TECHNISCHE HOCHSCHULE LÜBECK

Bachelor thesis

Setting up 3D Printer including Raspberry Pi Bridge

Submitted by: Yunxuan Zhu

Department: Electrical Engineering and Computer Science

Degree program: Information Technology

First examiner: Prof. Dr. Pelka

Date handing out: 01st March 2023
Date handing in: 01st June 2023

A. Hanemann

(Prof. Dr. Andreas Hanemann)
Head of Examination Board

Description:

3D printers are important devices in a smart factory. They allow printing of various components using low cost filaments. In a smart factory, such 3D printers print customized goods (e.g. shape, color) for the customer.

In this thesis you will set up two 3D printers and mount them onto a mobile table. They will be connected to a Raspberry Pi and a display which will provide a web interface to control the printers. You will create an that web interface to queue jobs to the printers and load different printing jobs to the printer. In order to mount the printer to the mobile table, you may use the 3D printer to manufacture parts. To display data, you wire a 10" industrial display to the Raspberry Pi.

Tasks

- Set up Prusa 3D printer.
- Mount a 3D printer onto a mobile table.
- Wire the 3D printer and mount the 3D printer towards the mobile table.
- Hook up the Raspberry Pi to control the printer.
- Install a Webcam to monitor the progress.
- Create software to control the 3D printer using a web interface and show it on the display.
- Specify an data protocol, based on MQTT, to transfer status data from one computer to another.

Results

- When done, we are able to control the 3D printer from a web interface running on the Raspberry Pi.
- It is possible to connect your Smart Phone to a web application to visualize the state of the 3D printer.
- Document all results in the Thesis.

TECHNISCHE HOCHSCHULE LÜBECK

Department of
Electrical Engineering and Computer Science
Dean's Office - Board of Examiners

Statement on the bachelor thesis

I assure you that I wrote the work independently, without outside help.

Only the indicated sources has been used in the preparation of the work.
The text or the sense of the text are marked as such.

I agree that my work may be published, in particular that the work may be submitted to third parties for inspection or that copies of the work may be made for forwarding to third parties.

24.05.2023
Date

Yunxuan Zhu
Signature

Zusammenfassung der Arbeit
Abstract of Thesis

Fachbereich:
Department: **Electrical Engineering and Computer Science**

Studiengang:
University course: **Information Technology**

Thema:
Subject: **Setting up 3D Printer including Raspberry Pi Bridge**

Zusammenfassung:
Abstract:

This thesis focuses on the development of a 3D printer system with remote functions for smart factories. The system utilizes Raspberry Pi and Message Queuing Telemetry Transport (MQTT) for real-time monitoring, control, and job scheduling of 3D printers, enhancing efficiency and productivity in smart factory environments. The thesis explores related technologies, identifies limitations in existing solutions, and proposes a new system structure. Hardware components such as the 3D printer, Raspberry Pi, camera, and MQTT broker are introduced, along with the operating systems and the software structure for information transfer. The evaluation of the solution highlights results obtained from the MQTT broker as well as other devices and the functionalities of the remote controller. Challenges including slow file transmission speed, video transmission on MQTT broker, poor usability and cross-platform file transfer are discussed in order to present future work opportunities.

Verfasser:
Author: **Yunxuan Zhu**

Betreuender Professor/in:
Attending Professor: **Prof. Dr. Mathias Pelka**

WS / SS: **SS <2023>**

Table of Contents

1. Introduction

3D printers play a crucial role in smart factories as they enable the production of diverse components using low cost filaments. These printers are particularly valuable in manufacturing customized goods, allowing for personalized items in terms of shape, color, and other specifications. Predictive maintenance is also considered to be significant in smart factories. Shubham has developed a system of a 3D printer with remote functions based on Raspberry Pi, including monitoring and the detection of material jamming or unavailability [1]. However, this sort of system can hardly meet the requirements of smart factories because smart factories need to use standard interfaces, such as Message Queuing Telemetry Transport(MQTT), to make all devices take part in the Internet of things (IoT). Being offline or using different protocols will both lead to difficulties in data communication.

