# 人工智能控

目录

**[人工智能控 1](#_Toc47)**

**[制组件包 2](#_Toc29106)**

[一、 登录树莓派操作： 2](#_Toc6616)

[二、 函数例程说明 3](#_Toc31492)

[三、 代码的执行 8](#_Toc7058)

[四、 2021人机协同比赛调试经验总结 9](#_Toc15076)

**[附件一： 12](#_Toc519)**

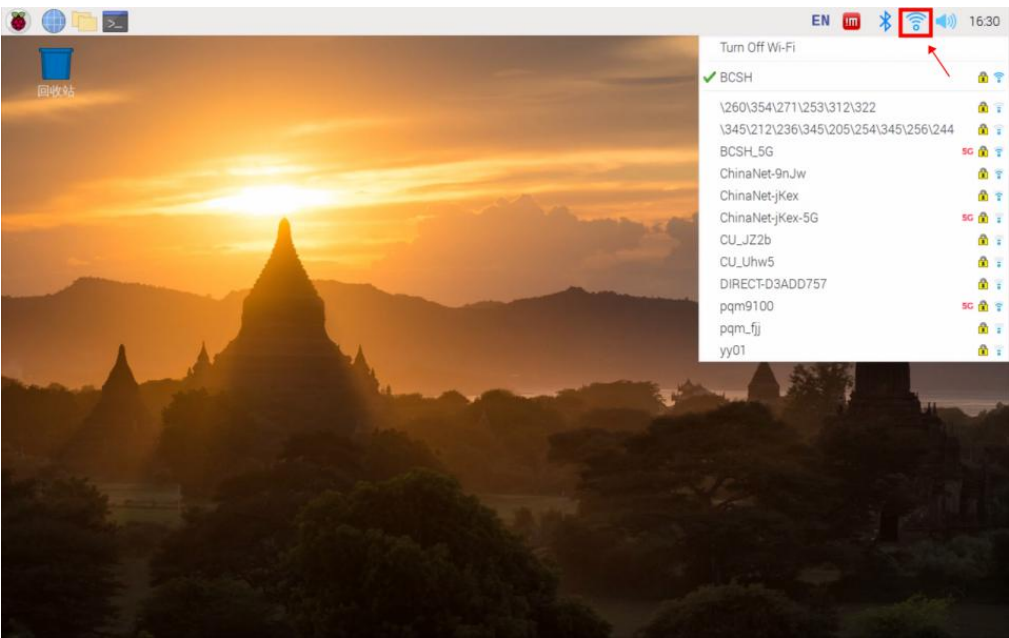
# 制组件包

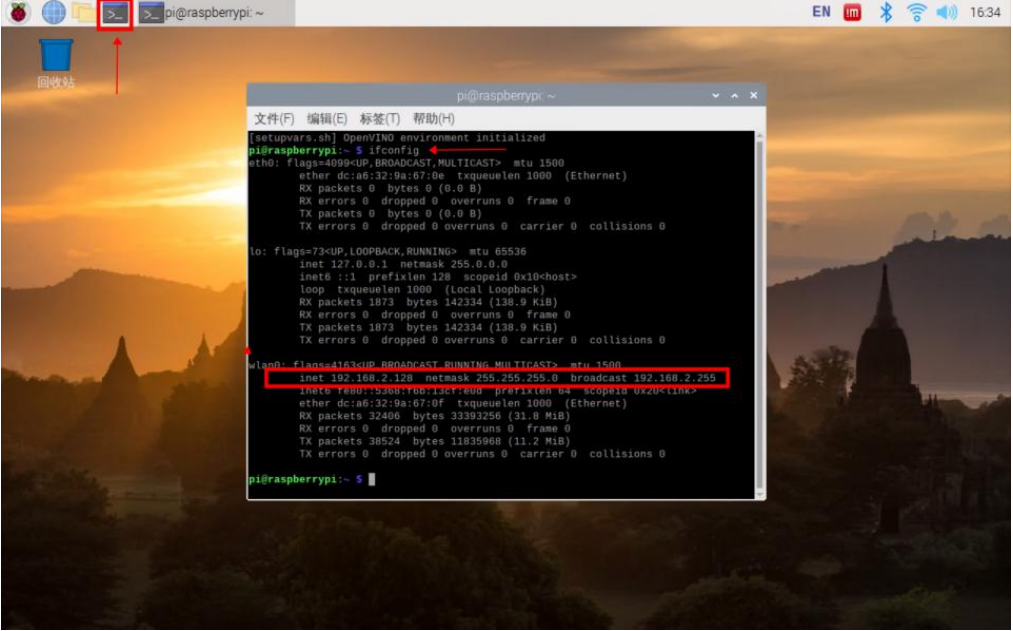
## 登录树莓派操作：

1. 有显示屏、键盘、鼠标：

鼠标、键盘插入组件包正面的USB孔，显示屏通过HDMI线小端链接树莓派，大端连接显示屏。

连接成功后，可先连接无线网。

并通过命令行窗口，输入ipconfig查看树莓派所分配的IP地址。



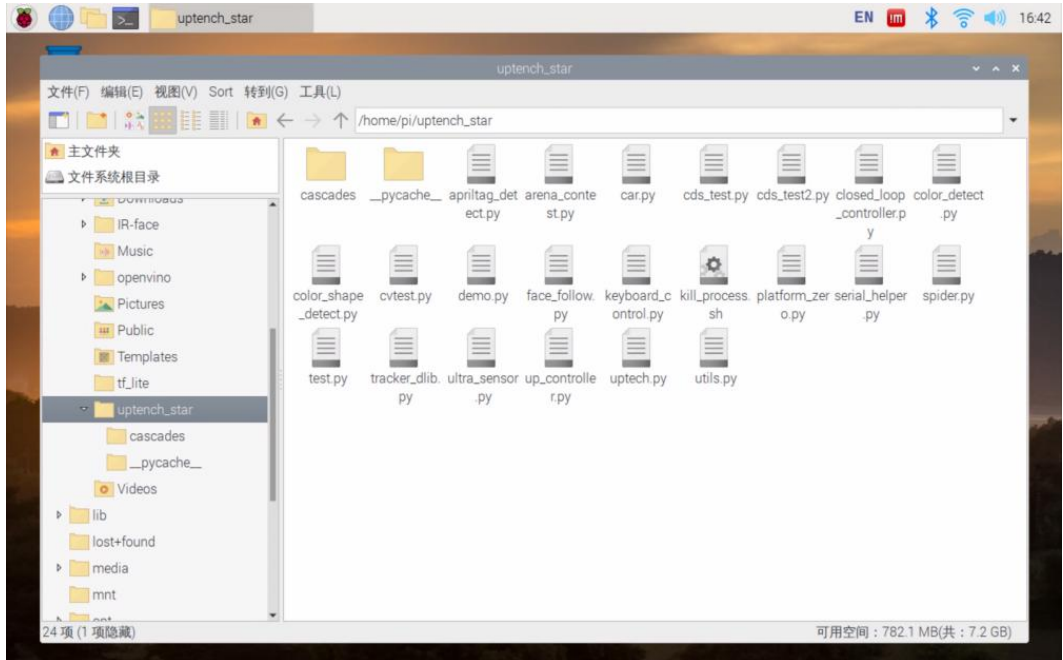
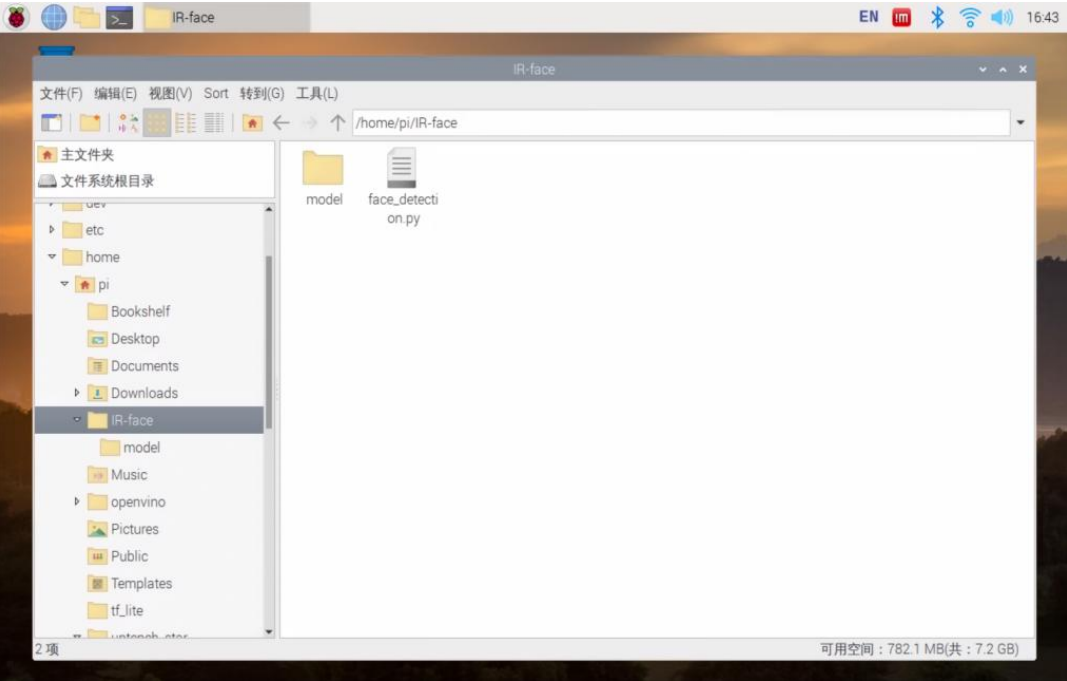
此操作目的在于实现后续的远程操控界面。

