Junior Secondary

Robotics Learning and Teaching Resources

Extension Unit 2: Cross-platform robotics and advanced robotics concepts

Technology Education Section
Curriculum Development Institute
Education Bureau
The Government of the HKSAR
May 2019

All comments and suggestions related to the resource materials may be sent to:

Chief Curriculum Development Officer (Technology Education)
Technology Education Section
Curriculum Development Institute
Education Bureau
Room W101, West Block, 19 Suffolk Road
Kowloon Tong
Hong Kong

The copyright of the materials in this package, other than those listed in the Acknowledgments section and the photographs mentioned there, belongs to the Education Bureau of the Government of the Hong Kong Special Administrative Region. Schools and educational organisations are welcome to use the content of this package for non-profit making educational purposes. In all cases, proper acknowledgements should be made.

Otherwise, all rights are reserved, and no part of these materials may be used for publication or other purposes in any form without the prior permission of the Education Bureau.

© Copyright 2019

The learning resources is developed by the Robot Institute of Hong Kong.

Content

Chapter 1: Basics of Communication and Automation **P.3**

Chapter 2: Wireless Communication – Bluetooth **P.16**

Chapter 3: Wireless Communication – APP Inventor **P.37**

Extension Unit 2

Cross-platform Robotics and Advanced Robotics Concepts

Chapter 1: Wi-Fi Technology

In Extension Unit 1, we mentioned wireless communication for data transmission in the communication interface of robots, i.e. communication with electromagnetic waves of a specific frequency. In the exercise of that unit, we practised using Bluetooth technology to create interaction between a mobile phone and an Arduino robot. In this unit, we will introduce another communication interface technology – Wi-Fi.

I. What is Wi-Fi?

Wi-Fi is the technology for wireless network communication based on the IEEE 802.11 standards. The 802.11 standards are a set of specifications that defines exactly what methods this technology uses to implement data transmission. The layers to be defined mainly include the physical layer, the media access control (MAC) layer and the data link layer.

We can start exploring the 802.11 standards from the bottom layer. The physical layer specifies the frequencies of radio waves and the transmission speeds of data. Taking 802.11 as an example, the physical layer of 802.11 specifies the frequency of the radio waves to be 2.4 GHz with the fastest data transmission rate of 2 Mbps. The middle layer, the media access control (MAC) layer, specifies the communication methods between hardware to achieve good communication quality. The carrier-sense multiple access with collision avoidance (CSMA/CA) method, as its name implies, suggests that signals are transmitted by broadcasting with avoidance in collision with other radio waves during transmission. The top layer, the data link layer, is responsible for data transmission and verification of the Wi-Fi network environment. A Wi-Fi network environment usually consists of two types of devices: access points (APs) and stations (STAs). A wireless AP allows other wireless devices to connect to and provides wireless connectivity to the network. For example, a wireless network base station at home or in a public area is a wireless AP. A public place that provides

wireless Internet access is also known as a Wi-Fi hotspot. A wireless STA is a device that connects to a wireless AP, generally referring to a product that can wirelessly access the Internet, such as a computer and a mobile phone. The following figure shows the relationship between a wireless AP and a wireless STA.

The mobile phone (a wireless STA) communicates with the router (a wireless AP) at a frequency of 2.4 GHz. Simply speaking, the user controls the data link layer because the choice to which AP to connect is determined by the user.

II. Wi-Fi and Bluetooth

Both Bluetooth and Wi-Fi are network communication technology. To determine which technology is to be used when making a robot, we need to take into account the functions of the robot, such as signal environment, signal strength and the need to penetrate walls. The following is a comparison table for Bluetooth and Wi-Fi.

	Bluetooth	Wi-Fi
Frequency	2.4 GHz	2.4 GHz, 5 GHz
Module cost	Low	High
Founded by	Bluetooth Special Interest Group (Bluetooth SIG)	Institute of Electrical and Electronics Engineers (IEEE)
General usage	Mobile phones, mice, keyboards, office and industrial automation equipment	Notebook computers, desktop computers, servers, televisions, smartphones
Hardware requirement	Bluetooth adapter	Wi-Fi adapter
Range	About 5-30 m	802.11: about 7 m maximum 802.11a: about 7-23 m 802.11b: about 46 m maximum
Energy consumption	Low (about 2 milliwatts of energy is required for 100 bits per second via Bluetooth)	High (about 80 milliwatts of energy is required for 100 bits per second via Wi-Fi)
Penetrating power	Low	High

III. ESP8266 Wi-Fi module and NodeMCU

Arduino has a Wi-Fi module, which uses an ESP8266 chip. In the previous section about data transmission control, we mentioned the relationship between wireless APs and wireless STAs. The ESP8266 Wi-Fi module operates as a wireless STA. Smartphones can connect to the ESP8266 module and the ESP8266 module can connect to wireless APs in schools or homes. The figure below shows the structure of the module.

The ceramic antenna on the ESP8266 Wi-Fi module is capable of transmitting and receiving radio waves and analysing signals. The metal box in the middle contains a microprocessor and 4 MB of flash memory, allowing us to store data and programs for controlling the module. In addition, the ESP8266 module has four pins. Pin 1 and pin 2 are TX pin and RX pin, respectively, which are responsible for transmitting and receiving signals through the antenna. The power and ground pins in the lower part are responsible for providing the power.

However, the ESP8266 Wi-Fi module has some limitations. For example, the module has very strict voltage control, which requires a 3.3V voltage. Excessive or insufficient voltage will cause the module to work improperly. However, most power supply units provide a 5V voltage, which makes it more difficult to use the module. Moreover, the module does not have additional pins for us to connect various input or output devices.

In view of this, we choose to use a module that has been modified by manufacturers. Some manufactures have added a power slot, a USB serial port chip and a processing core to the module, in order to make the module more suitable to be used as the control board for developing robots with specific software added. This control board is known as NodeMCU. Simply provide the power with a computer or a power supply unit through a USB cable, and then we can start developing robot applications. The figure below shows the control board and its structure.