1.1. Thesis structure

In this thesis, a 3D printer including the Raspberry Pi Bridge is set up in the smart factory laboratory. Chapter 2 discusses the related work in the field, exploring technologies such as Raspberry Pi, 3D printers, Smart factory, and MQTT. The limitations of current solutions are also examined to explain the meaning of the project. Chapter 3 presents the proposed solution, detailing the overall structure of the system. It firstly includes a breakdown of the different layers in Raspberry Pi and the MQTT broker. The hardware components, such as the 3D printer, Raspberry Pi, camera, and MQTT broker, are then described. After that, the operating systems utilized are explained, namely OctoPi in Raspberry Pi and Ubuntu in MQTT broker. Additionally, the software structure is outlined, covering the software in Raspberry Pi, MQTT broker, and the information transfer process. Other works, such as the automation program, self-starting program, remote controller, and documentation, are discussed in the end. Chapter 4 focuses on the evaluation of the solution. It presents the results obtained in the MQTT broker and shows the display of data in devices like Google Glass Enterprise 2 and Vive Pro 2. Furthermore, it explains the functionalities of the remote file transfer and control capabilities of the 3D printer. The chapter also highlights remaining challenges, including slow file transmission speed, the "send-start" problem, video stream transmission function, and cross-platform transfer of G-code files. Chapter 5 concludes the thesis by summarizing the findings and providing an outlook for future work in the field.

1.2. Methodology and Contributions

The project roughly consists of a Raspberry Pi, a 3D printer, and an MQTT broker. A program written in Raspberry Pi gets data from the 3D printer in real time. After the acquired data is integrated and processed, the program will use MQTT protocol to specify a topic and

publish the processed information to the MQTT broker. The data that conforms to a specific format will be automatically written into influxDB to facilitate the management of data in the smart factory laboratory.

Similarly, the MQTT broker can also control the 3D printer by using the remote controller program. The entire system not only guarantees real-time monitoring of the 3D printer's status by other devices in the smart factory but also provides users with the ability to control the printer's status and task scheduling through specific programs, either directly or indirectly.

2. Related work

2.1. Raspberry Pi

The Raspberry Pi is a microcomputer with the size of a card. Because it is a computer, it supports an external display, mouse and keyboard. It also supports Bluetooth, WiFi and other protocols to communicate with other devices. In addition, like other embedded devices, it can also be connected to a variety of sensors and actuators to control different devices. Figure 2.1 shows a picture of a Raspberry Pi.

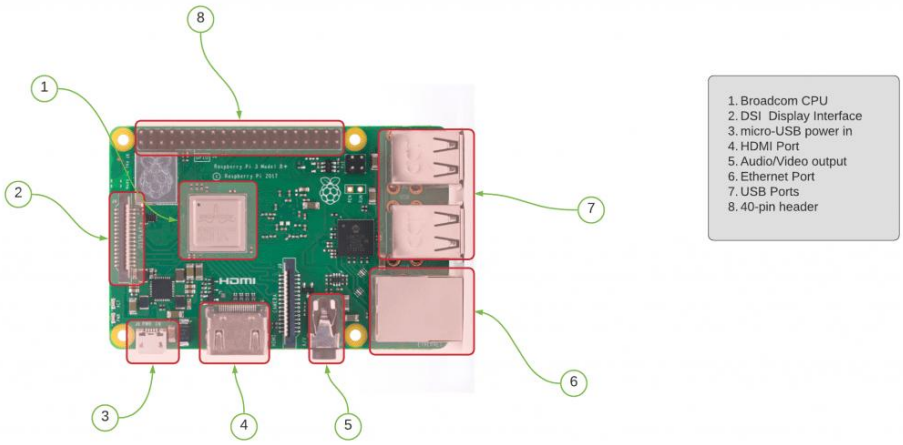

Figure 2.1: A picture of a Raspberry Pi

(https://www.circuitbasics.com/introduction-to-the-raspberry-Pi/)

Raspberry Pi is often used as the core control board in some IoT-based projects. For example, Lima et al. have used Raspberry Pi Zero W to make an edge-computing gateway, which uses MQTT communication protocol [2]. After testing, the performance of Raspberry Pi zero W is stable, and the temperature performance is very robust, which proves the feasibility of Raspberry Pi as the core control board in IoT-based projects. Due to the strong expansibility of Raspberry Pi, with the help of such as accessories such as a touch screen, a Global System for Mobile Communications (GSM) module, storage cards, a fully functional and flexible smartphone can also be designed [3]. With the help of an RFID module, Raspberry Pi with MQTT protocol can also be used to make a Digital Outing System[4]. The system has enough network security on the basis of low cost and simple structure. It reduces the workload on the security guards to a large extent. For actual industrial application, Raspberry Pi is often used with other development boards to perform data measurement of industrial components. For example, Mohammed et al. have developed a system based on Arduino and Raspberry Pi for industrial measurements [5]. The system can monitor and manually control the speed of the

DC motor with very high accuracy and send the obtained data to a ThingSpeak network website page. This system can play a very important role in actual industrial systems, such as conveyor lines.

