IoT Platform

Huanjia Liu ID: 831993149
Electronic Engineering
Colorado State University
Fort Collins, Colorado
sjmxx1414@gmail.com

***Abstract*—With the rise of IoT technology, more and more IoT devices are coming into everyone's life, but most of the devices have been customized by manufacturers to serve their own IoT systems. This makes some users have to pay for some IoT features that they can't use. So I envision a DIY IoT platform where users can easily add their own IoT client devices according to their needs. Also for security reasons, I added a face recognition system to this platform, so that only authenticated users can use the Internet features.**

I. Introduction

Because of the development of technology and communication methods, the Internet of Things has become a trend in just the last decade. Internet of Things (IoT) is a system that calculates the interrelationship of devices, machines, and digital machines, with a universal unique identification code (UID) and the ability to transmit data through the network without human interaction or human interaction with devices [1], so the definition itself is still focused on the Internet and is a way to connect things to things. It is a way to connect things. However, in terms of the current development trend of IoT, it is no longer a simple Internet, but further becomes a kind of abstract system, or ecology, and we are not focusing on its network connection, but what kind of system is constituted by using IoT.

So I'm thinking about building my own set of IoT ecology, including servers and major types of clients. I also want to design some kind of security device to set the access to IoT. I will talk about the topology of my IoT devices in section II, the functions and settings of the server in section III, and the various types of clients in section IV.

II. IoT topology structure

My IoT consists of a server, a google home mini, a secure authentication server, and several clients(Fig. 1). google home mini acts as a microphone, accepts and analyzes the commands conveyed by the user, and sends the commands to a cloud server, which sends an http ruquest from the remote Google server to my local server. Although google home mini and my local server are actually under a LAN, because of some limitation in Google's production, google home mini cannot send http requests directly to the local ip, and therefore google home mini's arrow in my topology diagram is one-way, and the user then sends any request back to my local server via my local The user then sends any request back to google home mini through my local server for it to have any reply, because google home mini does not have the ability to bind the public ip.

My server is built from a piece of Raspberry Pi 3b, which is directly connected to the Internet. The server has three types of clients, one is a single-end connected security authentication server, built by a Raspberry Pi 4, through the camera to complete face recognition, when the face passes the security authentication, send security authentication to the server, the server can only execute various commands issued by the user.

After getting authentication, the instructions are recognized by the server and will be distributed to various clients, who will complete the instructions and return the results to the server for storage and display on an OLED screen.

My client devices are broadly divided into two types. 1. sensor-shaped, containing various types of environmental detection sensors, which can detect environmental data, such as temperature, air pressure, humidity, carbon dioxide concentration. 2. control-shaped, which can remotely control various home appliances. Most of the clients are composed of ESP8266 and Arduino, the details will be discussed in Section IV.

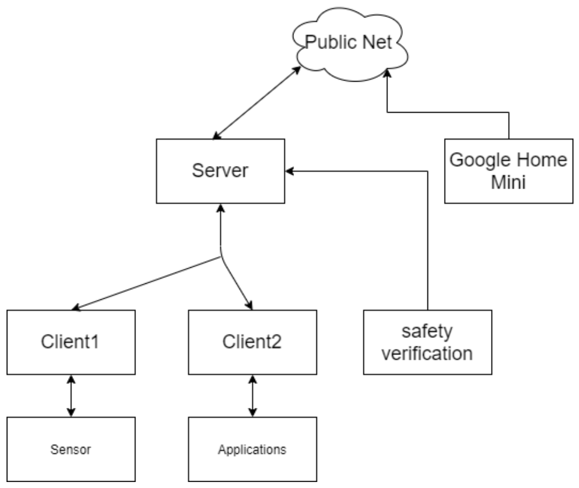


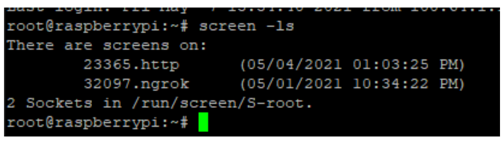
Fig. 1. IoT structure

III. Server

My server is built on a Raspberry Pi 3B with port 22, port 8030 and port 80 open. port 22 is responsible for ssh connections and remote access to the server. port 8030 is responsible for accepting and sending UDP datagrams, interacting directly with clients of all types. port 80 is the http/https server port and is responsible for accepting data from port 80 is the http/https server port, which is responsible for receiving commands from google home mini.