完成后，其以下载好nomachine平台，在PC上下载好nomachine并在同一无线网上输入用户名：pi 密码：bcsh ，IP地址（192.168.1.X），连接好后便可远程操控树莓派。

1. 无显示屏时，通过读卡器进入镜像文件夹，通过新建ssh文件，和配置WiFi文件，也可以实现连接无线网，通过advanced IP 软件搜索IP地址，电脑与树莓派连上同一WiFi后，通过VNC 或NOMACHINE进行远程控制。相关说明见附件一。

## 函数例程说明

1、提供的 demo 所在路径为/home/pi/uptech\_star 和/home/pi/IR-face。



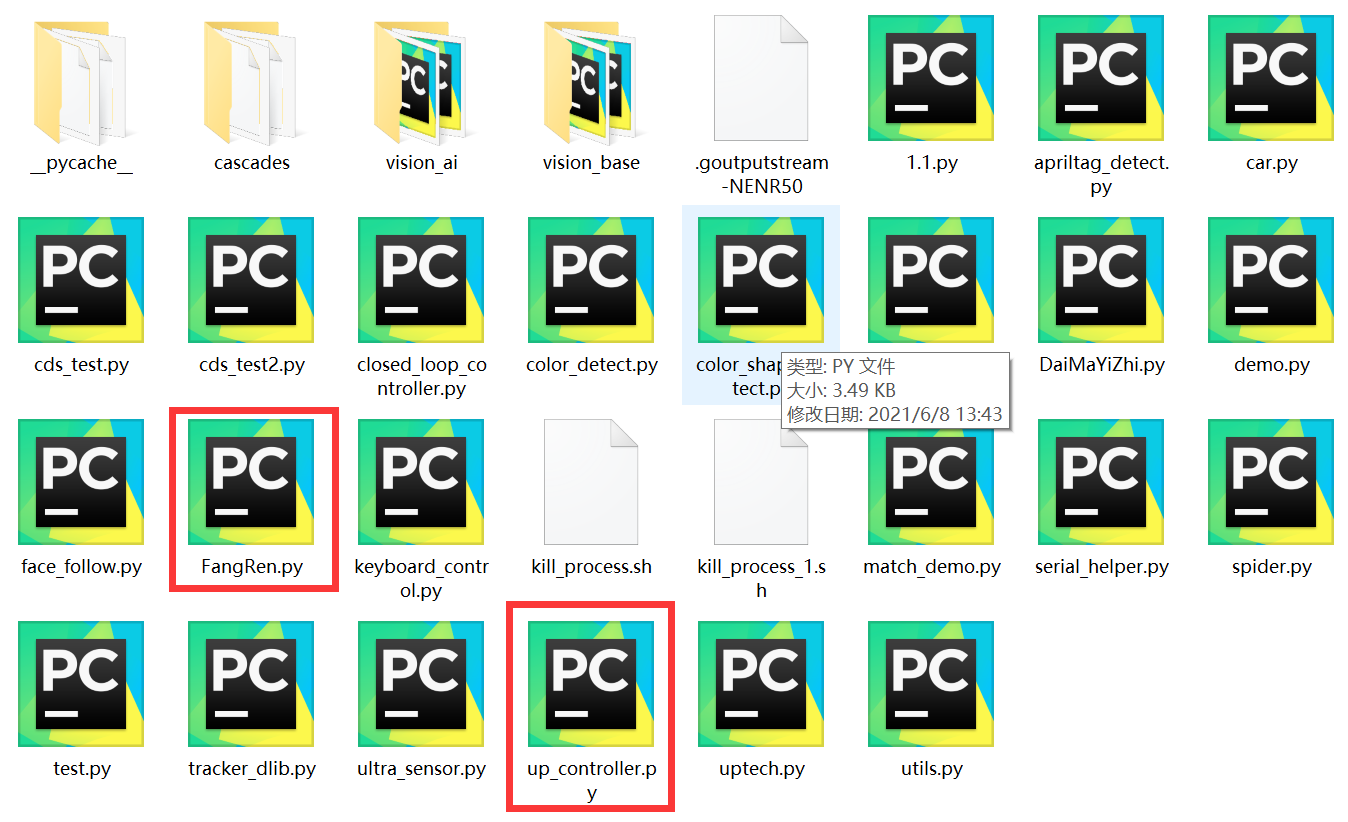
**/home/pi/uptech\_star 目录下的 demo 说明：**

* apriltag\_detect.py apriltag 识别，返回 apriltag 二维码的内容。
* arena\_contest.py 擂台边缘检测，当识别到 apriltag 二维码位置，将二维码推下擂台；
* car.py 避障小车
* cds\_test.py 舵机角度控制
* cds\_test2.py 红外光电控制舵机
* closed\_loop\_controller.py 闭环驱动控制 demo
* color\_detect.py 颜色检测、跟踪
* color\_shape\_dectect.py 单一颜色物体形状提取圆心坐标
* cvtest.py 摄像头测试
* demo.py cpu、ram、温度、电压显示
* face\_follow.py 人脸检测
* keyboard\_control.py 键盘控制小车运动
* kill\_process.sh 结束所有进程
* serial\_helper.py 串口通讯
* spider.py 四足机器人
* test.py 终端打印 9 路 AD，8 路 IO 数值；LCD 屏显示电压和 imu 信息；LED 一红一绿
* tracker\_dlib.py 目标跟踪
* ultra\_sensor.py 超声波传感器
* up\_controller.py 舵机运动库
* uptech.py Lcd led io adc 等定义
* utils.py 超声波协议库

**/home/pi/uptech\_star/vision\_ai 目录下的 demo 说明：**

* face\_detection\_ssd.py 人脸追踪（必须接神经计算棒）

1. 人机协防的代码



主要在FanRen.py当中，结合up\_controller.py的头文件编写代码。

**函数说明：**

* servo\_ids = [3, 4, 5, 6, 7, 8, 11, 12]

self.controller.set\_cds\_mode(servo\_ids, 0)--------为1时为电机模式

设置该舵机号并设置其为舵机模式。

* self.controller.up.CDS\_SetAngle(3, 512, 512)

舵机模式的控制输出，第一个数为ID号，第二个为角度，第三个为转动速度。

* self.controller.move\_cmd(300, -300)

电机模式的控制输出，在头文件中已经默认设置好了ID号为1,2。

第一个数控制左边ID号为1的速度，第二个数控制右边ID号为2的速度。

正值为正转，负值为反转（具体情况根据连接情况调节即可）

* time.sleep(0.01)