After modification, the NodeMCU control board has 16 general-purpose input/output (GPIO) pins and an analogue input pin. When writing programs for the control board, we use the GPIO pin numbers, rather than the combinations of letter and number marked on the control board. Therefore, we need a detailed data sheet of the NodeMCU control board to assist robot programming. The NodeMCU control board also has a voltage regulator that accepts a 5V supply and a processor core. We can use the NodeMCU control board as an independent control board to control a robot through programming. The following figure shows the data sheet of the control board.

We can find the 16 GPIO pin numbers from the data sheet. The special feature of this control board is that each pin can have two to three functions. Taking pin D8 as an example, pin D8 is GPIO15, TXD2 and HCS. We need the GPIO pin numbers on the data sheet to define the pins when programming the robot.

IV. Do it yourself – Set up the NodeMCU

The first task in this chapter is very simple – we will use the NodeMCU control board to control the LED on it. Due to the complexity of the setup of the control board, students should pay attention to the details. First we need to download a driver that lets the computer connect to the control board and install the ESP8266 module library in the Arduino IDE.

1. Since the Wi-Fi module used here is a special one, we need to download a driver that lets the computer connect to the control board and communicate with it. We can download the CP210 driver via the hyperlink: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers. **Note: Some variations of NodeMCU may need other drivers, such as CH340 or CH341. Pay attention to the instructions at the back of the control board or the user guide, and download the corresponding drivers from the hyperlink: https://github.com/himalayanelixir/Arduino_USB_Drivers.

2. Connect the NodeMCU control board to the power supply.

3. Open Arduino IDE. Press Arduino and open the **Preferences** dialogue box.

4. In the Preferences dialogue box, enter the information: http://arduino.esp8266.com/stable/package_esp8266com_index.json next to **Additional Boards Manager URLs** and press OK.

5. Next we need to download the library to the Arduino IDE. Under **Sketch**, select **Include Library**, and then press **Manage Libraries**.

6. Download the library – **esp8266 by ESP8266 Community 2.3.0**.

7. In the **Tools** menu, look for the **Board** options. Press it and select **NodeMCU 1.0 (ESP-12E Module)**.

8. In the **Tools** menu, look for **Port** and select **/dev/cu.SLAB_USBtoUART** to download the channel for the program.

9. When students complete the above settings, it is basically ready to program with the NodeMCU control board. First we need to test whether the control board is working properly. The first program is the LED blink in the fourth chapter of the Basic Unit 1. In the blink program, we need to define the position of the LED in setup() and define the current strength in loop(). The following shows the blink program. When the input is complete, press Verify and upload it to the control board.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

10. If the program is uploaded successfully, you will see the following changes happening on the NodeMCU control board.

V. Do it yourself – Control the on and off of the LED on the control board through IP address connection

The second task of this chapter is relatively difficult. In this section, we are also controlling the on and off of the LED on the NodeMCU control board. However, this time, we need to register a Wi-Fi client for the NodeMCU control board, and also set up a Wi-Fi server with a specific IP address so that we can put the Wi-Fi client content on the link. By inputting a specific URL to send a signal, the NodeMCU control board will execute the command after receiving the signal via Wi-Fi. Therefore, when we complete the hardware connection and programming, we will control the NodeMCU board by entering different URLs in the browser. The following is the entire program. We will analyse and explain the program in detail step-by-step.

The complete program codes for controlling the LED through IP address connection. Refer to file 4_1_url_control_led.

```
#include <ESP8266WiFi.h>

const char* wifi_name = "";
const char* wifi_password = "";

WiFiServer server(80);
```

Include the library and define the variable

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(115200);
  Serial.print("Connecting to ");
  Serial.println(wifi_name);
  WiFi.begin(wifi_name, wifi_password);

  Serial.print("Loading");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi connected");

  server.begin();
  Serial.println("Server started");
  String local_ip = WiFi.localIP().toString();
  Serial.println("You can turn on builtin led by visiting" + local_ip + "/gpio/0");
  Serial.println("You can turn off builtin led by visiting" + local_ip + "/gpio/1");
  Serial.println();
}
```

Open the serial communication channel and connect to the Wi-Fi

```
void loop() {
  WiFiClient client = server.available();
  if(!client) {
    return;
  }
  Serial.println("new client");
  while(!client.available()) {
    delay(1);
  }

  String req = client.readStringUntil('\r');
  client.flush();
```

Register a Wi-Fi client for the NodeMCU control board and check if it is connected

```
  int val;
  String html;
  if(req.indexOf("/gpio/0") != -1) {
    val = 0;
    html = "<html><body><h1>Built-in LED is ON!</h1></body></html>";
    Serial.println("NodeMCU received an 'On' Signal");
    Serial.println();
  } else if(req.indexOf("/gpio/1") != -1) {
    val = 1;
    html = "<html><body><h1>Built-in LED is OFF!</h1></body></html>";
    Serial.println("NodeMCU received an 'Off' Signal");
    Serial.println();
  } else {
    Serial.println("Waiting for New Request\n");
    client.stop();
    return;
  }

  digitalWrite(LED_BUILTIN, val);
  client.flush();

  client.print(html);
  delay(1);
}
```

Set the series of actions when an URL is entered

Program content explanation

1. Include the library and define the variables

```
#include <ESP8266WiFi.h>

const char* wifi_name = "";
const char* wifi_password = "";

WiFiServer server(80);
```

In the first part, we include the ESP8266 library and define the Wi-Fi name and password. We can find the Wi-Fi name and password from the Wi-Fi settings of a communication device such as a smartphone, and then put them down in the double quotes of the variables wifi_name and wifi_password, respectively. Note that in the above program, the data type we use to define the Wi-Fi name and password is 'const char*'. When this data type is used, it means that we are defining **a variable whose content cannot be changed**. That is, although we cannot do anything to change the Wi-Fi name and password in the program, we can use the variables to execute the program, such as connecting to the Wi-Fi. The last line aims to turn on the Wi-Fi server and add the port 80.

2. Open the serial communication channel and connect to the Wi-Fi