A. Public Network Connection

The public ipv4 resource is relatively short, so it is hard to apply for public ipv4 address[2], so we have to find a way to map the intranet ip address to the public network. Luckily, ngrok is a tool that does just that. Ngrok is a reverse proxy that creates a secure tunnel from a public endpoint to a locally running web service. The connection tunnel established by ngrok is secure and can only transmit data to the localhost port you have open. It would be difficult to do any damage[3]. So, here I mapped my port 80 to the public network and used "flask" a micro web framework written in Python, to get http request. In my server, there are two processes, (Fig. 2) one is for ngrok, and the other is for my http control center.



```
root@raspberrypi:~# screen -ls
There are screens on:
        23365.http      (05/04/2021 01:03:25 PM)
        32097.ngrok     (05/01/2021 10:34:22 PM)
2 Sockets in /run/screen/S-root.
root@raspberrypi:~#
```

Fig. 2. Server processes

B. HTTP Center

Here I set http://local/post as my http request accept address to get JSON post request. The Json post should be formatted as "obj":command. The command included in environment list will be set as mode 1, the command included in applications list will be as mode 2. If the command is authentication, recording the request time in a txt file for demonstration verification, which will be mentioned in the next subsection.

Here is my detail code.

```
from flask import Flask, request, jsonify
from control_center import control_center
import re
import datetime
import time
app = Flask(__name__)

en_list =
     ["humidity","temperature","pressure"]
light_list = ["on","off"]
@app.route('/post', methods=['POST'])
def post():

    content = request.json
    google_string = content["obj"]
    command = re.findall(r'[a-zA-Z]+',
         google_string)

    mode = 0
    if command[0] in en_list:
        mode = 1
        control_center(mode,command[0])
    elif command[0] in light_list:
        mode = 2
        control_center(mode,command[0])
    elif command[0] == "auth":
        with open("auth_time.txt","w") as f:
            f.write(str(time.time()+30))
    return 'get'

if __name__ == '__main__':
    app.run(host=
        '0.0.0.0',debug=True,port=8030)
```

C. UDP Center

UDP center is the place to process my UDP datagramsending and receiving. It's quite easy, that just gain the command and send to specific client ip address and port. I won't go into details here, just put out the code.

```
import socket
from datetime import datetime
import sys

class UDP_center:

    def __init__(self, ip, port):
        self.port = port      #Set server's port
        self.ip = ip

    #For environment client
    def client_en(self,message):
        IP = self.ip
        Port = self.port

        binary_message = message.encode('ascii')


        clientSock =
             socket.socket(socket.AF_INET,
             socket.SOCK_DGRAM)
        clientSock.bind(('',8020))

        try:
            clientSock.sendto(binary_message,(IP,Port))
        except RuntimeError:
            print(f"error: {RuntimeError}")

        got = False

        while(got != True):
            print('waiting')
            message,address =
                 clientSock.recvfrom(512)
            got = True
        return message.decode('utf-8')
    #For application client
    def client_light(self,message):
        IP = self.ip
        Port = self.port
        binary_message = message.encode('ascii')


        clientSock =
             socket.socket(socket.AF_INET,
             socket.SOCK_DGRAM)
        clientSock.bind(('',8020))

        try:
            clientSock.sendto(binary_message,(IP,Port))
        except RuntimeError:
            print(f"error: {RuntimeError}")
```

D. Control Center of Server

Control center is the center responsible for distributing various commands to the client and for verifying the authentication mentioned earlier. If the time now is to be greater than the request sending time plus 30 seconds, in other words, the authentication is valid within 30 seconds of the authentication request being sent. If the authentication is valid then the request command request is sent to the client. Meanwhile the function server_disp() can send the result to the OLED screen through the GPIO port of the Raspberry Pi. If authentication fails, error N/A is displayed on the screen(Fig. 3).