2.2. 3D Printer

3D printing is a relatively new technology compared with traditional machining technology, which allows users to transform virtual modeling files in the computer directly into physical objects [6]. In general, the method used by conventional 3D printers is the Fused deposition modeling (FDM) method [7], which is less expensive and does not require high-precision equipment. Only low-cost material, such as PLA and ABS, is needed to be supplied from a large spool to the machine. This filament passes through a heated nozzle that moves and deposits the material onto the object being printed. The movement of the nozzle is controlled by a microcomputer, allowing it to define the desired shape. Typically, the nozzle moves horizontally to deposit one layer at a time. After each layer is completed, the nozzle will move vertically by a small distance to begin the next layer. Additionally, the speed of the nozzle can be adjusted to control the printing process. In the process of printing, material waste occasionally occurs because of printing failure. Alvarado-Diaz et al. designed a system that could crush, extrude and heat those waste materials, serving them again as new material [8]. In order to lower the cost and improve portability, an L-shaped 3D printer was designed by Syed Bacha et al. [9]. Figure 2.2 shows the structure of the L-shaped 3D printer. The nozzle of the 3D printer is horizontally positioned and controlled by the x-axis, which allows it to move left or right through the utilization of a timing pulley. The y-axis employs a gear motor mechanism to rotate the printing baseplate. Vertical movement of the hot end is accomplished by the z-axis motor, which operates in conjunction with a threaded rod.

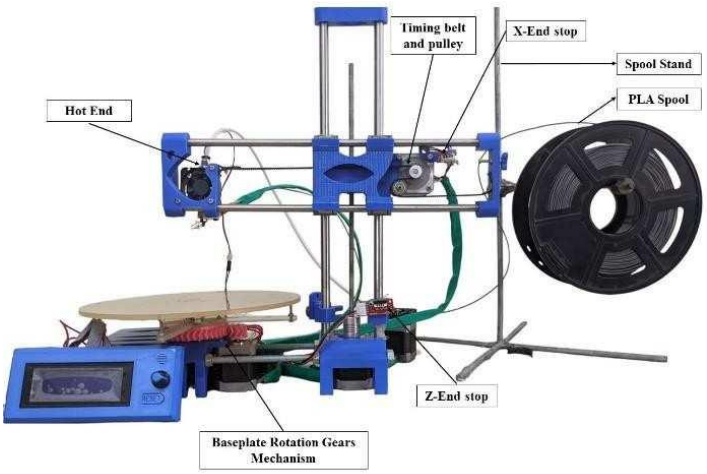

Figure 2.2: The structure of the L-shaped 3D printer [9]

2.3. Smart factory

With the advent of Industry 4.0, smart factories are playing an increasingly significant role in industrial production. Elvis defined the smart factory as a manufacturing solution capable of providing flexible and adaptive production [10]. Mohammed et al. have pointed out that the principles of designing a smart factory design are modularity, interoperability, decentralization, virtualization, service orientation, and real-time capability [11]. Taking the robotic delivery system as an example, the physical components of robots in a smart factory should be designed modularily, and the work groups composed of different robots should also be designed modularily to facilitate personalized customization of services and maintenance. All sensor data from robots should have a unified transmission protocol to facilitate information exchange across devices. In order to deal with complex and changeable situations, each robot should have the ability to make decisions on its own, rather than only obeying the control unit. For sudden and drastic changes in the environment, all individuals as well as the system as a whole should detect the changes quickly and recover in time. Through virtual reality and augmented reality technology, the administrator can observe the situation in the smart factory in real time, and if there is a failure, the administrator can remotely guide the maintenance work. Figure 2.3 shows the different layers in a smart factory. The adaptive and modular design of physical layer and data layer ensures the stability of the system. The remote management and intelligent decision-making function of the intelligence and control layer ensures the supervision of the lower layer, which maximizes the efficiency of the factory production.