延时函数，括号内单位为秒s。

* ad6 = self.controller.adc\_data[6]

ad口数值的采集

* io\_1 = self.controller.io\_data[1]

io 口数值的读取

* def test\_servo(self)：

…

自定义函数，类似于C语言中的子函数。

* class MatchFangren:

def \_\_init\_\_(self):

self.version = "v1.0"

self.controller = UpController()

self.controller.lcd\_display("MatchFangren")

servo\_ids = [3, 4, 5, 6, 7, 8, 11, 12]

self.controller.set\_cds\_mode(servo\_ids, 0)

self.k = 0

self.r = 0

self.m = 0

self.nh = 0

self.nq = 0

self.apriltag\_state = 0

初始化函数内定义添加各类全局变量。------self.X

* if \_\_name\_\_ == '\_\_main\_\_':

match = MatchFangren()-------联系初始化函数

match.start\_match()

主函数说明，执行函数代码。

* def edge\_detect\_thread(self):

while True:

self.adc\_data = self.up.ADC\_Get\_All\_Channle()

io\_all\_input = self.up.ADC\_IO\_GetAllInputLevel()

print("io\_vaule = {}".format(io\_all\_input))

io\_array = '{:08b}'.format(io\_all\_input)

self.io\_data.clear()

for index, value in enumerate(io\_array):

io\_value = (int)(value)

self.io\_data.insert(0, io\_value)

# print(self.io\_data)

up\_controller.py文件中封装的实际读取代码。

* def move\_cmd(self, left\_speed, right\_speed):

self.up.CDS\_SetSpeed(1, left\_speed)

self.up.CDS\_SetSpeed(2, -right\_speed)

速度指令，自由控制-开环控制器

* print("ad3={},ad6={},ad7={}".format(self.controller.adc\_data[3],self.controller.adc\_data[6],self.controller.adc\_data[7]))

打印输出函数，可用于检测代码的运行情况。

* up=uptech.UpTech()

up.LCD\_Open(2)

LCD初始化打开函数

* up=uptech.UpTech()

up.MPU6500\_Open()

6050初始化打开函数

* attitude=up.MPU6500\_GetAttitude() #读取6050三轴数值

str\_attitude\_pitch='Pitch:%.2f ' % attitude[0]

str\_attitude\_roll='Roll :%.2f ' % attitude[1]

str\_attitude\_yaw='Yaw :%.2f ' % attitude[2]

up.LCD\_PutString(0,30,str\_attitude\_pitch) #LCD显示函数

up.LCD\_PutString(0,44,str\_attitude\_roll)

up.LCD\_PutString(0,58,str\_attitude\_yaw)

up.LCD\_Refresh() #更新显示函数

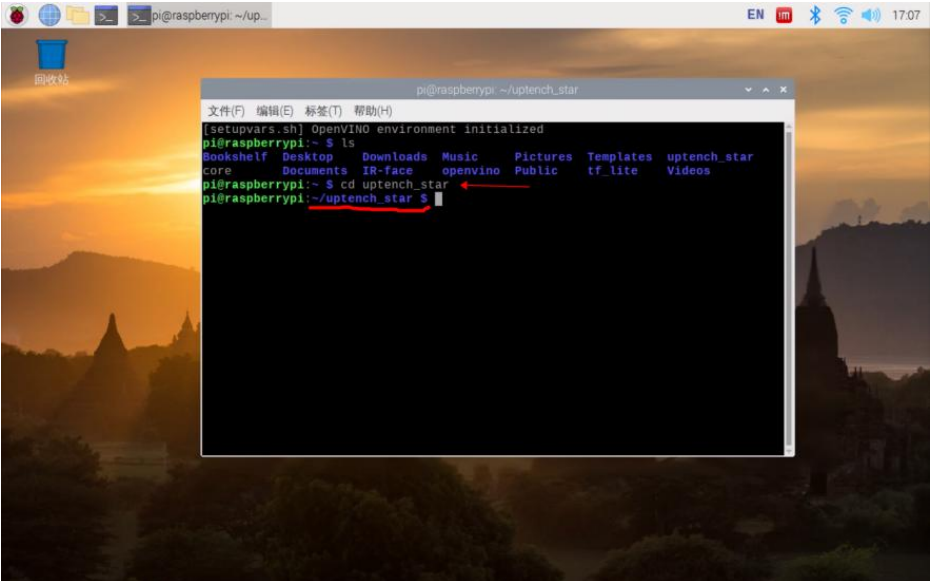
**注：Python语言十分强调缩进，切忌不可忽视！**

## 代码的执行

打开命令行窗口， 例如要想运行 /home/pi/uptech\_star/test.py

首先打开终端进入/home/pi/uptech\_star 目录下：

cd uptech\_star



运行 test.py 程序：

python3 test.py

## 2021人机协同比赛调试经验总结

1. 在调试人工智能模块之前，需要提前学习Python基础语法规则，基本语句的相关知识学习，无需非常深入，但基本语法规则必须学习，这有利于借鉴学习例程。
2. 在调试过程中可巧妙应用Print（）函数，判断代码实际运行位置，是否执行该代码，存在代码卡死时的具体卡死位置。这一点对于刚入手学习的人来说是一个很好用的方法。
3. ApriTag二维码函数说明：

tags = self.at\_detector.detect(gray)

print(tags)的输出如下：

[Detection(tag\_family=b'tag36h11', tag\_id=2, hamming=0, goodness=0.0, decision\_margin=44.4796028137207, homography=array([[ 5.66164684e-02, -6.76563277e-01, 7.74163167e-01],

[ 7.44587789e-01, 2.04775598e-02, 1.05098096e+00],

[ 2.54090716e-04, 2.53395218e-04, 6.64674578e-03]]),

center=array([116.47251037, 158.11962637]),

corners=array([[227.08111572, 46.57167435],

[226.75535584, 267.03375244],

[ 21.55596352, 253.84225464],

[ 6.16658306, 49.18270493]]))]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**tag\_family**：我们使用的是TAG36H11

AprilTag的数字家庭

返回：

• 返回AprilTag的数字家庭。

• image.TAG16H5

• image.TAG25H7

• image.TAG25H9

• image.TAG36H10

• image.TAG36H11

• image.ARTOOLKIT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**tag\_id：**

AprilTag的数字ID

返回AprilTag的数字ID。

• TAG16H5 -> 0 to 29

• TAG25H7 -> 0 to 241

• TAG25H9 -> 0 to 34

• TAG36H10 -> 0 to 2319

• TAG36H11 -> 0 to 586

• ARTOOLKIT -> 0 to 511

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**hamming:**

AprilTag的可接受的数位误差数值

TAG16H5 -> 最多可接受0位错误

TAG25H7 -> 最多可接受1位错误

TAG25H9 -> 最多可接受3位错误

TAG36H10 -> 最多可接受3位错误

TAG36H11 -> 最多可接受4位错误

ARTOOLKIT -> 最多可接受0位错误

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**goodness:**

AprilTag图像的色饱和度

返回AprilTag图像的色饱和度（取值0.0 - 1.0），其中1.0为最佳。

注解:目前这一数值通常是0.0。未来我们可以启用一个称为“标签细化”的功能，以实现对更小的AprilTag的检测。然而，现在这个功能将帧速率降低到1 FPS以下。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**decision\_margin:**

AprilTag匹配的色饱和度

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**homography:**

3x3 单应矩阵描述从“理想”标签（角位于 (-1,1)、(1,1)、(1,-1) 和 (-1, -1)）到像素中的像素的投影

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**center:**

AprilTag的中心位置（x,y)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**corners:**