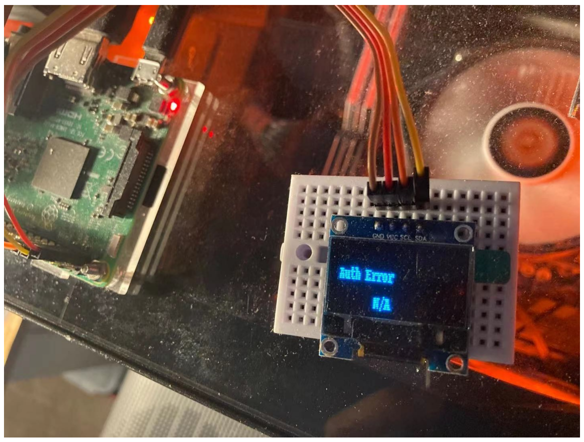


Fig. 3. Authentication failure

Here is part of my code.

```
class control_center():
   def __init__(self,mode, command):

      auth = False
      with open("auth_time.txt","r") as f:
         my_time = float(f.readline())
         if time.time()<my_time:
            if(mode == 1):
               result =
                  self.environment_udp(command)
               self.server_disp(result,command)
            elif(mode == 2):
               self.light_udp(command)
         else:
            self.server_disp("N/A","Auth
               Error")
```

E. Google Home Mini

Google Home mini is a voice-activated speaker powered by the Google Assistant.(Fig. 4) It is already knitted into the speech recognition technology that Google has[4], so all we need to do is add our own commands to it, perform keyword detection, and then package the recognized words into an http post request to the URL that specifies our local server.

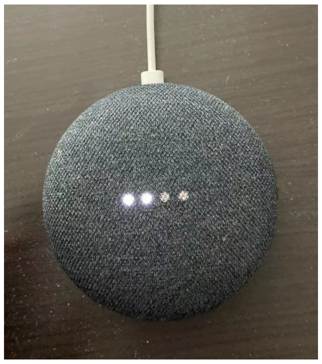
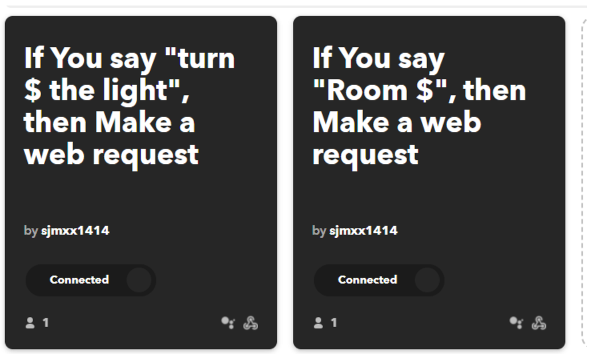


Fig. 4. Google Home mini

Here I create two command for two kinds of clients.(Fig .6) Using command "Room xxx" can run the specific environment sensor. Using command "Turn xxx the light" can turn off or turn on specific applications.



If You say "turn $ the light", then Make a web request
by sjmxx1414
Connected
1

If You say "Room $", then Make a web request
by sjmxx1414
Connected
1

Fig. 5. Key word detecting

IV. Client

Next is the client part, as I mentioned above, here we use a total of three types of clients, security authentication, environment detection, and application control, in this section I will explain each client one by one.

A. Security Authentication

The client for security detection consists of a Raspberry Pi 4b, camera, two servos, and an Arduino to control the servos. The Raspberry Pi 4b was chosen because it has 8G of memory and a processing rate of 1.5GHZ, which is quite suitable for real-time image detection[5].

1) Convolutional Neural Networks: I chose Keras as my convolutional neural network training tool at the beginning because I needed to detect the identity of the user through image recognition. Keras is an open-source software library that provides a Python interface for artificial neural networks.

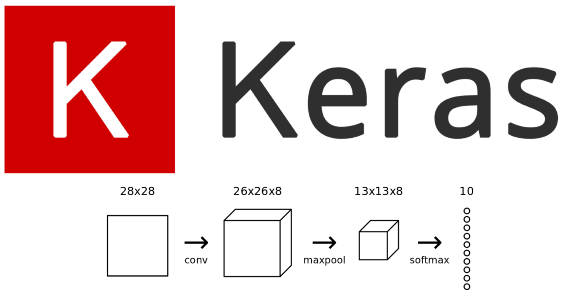


Fig. 6. Key word detecting

Because this IoT system is for personal use, I only used the dichotomous classification algorithm to train my model, but of course it is not difficult to add multi-person face detection, and it is enough to change the algorithm and retrain the network. Here I will only explain the training process of the binary classification network.

