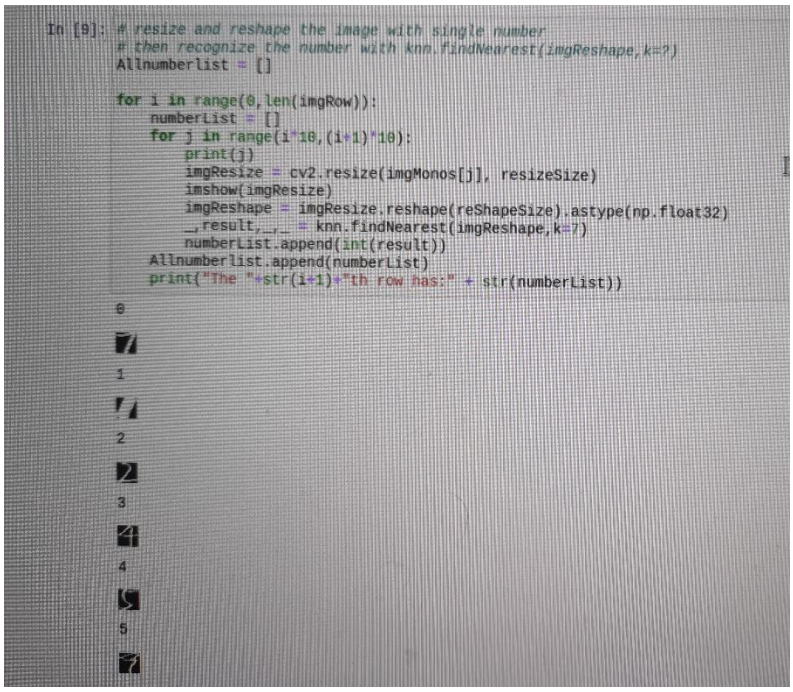
2.The Binary one



3. Splitting

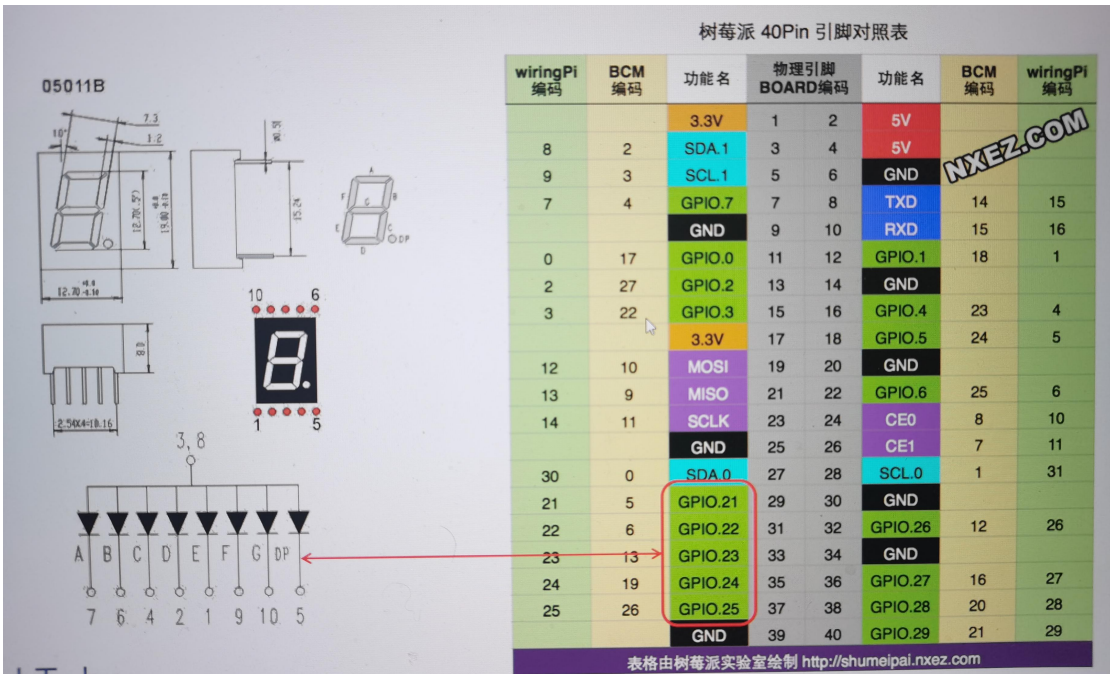


4. Recognition



4.Build a nixie tube circuit to display the results

Follow the link of the tablet



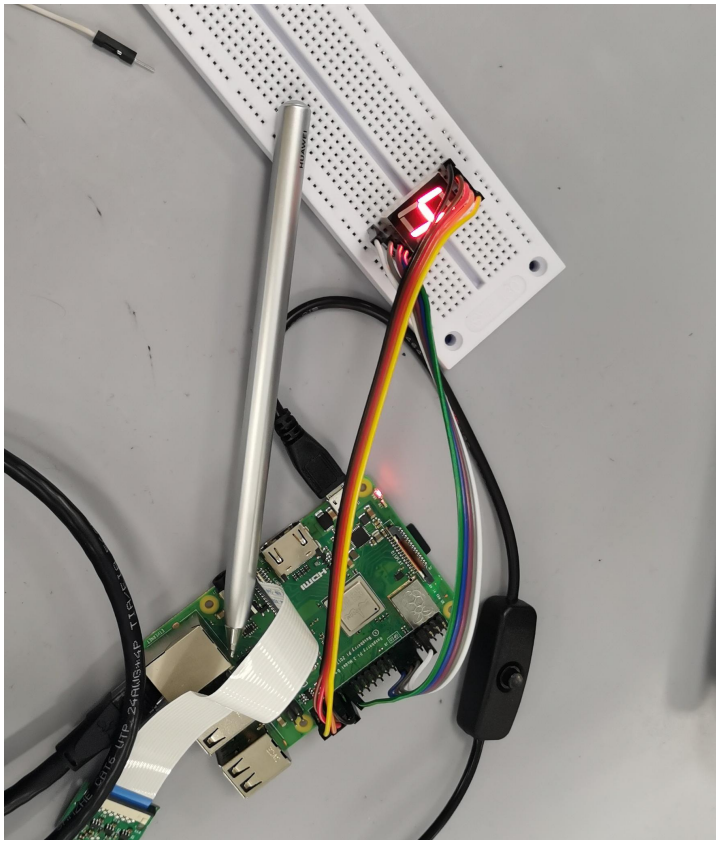
树莓派 40Pin 引脚对照表

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	2	5V		
8	2	SDA.1	3	4	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

表格由树莓派实验室绘制 http://shumeipai.nxez.com

NXEZ.COM

We build a circuit as follows

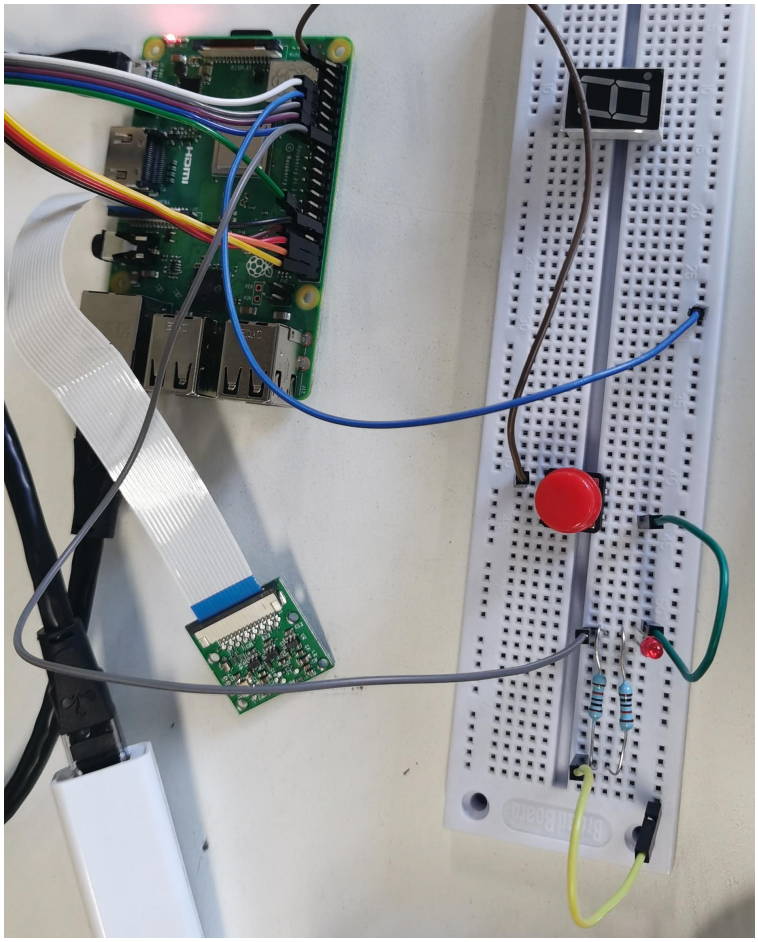


Teaching the pi to know exactly which pin connect with which part of the number (to be exact, wich small light in the digital tube) and then write a

circle to tell the pi which pins should have voltage based on the input number.as required, each number flashes and sleeps for 2 seconds, and then change to the next in order.

5. Shooting with camera

Once running the code, the screen is on and we can preview the picture that camera captured and when we press the button, the LED flashes and a picture is taken.



The result is as shown above.

Noting：

1.1.1 For trying to protect the LED we choose the 3.3v instead of 5v.

1.1.2 We use the resistor to protect the led.

1.1.3 We find that one resistor will make the voltage on the led too low that the led flash not so beatifully, so we use parallel connection with two resistors.

1.1.4 We change some parameters of the camera to ensure better quality of pictures.

Work distribution:

During this project, all of us work together so that we distribute the work evenly, breaking through the difficulties and finally come to success.