对象的四个角组成的四个元组(x,y)的列表,四个角通常是按照从左上角开始沿顺时针顺序返回的. 0 1

3 2

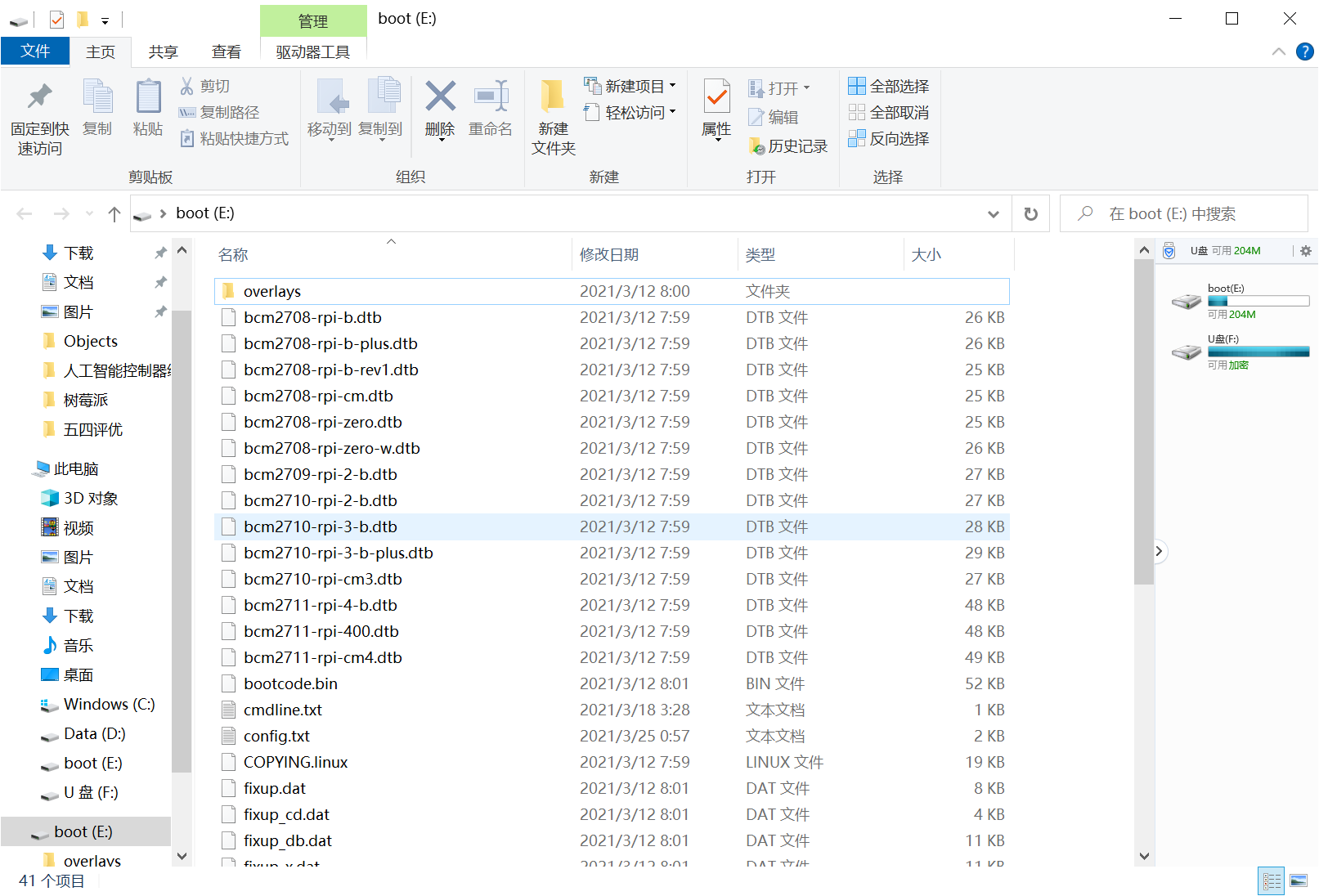
corners[0][0]：代表左上角点的相对于摄像头水平位置方向关系(左边数值小，右边数值大)

corners[0][1]: 代表左上角点的相对于摄像头竖直位置方向关系(上方数值小，下方边数值大)