Here I have built a 7-layer convolutional neural network, as shown in the code below, because this is only a binary classification.

```
def model():
   model = models.Sequential()
   model.add(Conv2D(32, (3, 3),
       activation='relu', input_shape=(500,
       500, 3)))
   model.add(MaxPooling2D((2, 2)))
   model.add(Conv2D(64,(3,3),activation='relu'))
   model.add(MaxPooling2D((2,2)))
   model.add(Conv2D(64, (3, 3),
       activation='relu'))
   model.add(MaxPooling2D((2, 2)))
   model.add(Conv2D(128, (3, 3),
       activation='relu'))
   model.add(MaxPooling2D((2, 2)))
   model.add(Conv2D(256, (3, 3),
       activation='relu'))
   model.add(MaxPooling2D((2, 2)))
   model.add(Conv2D(512, (3, 3),
       activation='relu'))
   model.add(MaxPooling2D((2, 2)))
   model.add(Flatten())
   model.add(Dropout(0.5))
   model.add(Dense(1024, activation='relu'))
   model.add(Dense(1, activation='sigmoid'))

   return model
```

At the same time, in order to improve the accuracy, I add some shift, rotate and scale to my training figure using the function "ImageDataGenerator" provided by Keras like the code below.

```
def processing_data(data_path, height, width,
    batch_size=32, test_split=0.1):


   train_data = ImageDataGenerator(

         rescale=1. / 255,

         shear_range=0.1,

         zoom_range=0.1,

         width_shift_range=0.1,

         height_shift_range=0.1,

         horizontal_flip=True,

         vertical_flip=True,

         validation_split=test_split
   )


   test_data = ImageDataGenerator(
         rescale=1. / 255,
         validation_split=test_split)

   train_generator =
       train_data.flow_from_directory(

         data_path,

         target_size=(height, width),

         batch_size=batch_size,

         class_mode='binary',
         classes=['benign','malignant'],

         subset='training',
         seed=0)
   test_generator =
       test_data.flow_from_directory(
         data_path,
         target_size=(height, width),
         batch_size=batch_size,
         class_mode='binary',
         classes=['User', 'Unknown'],
         subset='validation',
         seed=0)

   return train_generator, test_generator
```

Using the gradient descent optimizer, the training is quite smooth and the accuracy rate is quite high (Fig. 7). But when I applied trained h5 file into my raspberry pi, I found Keras is not very suitable for real-time monitoring, it runs too slowly[6]. Thus I had to drop this plan and moved to using Opencv directly.

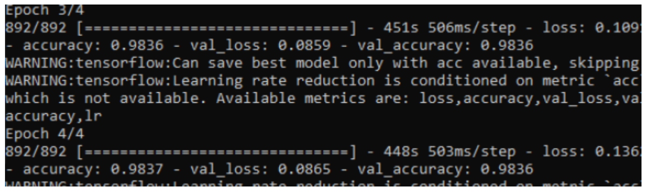
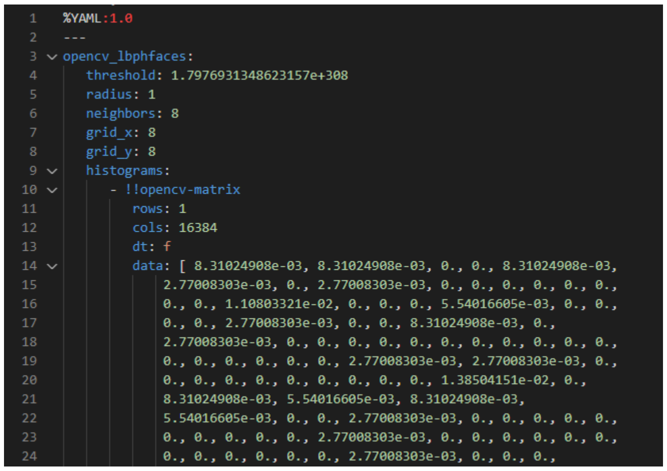

```
Epoch 3/4
892/892 [==============================] - 451s 506ms/step - loss: 0.109
- accuracy: 0.9836 - val_loss: 0.0859 - val_accuracy: 0.9836
WARNING:tensorflow:Can save best model only with acc available, skipping
WARNING:tensorflow:Learning rate reduction is conditioned on metric `acc
which is not available. Available metrics are: loss,accuracy,val_loss,va
accuracy,lr
Epoch 4/4
892/892 [==============================] - 448s 503ms/step - loss: 0.136
- accuracy: 0.9837 - val_loss: 0.0865 - val_accuracy: 0.9836
```

Fig. 7. Model Accuracy

2) *Opencv part:* As luck would have it, Opencv not only has a built-in pre-trained face detection file, but also provides an exhaustive face detection training function, “cv2.face”, which helped me a lot and saved some of the time wasted in the Keras section.