```
8  void setup() {
9    pinMode(LED_BUILTIN, OUTPUT);
10
11   Serial.begin(115200);
12   Serial.print("Connecting to ");
13   Serial.println(wifi_name);
14   WiFi.begin(wifi_name, wifi_password);
15
16   Serial.print("Loading");
17   while (WiFi.status() != WL_CONNECTED) {
18     delay(500);
19     Serial.print(".");
20   }
21   Serial.println();
22   Serial.println("WiFi connected");
23
24   server.begin();
25   Serial.println("Server started");
26   String local_ip = WiFi.localIP().toString();
27   Serial.println("You can turn on builtin led by visiting" + local_ip + "/gpio/0");
28   Serial.println("You can turn off builtin led by visiting" + local_ip + "/gpio/1");
29   Serial.println();
30 }
```

In the second part, we define the tools we need in setup(), such as opening the serial communication channel and connecting to the Wi-Fi. The baud rate of the serial communication channel is set as 115200, which means that the NodeMCU control board can transmit 115,200 bits of data per second. Since we need to connect to the Wi-Fi and receive data from the IP address this time, we define a serial communication channel with a higher baud rate to increase the update rate. In the beginning, we define the LED pin on the control board. We open the serial communication channel in lines 11 to 14 and display 'Connecting to' Wi-Fi name there. The function 'Wi-Fi.begin()' is used in line 14 to connect to the Wi-Fi. We continue to print 'Loading' on the serial monitor window in lines 16 to 20. The while loop in the middle prints a period every 0.5 second when Wi-Fi is being connected. Finally, in lines 24 to 32, we start the Wi-Fi server and type the word 'WiFi.localIP()'. Since IP address connection is used to control the on and off of the LED in this task, we use the Wi-Fi library to extract the IP address and print it to the serial communication channel (serial monitor), so that we can acquire the link conveniently.

3. Register a Wi-Fi client for the NodeMCU control board, and check if it is connected and updated

```
void loop() {
  WiFiClient client = server.available();
  if(!client) {
    return;
  }
  Serial.println("new client");
  while(!client.available()) {
    delay(1);
  }

  String req = client.readStringUntil('\r');
  client.flush();
```

In the third part, we assume that the Wi-Fi server has started. Next, we need to register a Wi-Fi client for the NodeMCU control board. If the registration is successful, 'new client' will be printed to the serial communication channel. The meaning of the while loop is that if the client fails to register successfully, wait for 0.5 second. When we register the Wi-Fi client, we will get a string from the server, whose contents include the IP address, you are using HTTP, etc. We will get the last sentence of this string for use in programming later.

4. Set the series of actions when a specific URL is entered and change the content of the URL

```
  int val;
  String html;
  if(req.indexOf("/gpio/0") != -1) {
    val = 0;
    html = "<html><body><h1>Built-in LED is ON!</h1></body></html>";
    Serial.println("NodeMCU received an 'On' Signal");
    Serial.println();
  } else if(req.indexOf("/gpio/1") != -1) {
    val = 1;
    html = "<html><body><h1>Built-in LED is OFF!</h1></body></html>";
    Serial.println("NodeMCU received an 'Off' Signal");
    Serial.println();
  } else {
    Serial.println("Waiting for New Request\n");
    client.stop();
    return;
  }

  digitalWrite(LED_BUILTIN, val);
  client.flush();