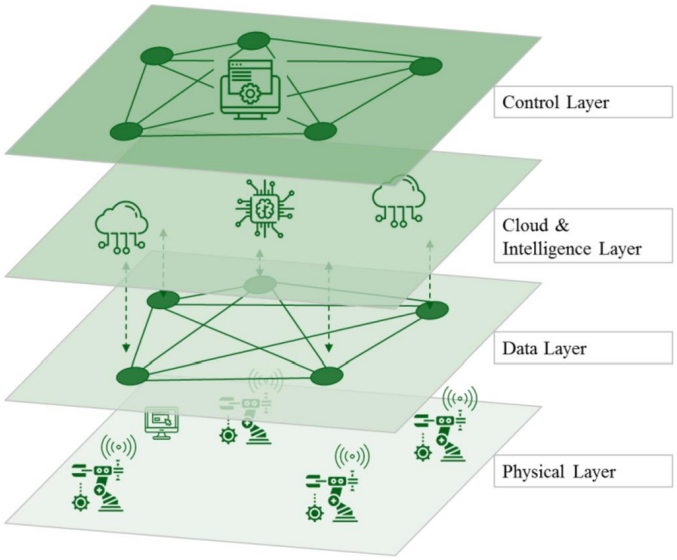

Figure 2.3: Different layers in Smart Factory[12]

2.4. MQTT

MQTT protocol is a lightweight transport protocol. It uses subscription and publishing as the transmission mechanism, which has very high effectiveness and accuracy for the information transmission of small amount of data. Therefore, it is particularly suitable for the Internet of Things, especially those devices with low bandwidth and high latency [13]. Through a specific topic on the MQTT broker, all users or devices that subscribe to this topic will receive information published on this topic from other devices. Figure 2.4 shows the publish/subscribe process utilized by MQTT. Compared with HTTP protocol, MQTT protocol has the advantages of high transmission efficiency and less protocol overhead. The disadvantage is the inability to transfer large amounts of data at the same time or files.

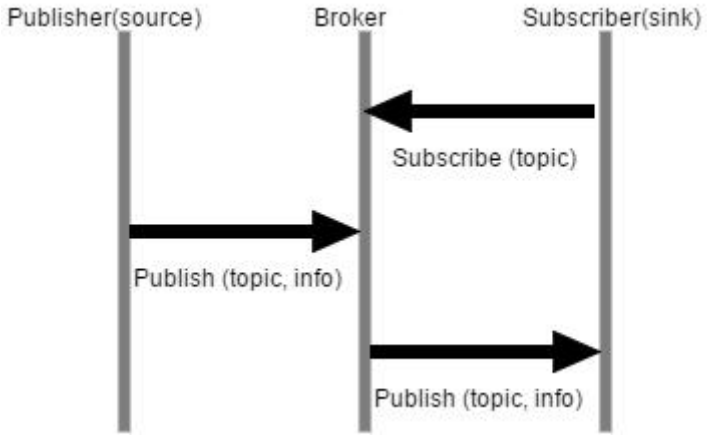

Figure 2.4: Publish/subscribe process utilized by MQTT [14]

2.5. Limitations of the current solutions

In summary, various projects have been undertaken to address different aspects of smart factory requirements, including offline monitoring and control of 3D printers, recycling of materials for environmental sustainability, redesigning 3D printers for enhanced portability, and implementing access control using Raspberry Pi and MQTT protocol. While these projects individually fulfill certain aspects of smart factory criteria such as interoperability and virtualization, none of them effectively integrate the information from 3D printers into the unified management database of the smart factory. This integration is crucial for achieving unified management and control within the smart factory management system. Therefore, the project is developed with the aim of resolving this issue.

3. Solution

3.1. Overall structure

3.1.1. Different layers in Raspberry Pi

As the core controller of the whole project, Raspberry Pi can be roughly divided into the following 4 layers: Hardware layer, operating system layer, software layer, and Application Programming Interface (API) layer. Figure 3.1 shows the layers in Raspberry Pi vividly. As the hardware of the controller, Raspberry Pi 400 has the advantages of high integration, strong compatibility, highly portable, which is very suitable as the central controller. OctoPi is the operating system for this project. The operating system is based on Linux system secondary development. It inherited all the functions of the traditional Linux system, and added some functions and features specifically designed for 3D printers. Octoprint, a software system that runs on top of this, is responsible for monitoring the status of the 3D printer and scheduling tasks. It is a user interface that users can control directly on their machine offline. In addition, Octoprint provides API to send real-time data and status of the 3d printer in a specific format. It is convenient for the code writing and MQTT transmission.