In addition, we can also use the official Opencv face detection pre-training model "haarcascade_frontalface_default.xml" for training, cutting out the extra parts of our photos and keeping only the face part, which not only greatly improves the training speed, but also improves the training accuracy. Here is the screen shot of my trained xml file.



```
%YAML:1.0
---
opencv_lbphfaces:
   threshold: 1.7976931348623157e+308
   radius: 1
   neighbors: 8
   grid_x: 8
   grid_y: 8
   histograms:
      - !!opencv-matrix
         rows: 1
         cols: 16384
         dt: f
         data: [ 8.31024908e-03, 8.31024908e-03, 0., 0., 8.31024908e-03,
             2.77008303e-03, 0., 2.77008303e-03, 0., 0., 0., 0., 0., 0.,
             0., 0., 1.10803321e-02, 0., 0., 0., 5.54016605e-03, 0., 0.,
             0., 0., 2.77008303e-03, 0., 0., 8.31024908e-03, 0.,
             2.77008303e-03, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 2.77008303e-03, 2.77008303e-03, 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 1.38504151e-02, 0.,
             8.31024908e-03, 5.54016605e-03, 8.31024908e-03,
             5.54016605e-03, 0., 0., 2.77008303e-03, 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 2.77008303e-03, 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 2.77008303e-03, 0., 0., 0.,
```

Fig. 8. XML trained file

3) Secure authentication: When the user's face detection passes authentication, Raspberry Pi sends an http request to our server for successful authentication and records the time. The authentication is valid for 30 seconds. To remind the user whether the authentication is passed or not, I added a small LED to the Raspberry Pi connected to GPIO port 15. The small light glows when the authentication is passed, and the small light goes off when the authentication fails, and the small light also goes off when the authentication time is exceeded, the specific code is as follows.

To prevent sending http requests to the server too often when the user is in front of the camera all the time, I set the authentication request to be updated every 5 seconds to maintain the stability of the server.

```
if(id != user):
   auth_time = time.time()
   if((last_time+5)<auth_time):
      last_time = auth_time
      requests.post(url, json =
          {"obj":"auth"})
if((auth_time+30) > time.time()):
   GPIO.output(15, GPIO.HIGH)
else:
   GPIO.output(15, GPIO.LOW)
```

4) Camera follow: This subsection is once part of the proposal of the balance of the plan, but because of time and economic reasons, not fully realized and left behind the appendage. In fact, it is not difficult to implement. Opencv will provide the x and y coordinates of the face as well as the width and height of the face when detecting the face. We only need to calculate the position of the center of the face and send the rotation direction to the Arduino control servo according to its position relative to the overall picture. Here I use the GPIO output high level to the Arduino port and let Arduino use "digitalRead" function to monitor the port voltage status in real time. 4 ports correspond to 4 directions

```
if center_v < 200:
   GPIO.output(pin_down, GPIO.HIGH)

elif center_v >280:
   GPIO.output(pin_up, GPIO.HIGH)

else:
   GPIO.output(pin_up, GPIO.LOW)
   GPIO.output(pin_down, GPIO.LOW)

if center_h < 300:
   GPIO.output(pin_right, GPIO.HIGH)

elif center_h >340:
   GPIO.output(pin_left, GPIO.HIGH)

else:
   GPIO.output(pin_right, GPIO.LOW)
   GPIO.output(pin_left, GPIO.LOW)
```