  client.print(html);
  delay(1);
}
```

In the last part, we use the resulting string obtained from the third part to determine the URL. Then we use the specific URL to assign the appropriate content: control the LED and the HTML content of the link, and finally print the above series of actions to the serial communication channel. First, we define the variables val and html. Since HTTPClient uses the IP address of the computer to open a separate page to control the NodeMCU, the first half of the URL is the IP address. The second half of the URL defines the part of the NodeMCU that we will use. As we need to set one of the GPIOs as high, the URL is 'IP address/gpio/0'. The IP address is a number tag showing that the device is using the network. We can find our own IP address by going to www.howtofindmyipaddress.com.

When the IP address is found, we can input the URL in the browser to perform the checking and operation. If the URL entered is 'IP address/gpio/0', set val to 0 and html to 'Built-in LED is ON!'. The LED of the NodeMCU control board will light up and the webpage will show the corresponding html content. If the URL entered is 'IP address/gpio/1', set val to 1 and html to 'Built-in LED is OFF!'. The LED of the NodeMCU control board will not light up and the webpage will show the corresponding html content. Finally, update the client link constantly.

5. Upload the program to the NodeMCU control board and test the content when the programming part is completed. If the test is successful, you will get the following results.

When 'IP address/gpio/1' is input	
When 'IP address/gpio/0' is input	

VI. Review questions

Can you answer the following questions after completing this chapter?

1. What is Wi-Fi?
2. What standards has Wi-Fi used?
3. What do the Wi-Fi standards include?
4. Can you differentiate between a wireless access point (AP) and a wireless station (STA)?
5. Can you point out the differences between Wi-Fi and Bluetooth?
6. If we want the Arduino to use Wi-Fi, what module can we use?
7. What is the difference between the ESP8266 module and the NodeMCU module?
8. How can you set up the ESP8266 and the NodeMCU by yourself?
9. How can you turn on and off the LED of the control board through IP address connection?

Chapter 2: IFTTT Integrated Network Services Platform

With the practice in Chapter 1, we have officially entered the area to explore the Internet and use hyperlinks to control the modules. In this chapter, we will introduce a powerful network platform, IFTTT, which allows us to use different software to develop robots with even more functions. For instance, your robot can remind you to bring an umbrella when leaving home tomorrow because it knows that it is going to rain.

I. What is IFTTT?

IFTTT is the abbreviation of 'IF This Then That'. It provides a service for developing a condition of a task, i.e. if XXX performs the YYY behaviour, execute ZZZ. This is an integrated platform of network services that determines whether to execute the preset commands by going through the conditions of different platforms. The following figure shows the entire setup process. The user needs to create an 'Applet', and define the Trigger and the Action.

The following are some automation applications of IFTTT in daily life.

II. HTTP request methods and status codes

When using Wi-Fi, we often use HTTP requests. In programming, these request methods and the codes of the status at the time when the requests are received are very important. Through some request method and status code examples as a brief introduction in this section, we hope students can use this knowledge when writing programs for robots in the future.

Let's begin with the request methods. When making an HTTP request, it is equivalent to visiting a hyperlink. In this section, we will introduce the GET method and the POST method. Both the GET method and the POST method can visit a hyperlink and transfer data. The GET method is like sending a postcard. Anyone who sees the postcard on the way it is being sent can know its specific content. The POST method can also be used to transfer data, but it is transferred in the form of a letter. People can only see the envelope and the actual content can be obtained only when the hyperlink request is completed. To further explain, the GET method is to input an URL to make the HTTP request. When using this method to transfer data, we must include the name and value of the information in the URL. When we use the POST method, we can define an HTTP header that includes the name and value of the information, and make the HTTP request with both the header and the URL. When using IFTTT, since the IFTTT platform will look for information in the HTTP header, we need to use the POST method to transfer data.

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped into five classes: informational responses, successful responses, redirects, client errors and server errors. Due to the large number of status codes, they cannot be fully listed here. When necessary, students can visit https://developer.mozilla.org/en-US/docs/Web/HTTP/Status to get the complete list of status codes.

Information responses	100: This interim response indicates that everything so far is OK and that the client should continue with the request or ignore it if it is already finished.
Successful responses	200: The request has succeeded. The meaning of a success varies depending on the HTTP method: GET: The resource has been fetched and is transmitted in the message body. POST: The resource describing the result of the action is transmitted.
Redirects	301: This response code means that the URL of the requested resource has been changed. Probably, the new URL would be given in the response.

Client errors	400: This response means that server could not understand the request due to invalid syntax.
Server errors	500: The server has encountered a situation it does not know how to handle.

III. Do it yourself – Set up IFTTT

The first task of this chapter is to do some set-up at ifttt.com so as to use the service WebHooks to send yourself an email when an HTTP request is received. In this example, a Google account will be used to do the setting.

1. Go to www.ifttt.com to do the setting. Press Continue with Google.

2. Choose Use another account. Enter the email and press Next, and enter the password and press Next again.

3. Now we have logged in IFTTT. Press My Applets to do the setting.

4. Press New Applet.

5. Now we have created a new Applet. Press the word circled with the red box ('this').

6. First select the service. Enter 'WebHooks' in the input box and select the following WebHooks icon.

7. Press Connect.

8. Press Receive a web request.

9. Then we can set the trigger. First enter the event name (NodeMCU_Button). Note that we cannot add a space to the name because the same name will be used in the hyperlink later and a space will cause the hyperlink to fail to work. Press the button circled with the red box in step 2 when finish inputting.

10. In this way, we have completed the THIS part. Then we will set the THAT part. Press the word circled with the red box ('that').

11. Choose the action service. Enter 'Gmail' in the input box and select the following Gmail icon.

12. Press Connect.

13. Choose Use another account. Enter the email and press Next, and enter the password and press Next again. Finally press Allow.

14. Press Send yourself an email.

15. The following shows the settings of the subject and body of the email. The items highlighted in white are variables, including the event name, sending time, and values 1, 2, and 3, which have been defined previously. We just need to make some simple settings now and then we can press Create action without making any changes.

16. Press Finish.

17. The set-up is done. Press Search.

18. When programming, we need to find the codes that belong to our own account. Enter 'WebHooks' in the input box and select the WebHooks icon.

19. Press Documentation.

20. Webhooks will generate a unique key. Everyone has a different key. In the following program codes, you need to enter your unique key in wh_key. Now copy the URL in the second red circle to make an HTTP request.

21. With the completion of the above 20 steps, you have basically set up the IFTTT. In fact, there are a lot more different platforms, such as Google Drive, Evernote and Skype. Students can explore and create new Applets on their own.

IV. Execute the IFTTT function with a GET request

The third task of this chapter is relatively difficult. Although we are only using the IFTTT function to trigger an action with a GET request, we need to process the HTTP status code for the GET request. The following is the entire program. We will analyse and explain it in detail step-by-step in the following.

The complete program codes of the IFTTT GET request. Refer to file 4_2_1_ifttt_get_req,

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

const char* wif  name = "";
const char* wifi_password = "";

HTTPClient http_client;
```

Include the library and define the variables

```
void IFTTT() {
  http_client.begin("maker.ifttt.com/trigger/NodeMCU_Button/with/key/wh_key");
  int http_code = http_client.GET();
  if (http_code > 0) {
    Serial.println(http_code);
    if (http_code == 200) {
      String received_request_message = http_client.getString();
      Serial.println(received_request_message);
    }
  } else {
    Serial.println("HTTP connection ERROR!");
  }
}
```

Send a GET request with a hyperlink and extract the received request message

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(wifi_name, wifi_password);
  Serial.println("");
  Serial.print("Loading");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi is ready...");
  IFTTT();
}
```

Open the serial communication channel, connect to the Wi-Fi and execute the IFTTT function

```
void loop() {}
```

Program content explanation

1. Include the library and define the variables

```
1 #include <ESP8266WiFi.h>
2 #include <ESP8266HTTPClient.h>
3
4 const char* wifi_name = "";
5 const char* wifi_password = "";
6
7 HTTPClient http_client;
```

In the first part, we include the ESP8266WiFi library and the ESP8266HTTPClient library, and define the Wi-Fi name and password. The data type used to define the Wi-Fi name and password is 'const char*', meaning that we are defining **a variable whose content cannot be changed**. That is, although we cannot do anything to change the Wi-Fi name and password in the program, we can use the variables to execute the program, such as connecting to the Wi-Fi. The last line defines the HTTPClient so that we can make the HTTP request in the Arduino IDE.

2. Send a GET request with a hyperlink and extract the received request message