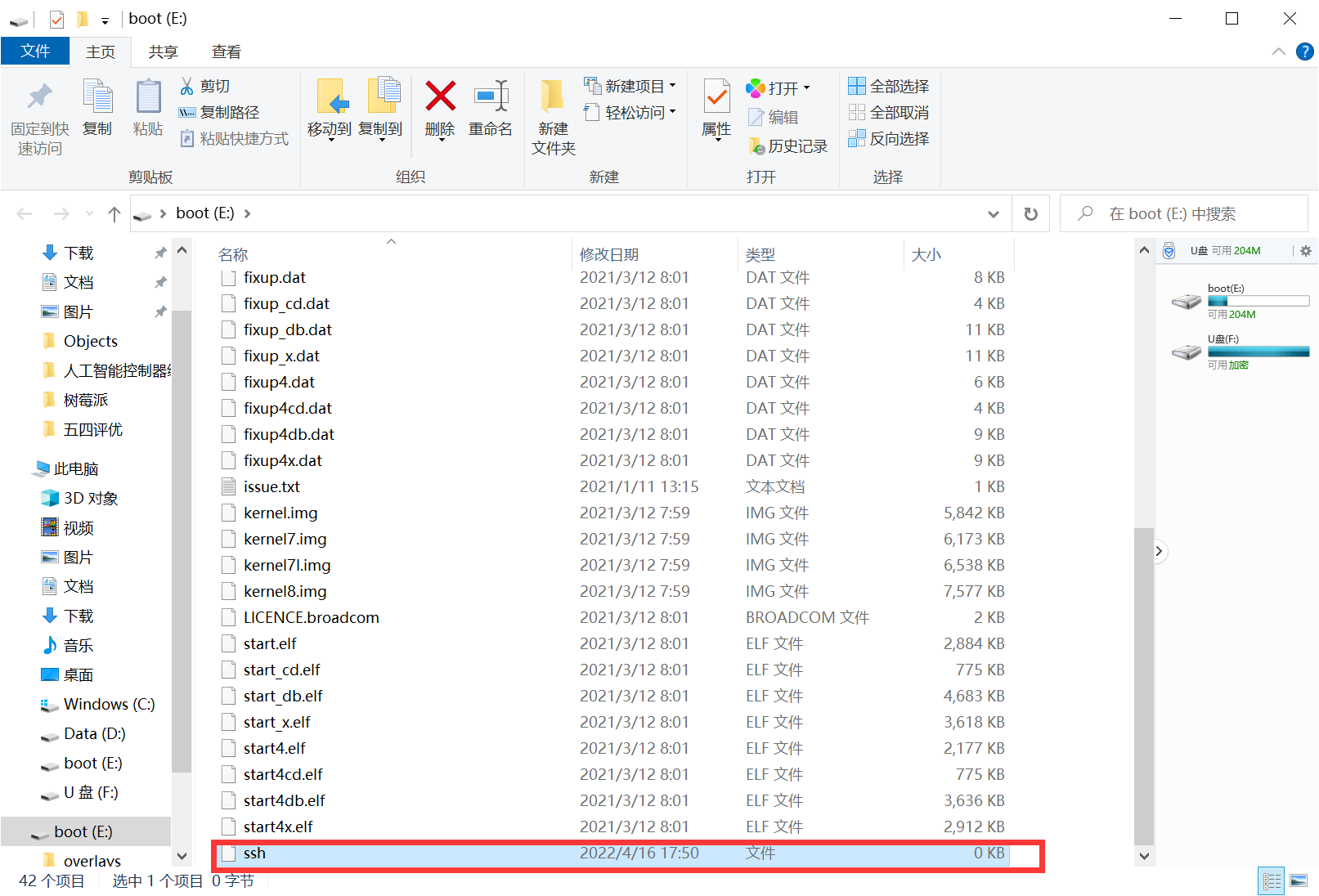
**附件一：**

1. 所需器件：一张已经镜像过的SD卡，一台电脑，一个读卡器
2. 步骤：

* 通过读卡器打开E盘



* 新建添加ssh文件（空文本文档只需更改文件名）



* 新建WiFi配置文件文档（新建空白文本）

打开并填入以下内容：

country=CN

ctrl\_interface=DIR=/var/run/wpa\_supplicant GROUP=netdev

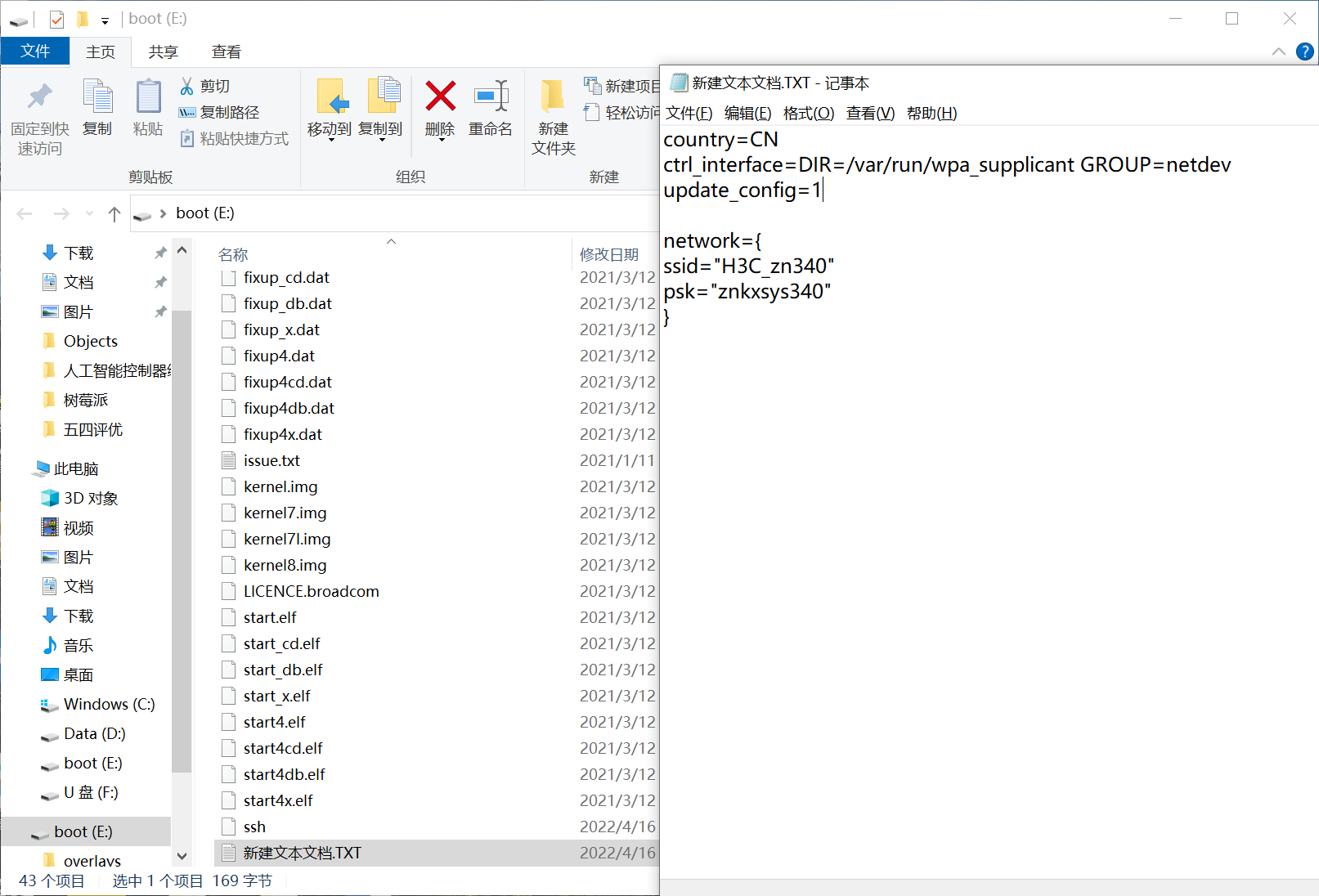
update\_config=1

network={

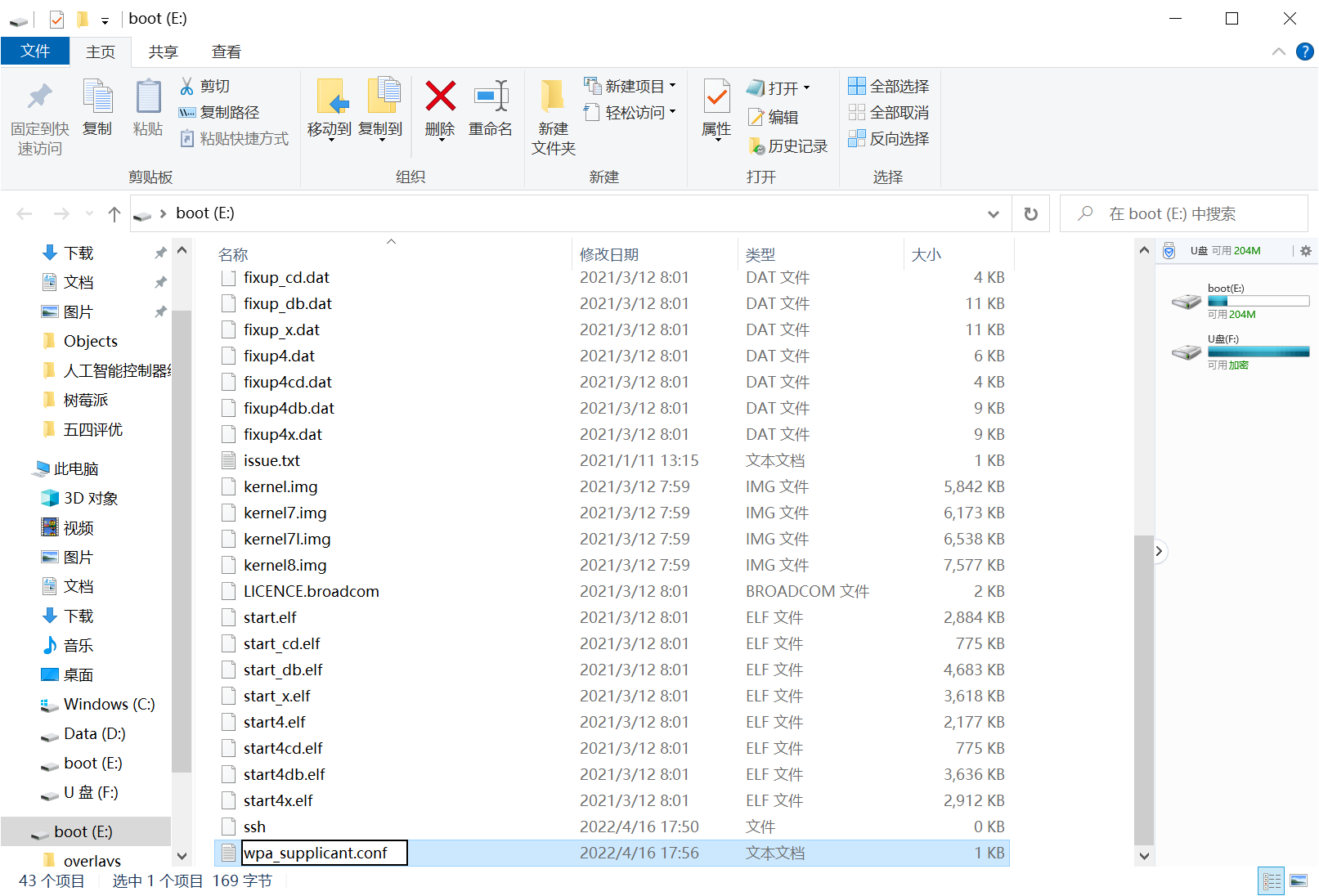
ssid="你的WIFi的名字"

psk="你的WIFI的密码"

}



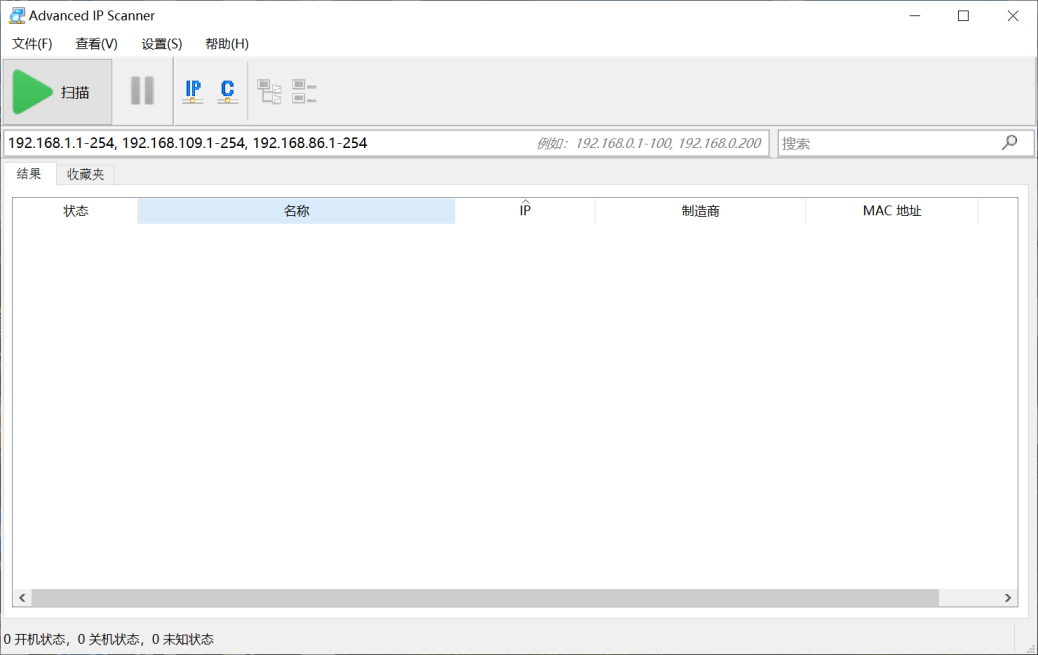
保存好后，需更改文件名称为：wpa\_supplicant.conf



这样就将WiFi配置文件完成了。

* 将配置好的SD卡插入树莓派并进开机，电脑也连入同一WiFi网络。
* 紧接着进行远程操控。（这里主要介绍VNC的使用方法）
  + 下载advanced IP 软件搜索IP地址