This is a picture of the final product

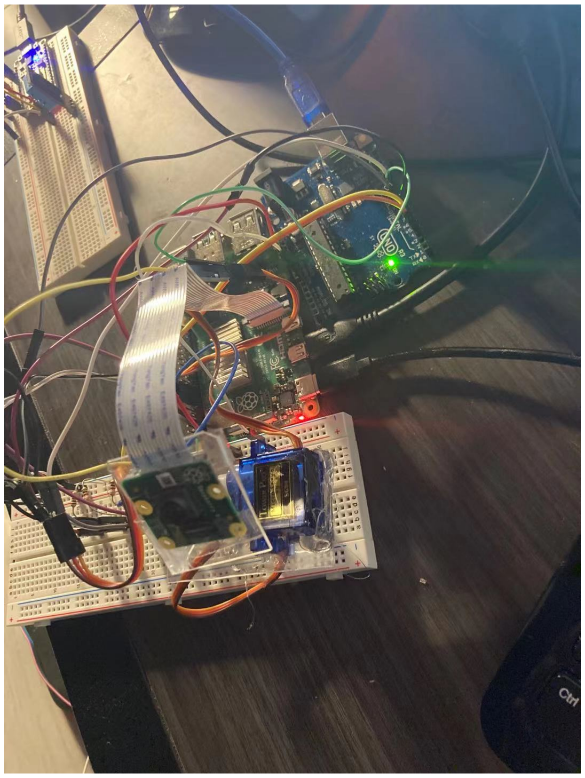


Fig. 9. Security Client

B. Environment Detection

Here I used a combination of ESP8266 and Arduino to build the client. The ESP8266 is a low-cost Wi-Fi microchip, with a full TCP/IP stack and microcontroller capability[7]. The ESP8266 is responsible for communicating with the server, while the Arduino is responsible for reading the degrees of the sensors. Actually the ESP8266 comes with some of the Arduino's functions, but due to its memory limitation and inability to use multi-threading, I added an extra Arduino Uno board here to take care of the sensors exclusively. (Fig. 10)

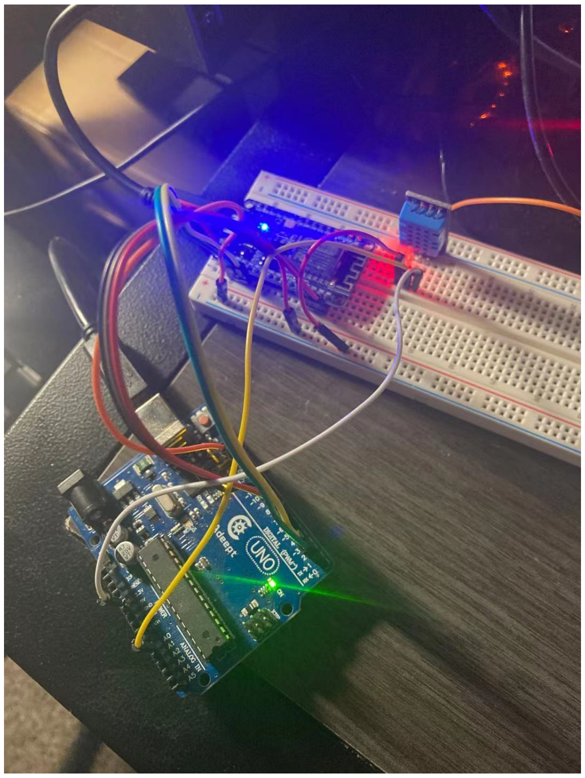


Fig. 10. Environment Detection Client

1) Connection: My original plan was to use serial communication to achieve the connection between two boards, but after testing, I found that the bidirectional serial communication is not stable, when one board is closed for reading, the serial communication data of the other board will be left in the board's memory, and the next time it is read, the data left behind will be read, resulting in data errors. To solve this error, we need to do accurate reading and closing at the same time, I have not thought of any good way to solve this problem, so I still use the serial port plus the high and low level with the way. Because what the Arduino needs is only the switch signal of the corresponding sensor, while what the ESP8266 needs is the complete data obtained from the sensor.

Therefore, from the ESP8266 to the Arduino I still use the same method of changing the level from the ESP8266 to the Arduino that I mentioned above for the Raspberry Pi to manipulate the servo. The Arduino to ESP8266 still uses serial communication.