```
9  void IFTTT() {
10   http_client.begin("maker.ifttt.com/trigger/NodeMCU_Button/with/key/wh_key");
11   int http_code = http_client.GET();
12   if (http_code > 0) {
13     Serial.println(http_code);
14     if (http_code == 200) {
15       String received_request_message = http_client.getString();
16       Serial.println(received_request_message);
17     }
18   } else {
19     Serial.println("HTTP connection ERROR!");
20   }
21 }
```

In the second part, we send a GET request in IFTTT() and extract the received request message. In the beginning, we use the function 'http_client.begin()' to connect to the hyperlink. You need to find your own Webhooks key and change the wh_key in the URL of the function 'http_client.begin()' to your Webhooks key. In line 11, we use the function 'http_client.GET()' to send the GET request. In lines 12 to 18, the first If loop checks whether an HTTP status code is returned, and the second If loop checks whether the HTTP status code 200 is returned. When the request is successful, extract the received request message and show it to us.

3. Open the serial communication channel, connect to the Wi-Fi and execute the IFTTT function

```
23 void setup() {
24   Serial.begin(115200);
25   WiFi.begin(wifi_name, wifi_password);
26   Serial.println("");
27   Serial.print("Loading");
28   while (WiFi.status() != WL_CONNECTED) {
29     delay(500);
30     Serial.print(".");
31   }
32   Serial.println();
33   Serial.println("WiFi is ready...");
34   IFTTT();
35 }
```

In the third part, we define the required tools in setup(), for example, opening the serial communication channel and connecting to the Wi-Fi. In the beginning we define the pin for the LED of the control board. In line 27, we open the serial communication channel and display 'Loading' there. In line 25, we use the function 'WiFi.begin()' to connect to the Wi-Fi. The While loop in lines 28 to 31 prints a period every 0.5 second when Wi-Fi is being connected. Then in line 33, we open the serial communication channel and print 'Wi-Fi ready...'. Finally, execute the IFTTT function.

4. When the programming part is completed, we can upload the program to the NodeMCU control board and test the content. The following figure shows the result of a successful test. We will receive an email reminder on the mobile device, with the word 'NodeMCU_Button' included in the email. 'Congratulations! You've fired the NodeMCU_Button' will be shown on the serial communication channel. If the test is successful, you will see the following result on the serial communication channel and receive an email.

If there is a problem connecting to the Wi-Fi, the following result will be shown on the serial communication channel.

V. Execute the IFTTT function with a POST request

The fourth task of this chapter is also relatively difficult. Although we are only using the IFTTT function to trigger an action with a POST request, we need to process the JSON push for the POST request. In this task, we are going to push the data we define in the Arduino IDE with a JSON object. JSON is a file format used to store data that is widely used for network communication. In the program codes of this chapter, we need to transfer two pieces of data: the variable 'value1' that stores the value 22 and the variable 'value2' that stores the value 56. By defining the header as JSON format in the program, the browser will be able to identify the data. The following shows the result of putting the JSON formatted object in a browser.

The following is the entire program. We will analyse and explain it in detail step-by-step in the following.

The complete program codes of the IFTTT POST request. Refer to file 4_2_2_ifttt_post_req.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

const char* wifi_name = "";
const char* wifi_password = "";

const byte button_pin = 0;
const byte led_pin = 2;
unsigned long previous_millis = 0;
const long pressed_period = 60000;

HTTPClient http_client;
```

Include the library and define the variables

```
void IFTTT() {
  http_client.begin("maker.ifttt.com/trigger/NodeMCU_Button/with/key/wh_key");
  http_client.addHeader("Content-Type", "application/json");
  int http_code = http_client.POST("{\"value1\":22, \"value2\":56}");
  if (http_code > 0) {
    Serial.println(http_code);
    if (http_code == 200) {
      String payload = http_client.getString();
      Serial.println(payload);
    }
  } else {
    Serial.println("HTTP connection ERROR!");
  }
}
```

Send a POST request with a hyperlink and extract the received request message

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(wifi_name, wifi_password);
  Serial.print("Loading");
  Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi ready...");
}
```

Open the serial communication channel and connect to the Wi-Fi

```
void loop() {
  unsigned long current_millis = millis();
  boolean button_state = digitalRead(button_pin);
  if (button_state == LOW) {
    digitalWrite(led_pin, LOW);
    if ((previous_millis == 0) || (current_millis - previous_millis) >= pressed_period) {
      previous_millis = current_millis;
      IFTTT();
    }
  } else {
    digitalWrite(led_pin, HIGH);
  }
}
```

Detect the potential change of the button and determine whether or not to execute the IFTTT function

Program content explanation

1. Include the library and define the variables

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

const char* wifi_name = "";
const char* wifi_password = "";

const byte button_pin = 0;
const byte led_pin = 2;
unsigned long previous_millis = 0;
const long pressed_period = 60000;

HTTPClient http_client;
```

In the first part, we include the ESP8266WiFi library and the ESP8266HTTPClient library, and define the Wi-Fi name and password. The data type used to define the Wi-Fi name and password is 'const char*', meaning that we are defining **a variable whose content cannot be changed**. That is, although we cannot do anything to change the Wi-Fi name and password in the program, we can use the variables to execute the program, such as connecting to the Wi-Fi. In addition, we also define the pins for the button and the LED. We define the pin for the button as 0, which means that we are controlling the FLASH button on the NodeMCU control board. 'pressed_period' is a time value, and 'previous_millis' is zero, representing the status when the button is not pressed. The last line defines the HTTPClient so that we can make the HTTP request in the Arduino IDE.

2. Send a POST request with a hyperlink and extract the received request message

```
14 void IFTTT() {
15   http_client.begin("maker.ifttt.com/trigger/NodeMCU_Button/with/key/wh_key");
16   http_client.addHeader("Content-Type", "application/json");
17   int http_code = http_client.POST("{\"value1\":22, \"value2\":56}");
18   if (http_code > 0) {
19     Serial.println(http_code);
20     if (http_code == 200) {
21       String payload = http_client.getString();
22       Serial.println(payload);
23     }
24   } else {
25     Serial.println("HTTP connection ERROR!");
26   }
27 }
```

In the second part, we send a POST request in IFTTT() and extract the received request message. In the beginning, we use the function 'http_client.begin()' to connect to the hyperlink. In line 16, we add a header in JSON format for the Wi-Fi client request. In line 17, we use the function 'http_client.POST()' to send the POST request. In lines 18 to 25, the first If loop checks whether an HTTP status code is returned, and the second If loop checks whether the HTTP status code 200 is returned. When the request is successful, extract the received request message and show it to us.