点击扫描即可查找树莓派IP地址。

通过Mac地址去判断哪一个IP地址是树莓派的。（第一次只能自己去尝试，之后可记住Mac地址以后更换网络可直接查找该MAC地址即可）

* + 下载putty进行开VNC服务连接



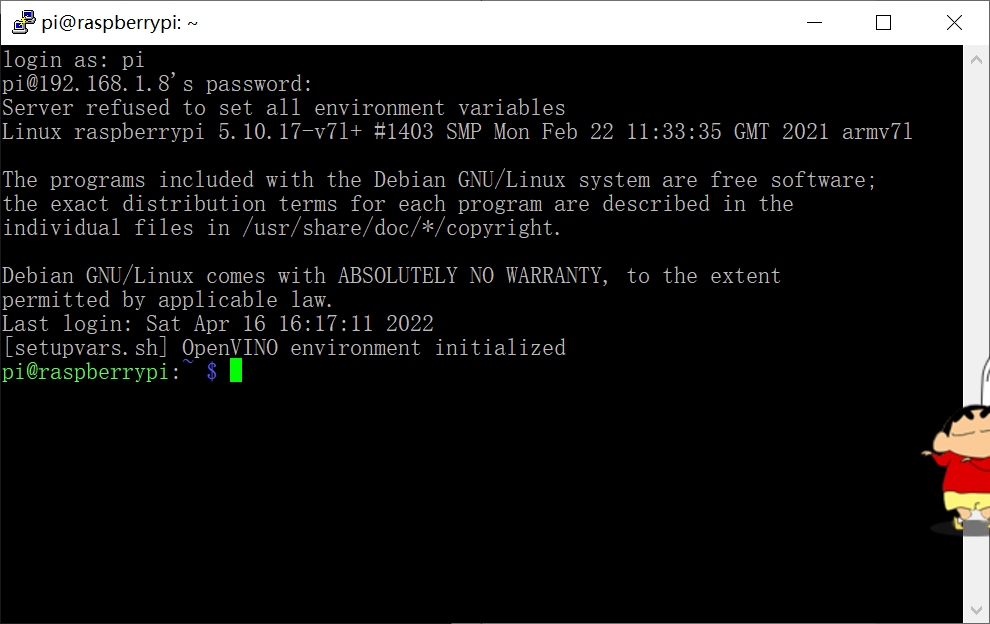
第一个方框填写IP地址，下面选择ssh模式。回车，如果IP地址正确则会弹出一个登陆界面