Here is my code of ESP8266 UDP function. The variable "sum" here is used to limit the runtime of the packet acceptance function. Because the UDP packet receiving function "parsePacket" in ESP8266 does not work repeatedly, it will choose to accept the empty data even if it is received. At the same time, ESP8266 is a single-threaded board, we can't let the UDP packet accept function take up all the running time, we still need to run other instructions. Therefore, after every 500 packet data acceptance, we jump out of the loop to execute other instructions such as manipulating electrical equality, and then continue the UDP acceptance loop after completing the instructions. Thanks to the board's fast enough core frequency, this single-threaded multi-purpose approach doesn't affect the reception of UDP packets too much.

```
void test::Udp_Server(WiFiUDP Socket){
  int sum = 0;
  while(sum != 500)
  {
    char* packetBuffer = (char*) malloc (2);

    int packetSize = Socket.parsePacket();

    if (packetSize!=0)
    {

      Socket.read(packetBuffer,
          UDP_TX_PACKET_MAX_SIZE);

      Serial.println(packetBuffer);
      char* toUno_buf =
          uno_center(packetBuffer);
      char temp[5] = "0000";
      for(int i =0; i<4;i++)
      {
        temp[i] = toUno_buf[i];
      }

      Socket.beginPacket(server_ip,
          server_port);
      Serial.println(temp);
      Socket.write(temp);

      Socket.endPacket();
    }

    free(packetBuffer);
    sum ++;
  }
}
```

Also here are my two function two communication with Arduino boards. "uno_center" sends the high voltage signal to Arduino board to turn on the sensor, and "read_data" function uses serial communication to get sensor data from Arduino. Note that because of the instability of the serial communication, we need to turn off the Arduino first, and then turn off the ESP8266 when all the data has been read. That's the usage of 2 seconds delay in "read_data" function.

```
char* test::uno_center(char* udp_words){
  char temp[3] = "00";
  temp[0] = udp_words[0];
  temp[1] = udp_words[1];
  byte pin = 0;

  if(strcmp(temp, "hu")==0)
  {
    pin = D1;


  }
  if(strcmp(temp, "tm")==0)
  {
    pin = D0;

  }
  if(strcmp(temp, "pr")==0)
  {
    pin = D2;
  }
  char* final_data = read_data(pin);
  return final_data;

}

char* test::read_data(byte pin){

  char toUno_buf[5] = "0000";
  char* temp = (char*) malloc (2);

  digitalWrite(pin,HIGH);
  delay(500);
  Serial.readBytes(toUno_buf,5);

  Serial.println(toUno_buf);
  digitalWrite(pin,LOW);
  delay(2000);

  return toUno_buf;
}
```

2) Sensor Example: Here I show some code of temperature sensors and humidity sensors. Adding other sensors is not difficult and is basically the same as simply using Arduino with sensors, users can add their own sensors according to their preferences, but note that the serial port output is not stable, so the data length of ESP8266 needs to be consistent with Arduino. For example, here I use 4-bit char data.

```
#include "DHTesp.h"
#include <Arduino.h>
#include <Wire.h>
#define I2C_ADDRESS 0x77
#include <BMP180I2C.h>

DHTesp dht;
BMP180I2C bmp180(I2C_ADDRESS);
void setup() {
   Serial.begin(115200); // put your setup
       code here, to run once:
   pinMode(7,INPUT);

   dht.setup(A1,DHTesp::DHT11);

    Wire.begin();
    bmp180.begin();
  //reset sensor to default parameters.
  bmp180.resetToDefaults();

  //enable ultra high resolution mode for
      pressure measurements
  bmp180.setSamplingMode(BMP180MI::MODE_UHR);


}

char mystr[12] = "hello";
void loop() {

  int val[] = {0,0};
  val[0] = digitalRead(7);
  val[1] = digitalRead(6);
  val[2] = digitalRead(8);

  delay(100);
  //Sending tempreature data
  if(val[0] == 1)
  {
    bmp180.measureTemperature();
    do
    {
      delay(100);
    } while (!bmp180.hasValue());
    float temp_num = bmp180.getTemperature();
    String temp = String(temp_num);
    temp.toCharArray(mystr,5);
    Serial.write(mystr,5 );
  }
 //Sending humidity data
  if(val[1] == 1)
  {
    TempAndHumidity lastValues =
         dht.getTempAndHumidity();
    String temp = String(lastValues.humidity);
    temp.toCharArray(mystr,5);
    Serial.write(mystr,5 );
  }

 //Sending pressure data
if(val[2] == 1)
  {
    bmp180.measurePressure();
    do
    {
      delay(100);
    } while (!bmp180.hasValue());
    float pressure_num = bmp180.getPressure();
    String pressure = String(pressure_num);
    pressure.toCharArray(mystr,5);
    Serial.write(mystr,5 );
  }

  delay(1000);

}
```

C. Application Control

This type of client is roughly the same as the environment detection client, which is also built by ESP8266 and Arduino, and does not need to return specific data to the server, only a one-way connection is needed, so the UDP server is also simpler to build. Because the specific process is similar to the environment detection, we will not repeat it here, readers can refer to the previous subsection.