3. Open the serial communication channel and connect to the Wi-Fi

```
29 void setup() {
30   Serial.begin(115200);
31   WiFi.begin(wifi_name, wifi_password);
32   Serial.print("Loading");
33   Serial.println("");
34   while (WiFi.status() != WL_CONNECTED) {
35     delay(500);
36     Serial.print(".");
37   }
38   Serial.println("WiFi ready...");
39 }
```

In the third part, we define the required tools in setup(), for example, opening the serial communication channel and connecting to the Wi-Fi. In line 32, we open the serial communication channel and display 'Loading' there. In line 31, we use the function 'WiFi.begin()' to connect to the Wi-Fi. The While loop in lines 34 to 37 prints a period every 0.5 second when Wi-Fi is being connected. Then in line 38, we open the serial communication channel and print 'Wi-Fi ready...'.

4. Detect the potential change of the button based on the time and execute the IFTTT function if it is detected that the button is pressed

```
41 void loop() {
42   unsigned long current_millis = millis();
43   boolean button_state = digitalRead(button_pin);
44   if (button_state == LOW) {
45     digitalWrite(led_pin, LOW);
46     if ((previous_millis == 0) || (current_millis - previous_millis) >= pressed_period) {
47       previous_millis = current_millis;
48       IFTTT();
49     }
50   } else {
51     digitalWrite(led_pin, HIGH);
52   }
53 }
```

In the fourth part, we define the variable 'current_millis' and the boolean 'button_state' to detect the potential change of the button in the beginning. In this case, we use the timer function of the IDE. At the beginning of the program, the timer will start to detect the potential change of the button. If the button is not pressed, the variable 'current_millis' will always be 0; but if the button is pressed, the variable 'previous_millis' defined in the first part will detect a potential change in the button and the variable 'current_millis' will be stored. The 'if' condition in the program will note that there is a change in the variable 'current_millis' and the IFTTT function will be executed.

5. When the programming part is completed, we can upload the program to the NodeMCU control board and test the content. The following figure shows the result of a successful test.

VI. Execute the IFTTT function using the DHT11 with a POST request

The final task of this chapter is more difficult than the previous ones. In this task, we are also using the IFTTT function to trigger an action with a POST request. However, in this task, we need to take the readings of the temperature and humidity sensor at the time we press the button, and send the POST request with a JSON push.

The following is the entire program. We will analyse and explain it in detail step-by-step in the following.

The complete program codes of the IFTTT POST request with the temperature and humidity readings. Refer to file 4_2_3_ifttt_post_dht11_value_req.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "DHT.h"

#define DHTPIN 1
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

const char* wifi_name = "";
const char* wifi_password = "";

const byte button_pin = 0;
const byte led_pin = 2;
unsigned long previous_millis = 0;
const long pressed_period = 60000;

HTTPClient http_client;

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(wifi_name, wifi_password);
  Serial.print("Loading");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi ready...");
}

void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  String str_temp = String(temperature);
  String str_humidity = String(humidity);

  unsigned long current_millis = millis();
  boolean button_state = digitalRead(button_pin);
  if (button_state == LOW) {
    digitalWrite(led_pin, LOW);
    if ((previous_millis == 0) || (current_millis - previous_millis) >= pressed_period) {
      previous_millis = current_millis;
      http_client.begin("maker.ifttt.com/trigger/NodeMCU_Button/with/key/wh_key");
      http_client.addHeader("Content-Type", "application/json");
      int http_code = http_client.POST("{\"value1\":" + str_temp +", \"value2\":" + str_humidity +"}");
      if (http_code > 0) {
        Serial.println(http_code);
        if (http_code == 200) {
          String payload = http_client.getString();
          Serial.println(payload);
        }
      } else {
        Serial.println("HTTP connection ERROR!");
      }
    }
  } else {
    digitalWrite(led_pin, HIGH);
  }
}
```

Include the library and define the variables

Open the serial communication channel and connect to the Wi-Fi

Take the readings of the temperature and humidity sensor

Detect the potential change of the button, determine whether or not to send the POST request with the hyperlink and extract the received request message

Program content explanation

1. Include the library and define the variables

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "DHT.h"

#define DHTPIN 1
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

const char* wifi_name = "";
const char* wifi_password = "";

const byte button_pin = 0;
const byte led_pin = 2;
unsigned long previous_millis = 0;
const long pressed_period = 60000;

HTTPClient http_client;
```

In the first part, we include the ESP8266WiFi library and the ESP8266HTTPClient library, and define the Wi-Fi name and password. '#define' is another syntax for defining constants. As the part that needs to be defined this time is the string of 'DHT11', it is more convenient and easier to understand by using '#define'. Taking the example of defining the 'led_pin' as pin 13, the syntax is usually 'int led_pin = 13;'. If '#define' is to be used, the syntax will become '#define led_pin 13'. Note that when using the '#define' syntax, the # symbol must appear in front of 'define' and no semicolon should be added. Therefore, the codes above mean that 'DHTPIN' is defined as pin 1 and 'DHTTYPE' is defined as 'DHT11'. Moreover, we also set the pins for the button and the LED. 'pressed_period' is a time value, and 'previous_millis' is zero, representing the status when the button is not pressed. The last line defines the HTTPClient so that we can make the HTTP request in the Arduino IDE.