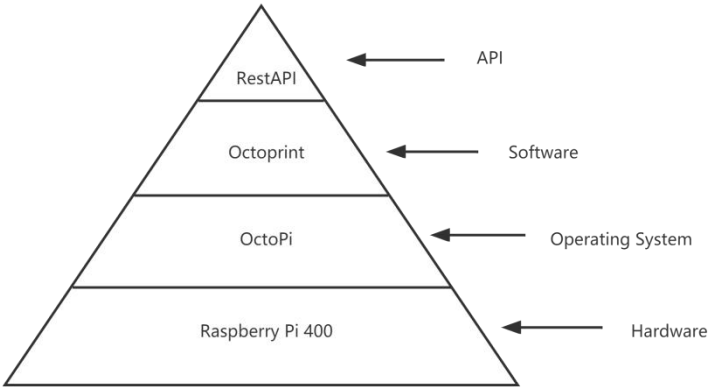

Figure 3.1: Layers in Raspberry Pi

3.1.2. Different layers in MQTT broker

The MQTT broker is generally the heart of an IoT project communicate over the MQTT protocol. As we all know, there are often thousands of devices in a smart factory, and if they speak the same protocol stack, then the requirements for the central processing server are relatively strict. In this project, the MQTT broker can be roughly divided into the following layers: the hardware layer, the operating system layer, the software layer, and the data display layer (database). Figure 3.2 illustrates the layers in MQTT broker. Due to the requirements of the intelligent factory for real-time data interaction, the traditional embedded hardware, such

as STM32, STC89C52, Raspberry Pi and other devices does not satisfy the condition. Therefore, the hardware we use in this project is FUJITSU CLIENT COMPUTING LIMITED ESPRIMO D7012. For the operating system layer, we use Ubuntu operating system, which is also a Linux operating system in essence, with strong compatibility and developer-friendly characteristics. At the software layer, we use the MQTT explorer, which can receive information from different devices in real time for conveying to other receiving devices or storing in a database. On this basis, if the data conforms to a certain format, it will be automatically entered into the database InfluxDB. Under the premise of storing these data, it can dynamically display them in the form of charts, so that the status of each device can be monitored remotely.

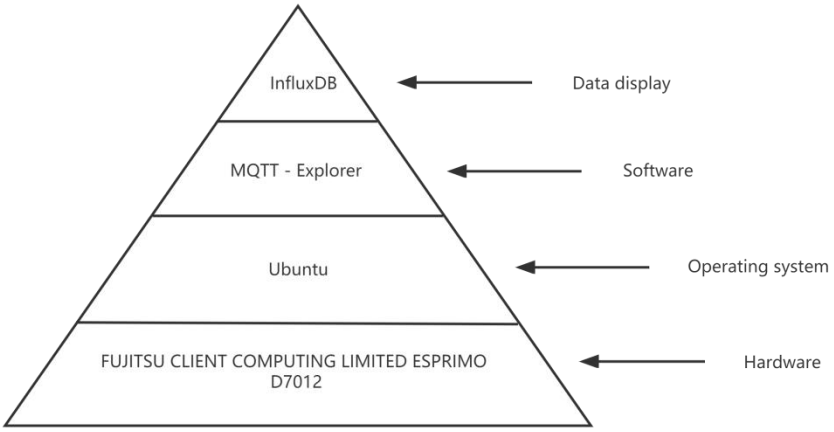

Figure 3.2: Layers in MQTT explorer

3.2. Hardware

3.2.1. Overall structure of hardware

The hardware part of the project consists of a Raspberry Pi, a 10-inch touchable industrial display, a 3D printer, a camera, and an MQTT broker. Figure 3.3 shows the overall structure of hardware and connection. Two cables are needed to connect the display and the Raspberry Pi. One is a HDMI to micro-HDMI cable, and the other is a USB-B 3.0 to USB-A cable. The first cable ensures the normal display of the screen, and the second cable ensures the usage of the touch function. A USB-A to USB-B cable is needed to connect the 3D printer and the Raspberry Pi. The camera comes with a USB-A cable for transmitting video streams. MQTT protocol connects Raspberry Pi and the broker wirelessly. Besides, different power cords are also needed to supply power to the 3D printer, Raspberry Pi, and industrial display. Figure 3.4 illustrates the hardware of the project in the lab. Figure 3.5 shows the MQTT broker.