Here it is necessary to mention the application of relays. A relay is an electrically operated switch, it allow us use low-power signal to control high voltage[8].

Like the figure below (Fig. 11), we use a relay to connect the ESP8266, which can realize a small voltage of 3.3V DC to control 110V AC high voltage. Readers must pay attention to safety when practicing, here we use the bulb is actually only a small 12V voltage drive.

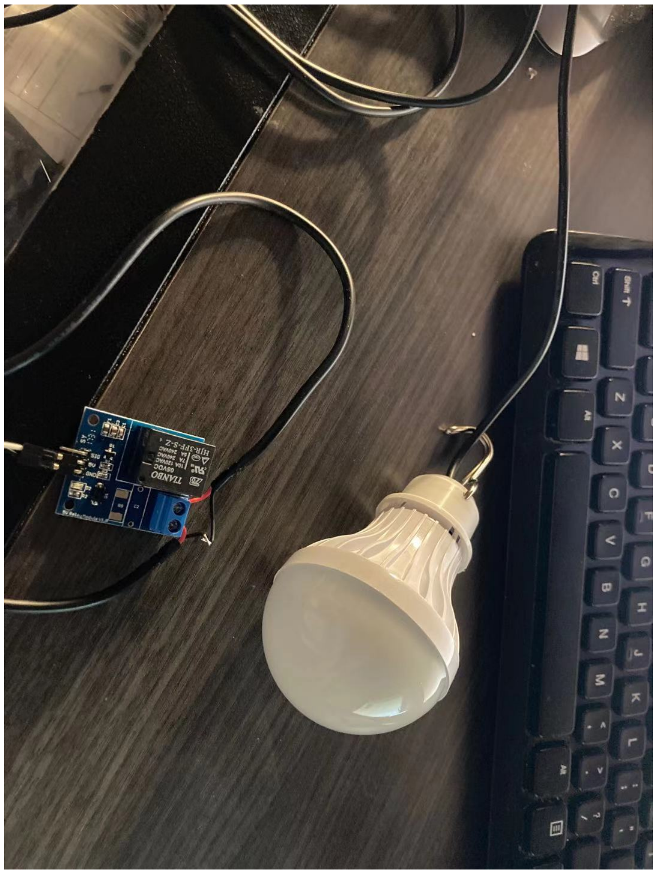


Fig. 11. Application of relay

.

V. Conclusion

IoT as the current big fire application technology, a complete ecosystem is absolutely an integral part. The IoT platform built in this project solves most of the connection and interaction functions, users just need to add sensors or relays according to their needs. Meanwhile, most of the functions are left with interfaces for users to change the ip and port according to their needs. At the same time, considering the security, the face recognition system is added to set the permission of the IoT. This can help users to protect their privacy to some extent.

VI. Reference

[1] Rouse, Margaret. "internet of things (IoT)". IOT Agenda. Retrieved 14 August 2019.

[2] Smith, Lucie; Lipner, Ian. "Free Pool of IPv4 Address Space Depleted". Number Resource Organization. Retrieved 3 February 2011.

[3]Buckler, Craig. “How to Use Ngrok to Share a Local Development Site - SitePoint.” SITEPOINT, 22 Apr. 2021, www.sitepoint.com/use-ngrok-test-local-site.

[4] "The future is AI, and Google just showed Apple how it’s done" Published October 5, 2016, Retrieved July 5, 2018

[5]Eben Upton. "Raspberry Pi 4 on sale now from $35". Raspberry Pi Foundation

[6] DataFlair Team. “Keras vs OpenCV – Differences Between OpenCv and Keras.” DataFlair, 9 July 2020, data-flair.training/blogs/keras-vs-opencv/

[7] "ESP8266 Overview". Espressif Systems. Retrieved 2017-10-02.

[8] BenMimoune A., Kadoch M. Relay Technology for 5G Networks and IoT Applications. In: Acharjya D., Geetha M. (eds) Internet of Things: Novel Advances and Envisioned Applications. Studies in Big Data, vol 25. Springer, Cham. https://doi.org/10.1007/978-3-319-53472-5_1