2. Open the serial communication channel, turn on the temperature and humidity sensor and connect to the Wi-Fi

```
19 void setup() {
20   Serial.begin(115200);
21   dht.begin();
22   WiFi.begin(wifi_name, wifi_password);
23   Serial.print("Loading");
24   while (WiFi.status() != WL_CONNECTED) {
25     delay(500);
26     Serial.print(".");
27   }
28   Serial.println("WiFi ready...");
29 }
```

In the second part, we define the required tools in setup(), for example, opening the serial communication channel, turning on the temperature and humidity sensor and connecting to the Wi-Fi. In line 21, we turn on the temperature and humidity sensor. In line 23, we open the serial communication channel and display 'Loading' there. In line 22, we use the function 'WiFi.begin()' to connect to the Wi-Fi. The While loop in lines 24 to 27 prints a period every 0.5 second when Wi-Fi is being connected. Then in line 28, we open the serial communication channel and print 'Wi-Fi ready...'.

3. Begin the loop and take the temperature and humidity sensor readings

```
31 void loop() {
32   float humidity = dht.readHumidity();
33   float temperature = dht.readTemperature();
34   if (isnan(humidity) || isnan(temperature)) {
35     Serial.println("Failed to read from DHT sensor!");
36     return;
37   }
38   String str_temp = String(temperature);
39   String str_humidity = String(humidity);
```

In the third part, we take the temperature and humidity sensor readings in loop(). If the sensor cannot get any readings, the serial communication channel will print 'Failed to read from DHT sensor!'; if readings are obtained, the temperature and humidity readings will be converted into strings, facilitating the data transfer in the following part of the program.

4. Detect the potential change of the button based on the time

```
41   unsigned long current_millis = millis();
42   boolean button_state = digitalRead(button_pin);
43   if (button_state == LOW) {
44     digitalWrite(led_pin, LOW);
45     if ((previous_millis == 0) || (current_millis - previous_millis) >= pressed_period) {
46       previous_millis = current_millis;
```

In the fourth part, we define the variable 'current_millis' and the boolean 'button_state' to detect the potential change of the button.

5. Send a POST request with a hyperlink

```
47       http_client.begin("maker.ifttt.com/trigger/NodeMCU_Button/with/key/wh_key");
48       http_client.addHeader("Content-Type", "application/json");
49       int http_code = http_client.POST("{\"value1\":" + str_temp +", \"value2\":" + str_humidity +"}");
```

In the fifth part, we send a POST request in IFTTT(). In the beginning, we use the function 'http_client.begin()' to connect to the hyperlink. In line 48, we add a header in JSON format for the Wi-Fi client request. In line 49, we use the function 'http_client.POST()' to send the POST request.

6. Extract the received request message

```
50       if (http_code > 0) {
51         Serial.println(http_code);
52         if (http_code == 200) {
53           String payload = http_client.getString();
54           Serial.println(payload);
55         }
56       } else {
57         Serial.println("HTTP connection ERROR!");
58       }
59     }
```

In the sixth part, the first If loop checks whether an HTTP status code is returned, and the second If loop checks whether the HTTP status code 200 is returned. When the request is successful, extract the received request message and show it to us.

7. Debug

```
60   } else {
61     digitalWrite(led_pin, HIGH);
62   }
63 }
```

In the seventh part, we set the potential of the button as high, i.e. light up the LED when the button is not pressed, making it easier for us to observe the result. Finally, end the loop.

8. When the programming part is completed, we can upload the program to the NodeMCU control board and test the content. The following figure shows the result of a successful test.

VII. Review questions

Can you answer the following questions after completing this chapter?

1. What is IFTTT?
2. What is an HTTP request method and what is an HTTP status code?
3. Can you give some examples of HTTP request methods?
4. Can you give some examples of HTTP status codes?
5. How can you set up IFTTT yourself?
6. How can you make a GET request with IFTTT?
7. How can you make a POST request with IFTTT?
8. How can you send sensor readings with IFTTT?

Chapter 3: Unit Project – Elderly Care Robot

As a family member, we should care about the needs of the elderly. As we grow older, our body immunity will weaken and the chance of getting sick will increase. We need to always keep an eye on the status of the elderly. However, while adults need to go to work and students need to go to school, it seems difficult to always monitor the status. Therefore, we would like to try to solve the problem from the perspective of a robot.

This project is relatively difficult. Although we are only using ThingSpeak to make a GET request to trigger an action, we need to process the linkage of data for the GET request.

Stage / step of engineering design	Relevant knowledge	Investigation and design considerations	Applying the relevant knowledge and investigation results on the engineering design
Define the problem (confirm the requirements and limitations)	Understand the needs of the elderly and take care of them with love, making it possible for them to contact us whenever necessary.		Clearly set the design requirements and limitations
Research	· Performance and transmission distance of various wireless connection · What tasks need to be completed when caring for the elderly · What sensors can learn about the situation of the environment where the elderly are in · What platform		Choose a handling method (if conducted in class, one or more options can be chosen, depending on the teaching objectives, lesson time and material restrictions)
Design consideration	Research data show that Wi-Fi transmission has the highest efficiency when operating in long distances		Test the operation of the sensor and the robot with simple programs first
Test the model	Connect the NodeMCU with ThingSpeak and add the DHT11 to send the readings of the sensor to the Internet according to the environment of the user instantly. Family members can browse the custom ThingSpeak channel and note the change of the environment around the elderly.	How to effectively speed up the receiving of the needs of the elderly	
		Fair test: Find the fastest way for a robot to access the Internet · Independent variable: various ways · Dependent variable: length of time	

		• Control variables: venue, sensing environment, voltage of the battery, etc.	
Solve problems encountered during the design / production / testing processes		The program must allow the action robot to successfully send the valid information (sensor readings) to their family members via email	Take these findings as the necessary procedures of this engineering design
Analyse and evaluate test results and problems occurred		Using sensors and programs to allow the robot to send the valid information (sensor readings) to their family members via email	Based on the analysis and evaluation results, find out ways to improve the design
Improve	In addition to installing a single temperature and humidity sensor, multiple different sensors can be installed to help the robots act quickly and effectively	Improve the efficiency of the robot with an inertial measurement unit (IMU)	Explore together how an elderly care robot is made today with the current technology

1 Preparing the circuit of the robot

1. First, connect the temperature and humidity sensor to the NodeMCU.	2. Find a suitable material for the wristband for mounting the electronic components

2 What is ThingSpeak

ThingSpeak is a platform that can be used to set up Internet of Things. The principle is to use the GET method of HTTP request to add the data name and value to the URL, and transfer the data to the unique channel of the user. The ThingSpeak platform supports real-time data updates and will generate a line graph of the data received based on the average update time, allowing users to draw data trends from the graphs.