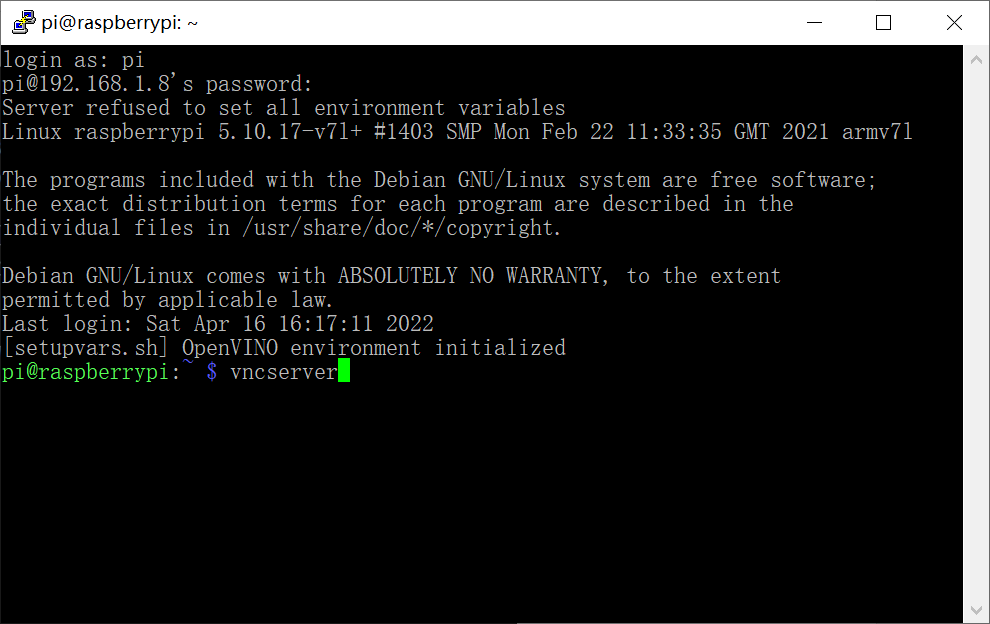
用户名：pi

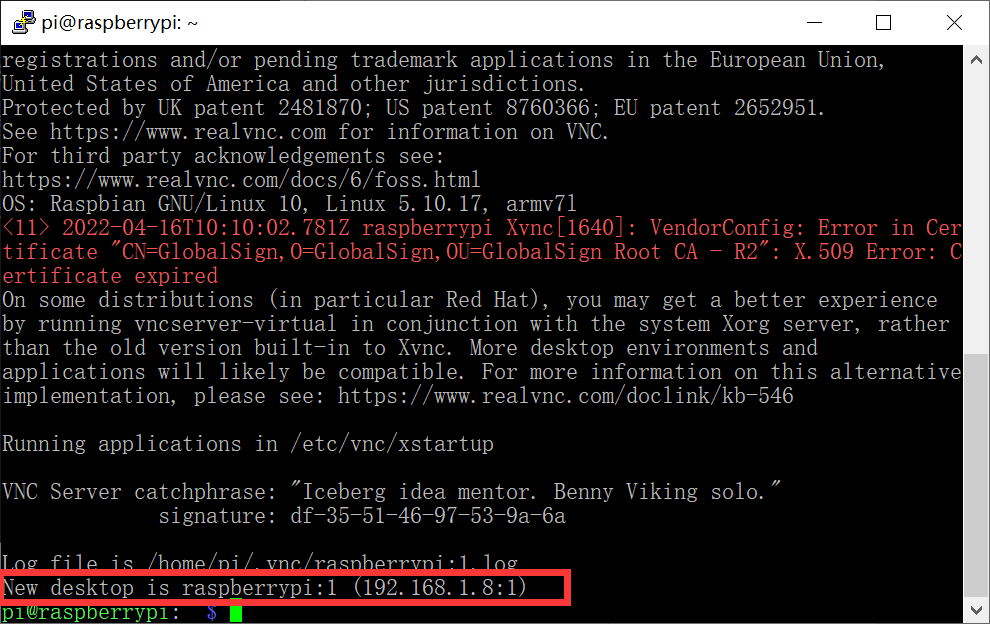
密码：bcsh

成功登陆后界面：

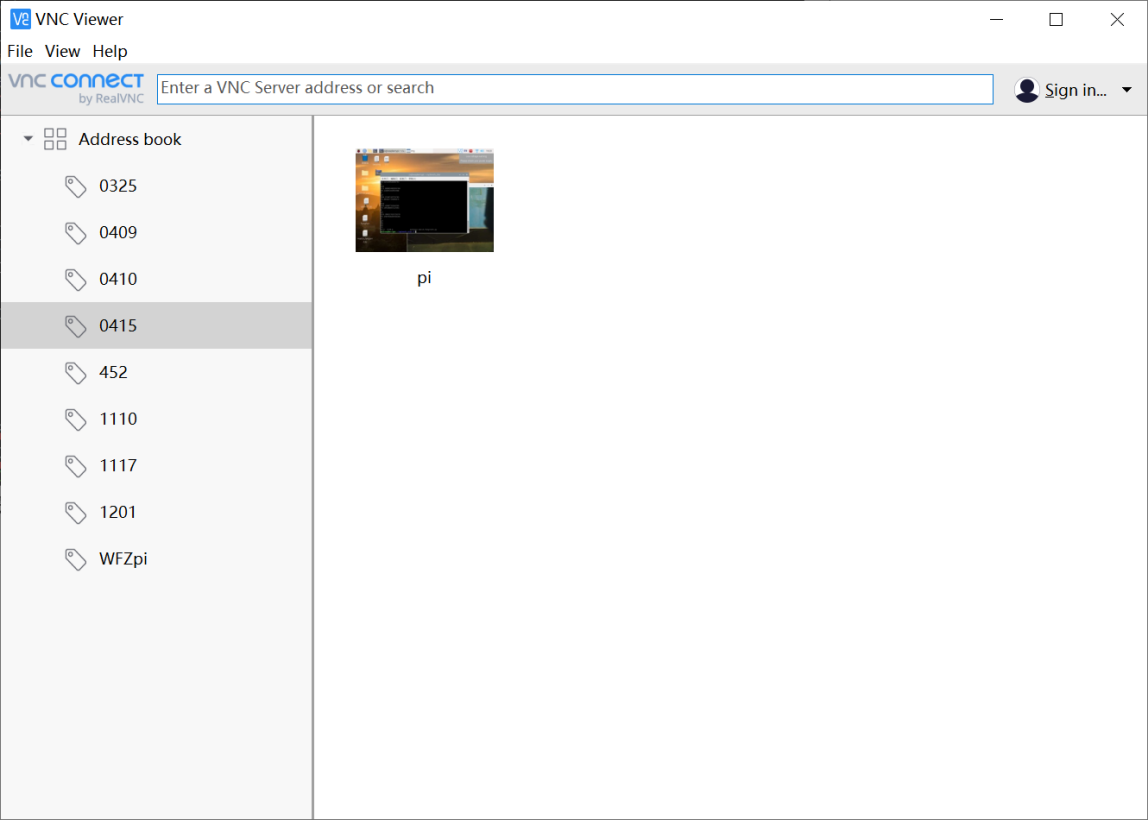


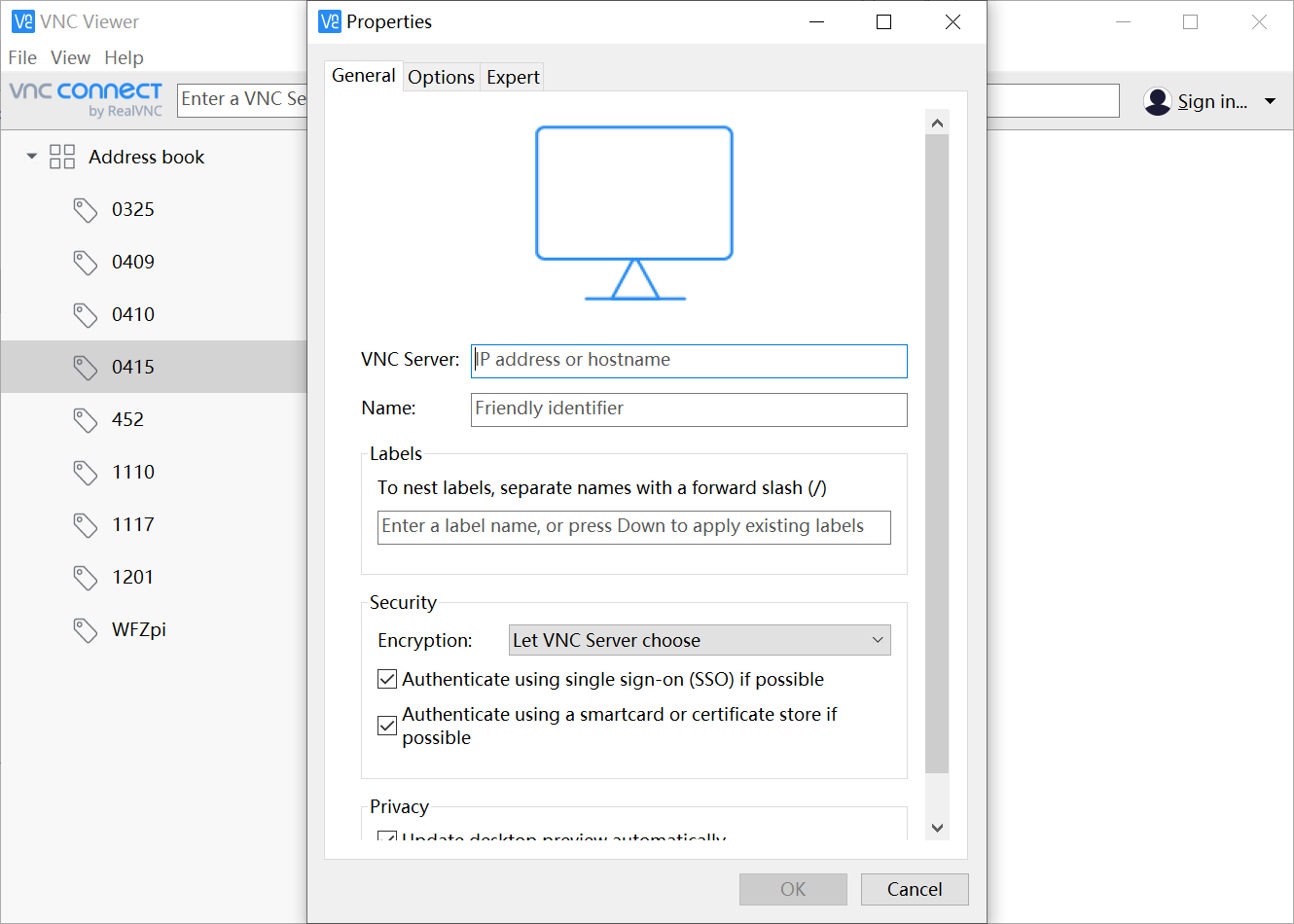
紧接着输入打开VNC服务的指令：





红框部分为打开VNC服务的端口号，在VNC打开配置时需要配置。