1. Open the browser and enter the URL: www.thingspeak.com.

2. Press Channels on the top of the webpage.

3. Enter the name of the graph and the measurement units of graph 1 and graph 2 in Name of graph, Graph 1 and Graph 2, respectively. When completed, check the boxes next to Graph 1 and Graph 2.

4. Press API Keys in Channels.

5. Copy your unique 'Write API Key' for use in the programming part.

3 Program content explanation

The complete program codes of the elderly care robot. Refer to file 4_3_elderly_caring_robot.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include "DHT.h"

#define DHTPIN 1
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

Include the library and define the variables

```
const char* wifi_name = "";
const char* wifi_password = "";

HTTPClient http_client;

void setup() {
  Serial.begin(115200);
  WiFi.begin(wifi_name, wifi_password);
  dht.begin();
  Serial.println("Serial monitor testing");
  while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
  }
  Serial.println("WiFi ready");
}
```

Start the DHT11 sensor, open the serial communication channel and connect to the Wi-Fi

```
void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT11 sensor");
    return;
  }
  String strTemp = String(temperature);
  String strHumidity = String(humidity);
```

Use the DHT11 sensor to take the readings and convert them into strings

```
  http_client.begin("api.thingspeak.com/update?api_key=[redacted]&field1=" + strTemp + "&field2=" + strHumidity);
  int http_code = http_client.GET();
  if (http_code > 0) {
    Serial.println(http_code);
    if (http_code == 200) {
      String payload = http_client.getString();
      Serial.println(payload);
    }
  } else {
    Serial.println("HTTP connection error");
  }
  http_client.end();
```

Send ThingSpeak API link

```
  delay(10000);
}
```

Take the readings and send the link every 10 seconds

Program content explanation

1. Include the library and define the variables

```
1  #include <ESP8266WiFi.h>
2  #include <ESP8266HTTPClient.h>
3  #include "DHT.h"
4
5  #define DHTPIN 1
6  #define DHTTYPE DHT11
7  DHT dht(DHTPIN, DHTTYPE);
8
9  const char* wifi_name = "";
10 const char* wifi_password = "";
11
12 HTTPClient http_client;
```

In the first part, we include the ESP8266WiFi library, the ESP8266HTTPClient library and the DHT library, and define the Wi-Fi name and password. The data type used to define the Wi-Fi name and password is 'const char*', meaning that we are defining a variable whose content cannot be changed. That is, although we cannot do anything to change the Wi-Fi name and password in the program, we can use the variables to execute the program, such as connecting to the Wi-Fi. '#define' is another syntax for defining constants. As the part that needs to be defined this time is the string of 'DHT11', it is more convenient and easier to understand by using '#define'. Therefore, the codes above mean that 'DHTPIN' is defined as pin 1 and 'DHTTYPE' is defined as 'DHT11'. The last line defines the HTTPClient so that we can make the HTTP request in the Arduino IDE.

2. Start the temperature and humidity sensor, open the serial communication channel and connect to the Wi-Fi

```
14 void setup() {
15   Serial.begin(115200);
16   WiFi.begin(wifi_name, wifi_password);
17   dht.begin();
18   Serial.println("Serial monitor testing");
19   while (WiFi.status() != WL_CONNECTED) {
20   delay(500);
21   Serial.print(".");
22   }
23   Serial.println("WiFi ready");
24 }
```

In the second part, we define a serial communication channel with a baud rate of 115200 and start the DHT11 temperature and humidity sensor in setup(). In the beginning, we use the function 'Serial.begin()' to define the serial communication channel. In line 17, we use the function 'dht.begin()' to initiate the DHT11 temperature and humidity sensor. In line 15, we open the serial communication channel and display 'Serial monitor testing' in line 18. In line 16, we use the words 'WiFi.begin(wifi_name, wifi_password);' to connect to the Wi-Fi.

3. Begin the loop and take the readings of the temperature and humidity sensor

```
void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT11 sensor");
    return;
  }
  String strTemp = String(temperature);
  String strHumidity = String(humidity);
```

In the third part, we use the temperature and humidity sensor to get the readings in loop(). In the If conditional statement, the function 'isnan()' will return 'true' if the variable is empty. So the whole line means that if the temperature and humidity sensor does not have a reading, print 'Failed to read from DHT11 sensor' on the serial communication channel. If there is a reading, convert the temperature and humidity readings into strings, facilitating the data transfer in the following part of the program.

4. Transfer the data to ThingSpeak.

```
  http_client.begin("api.thingspeak.com/update?api_key=  &field1=" + strTemp + "&field2=" + strHumidity);
  int http_code = http_client.GET();
  if (http_code > 0) {
    Serial.println(http_code);
    if (http_code == 200) {
      String payload = http_client.getString();
      Serial.println(payload);
    }
  } else {
    Serial.println("HTTP connection error");
  }
  http_client.end();
```

Enter your API Key here

In the fourth part, we define the HTTPClient and send the data to a specific URL. We use the function 'http.begin()' to send the URL. You need to enter your API Key in the yellow box in the codes. As with the previous codes for making HTTP requests, we need to use the If conditional statement to determine what HTTP status code should be printed to the serial communication channel. If the HTTP status code is 200, the serial communication channel shows the message that the ThingSpeak request is successful; if the HTTP status code is anything other than 200, such as 500, print 'HTTP connection error' and the HTTP status code to the serial communication channel, so as to help us debug later.

5. Update the data automatically

```
  delay(10000);
}
```

In the fifth part, we use the function 'delay(10000)' for the program to update the readings of the temperature and humidity sensor and send the URL every 10 seconds.

6. When the programming is completed, we can upload the program to the NodeMCU and go to the ThingSpeak website to see the results.