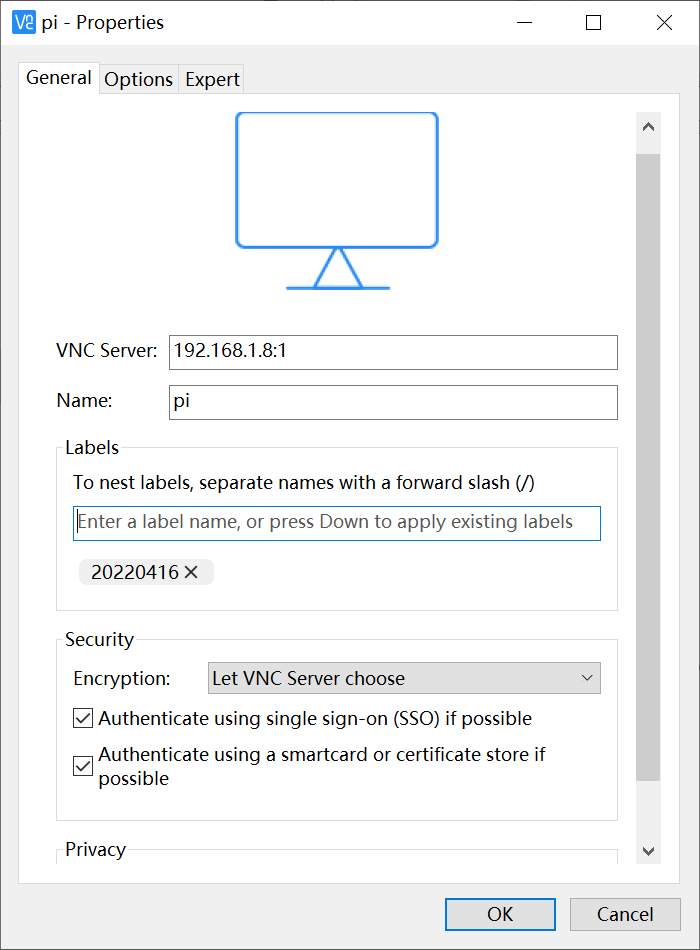




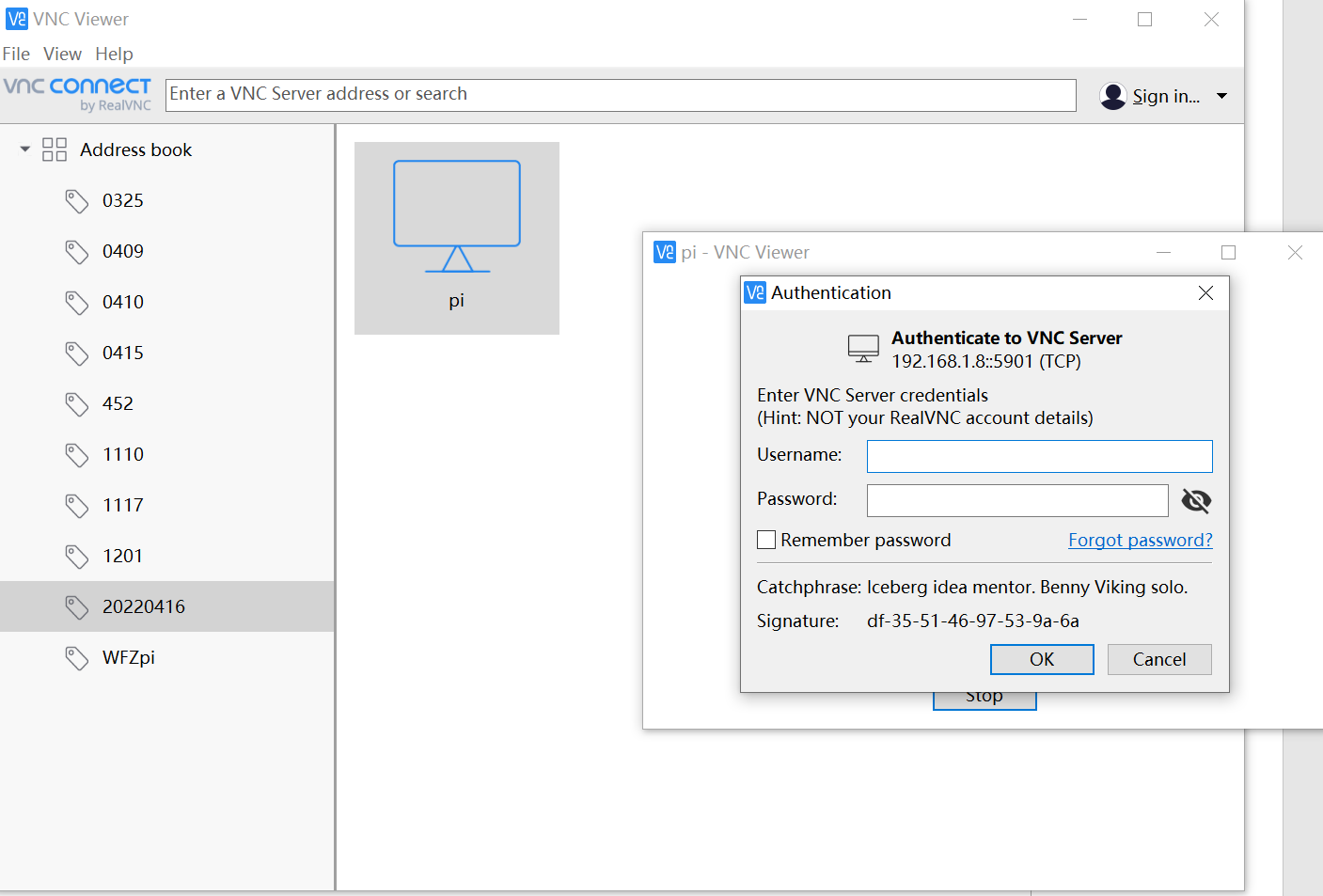
VNC Server 为：刚刚红框内的内容（具体网络根据相关内容）

Name:pi

Labels:可根据自己情况填写便于下次查找

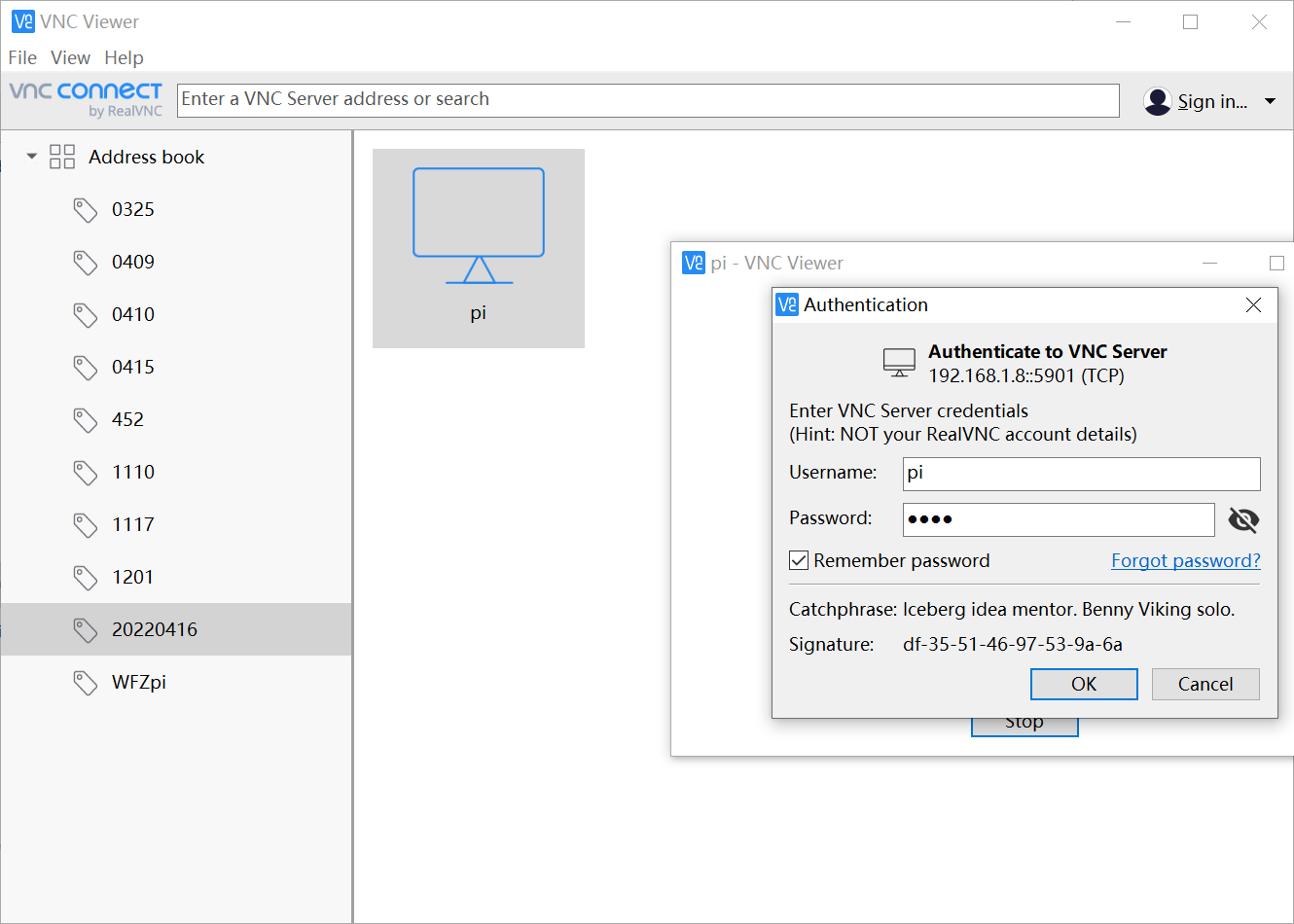


回车后找到该标签并双击pi后：



Username为：pi

Password为：bcsh



打开后便可登陆树莓派界面，到这远程操控便完成了。