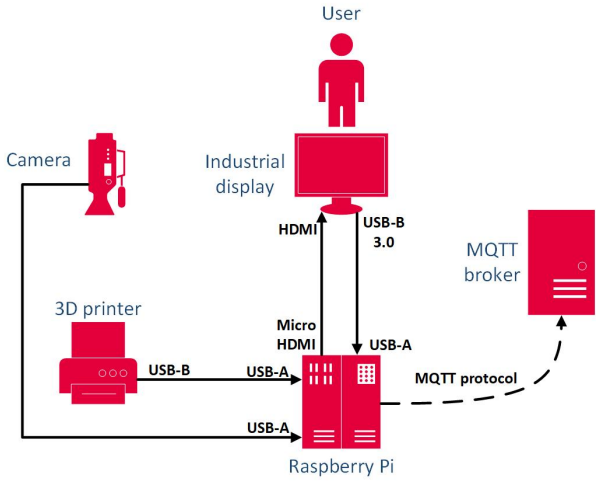

Figure 3.3: Overall structure of hardware

Figure 3.4: Hardware of the project in the lab

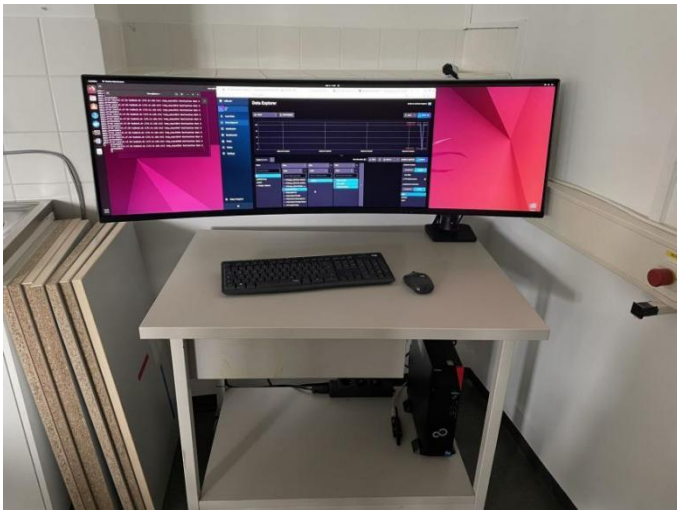

Figure 3.5: MQTT broker

3.2.2. 3D printer

3.2.2.1. General information

The model of the 3D printer in this project is Prusa i3 MK3S+. This 3D printer comprises a printing nozzle, a heating bed, a LCD display for managing tasks offline and monitoring, a spool used for filament supply, several fans for heat dissipation, and a USB interface for transmitting data. Figure 3.6 shows the printing nozzle and heating bed of Prusa i3 MK3S+. Figure 3.7 shows the filament of the printer. Figure 3.8 shows the LCD display.

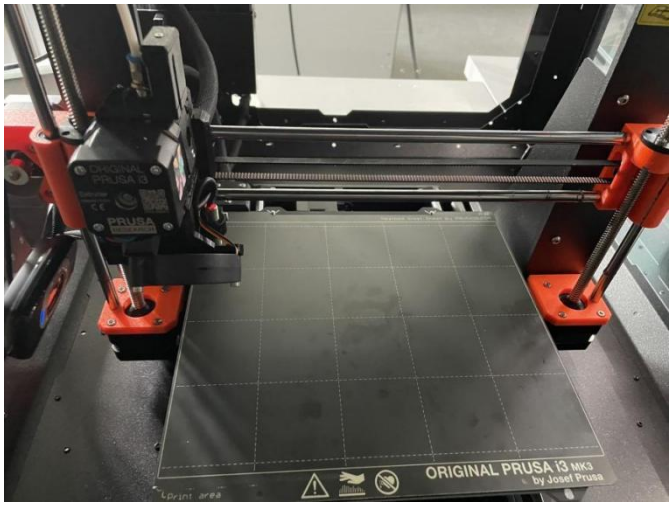

Figure 3.6: Printing nozzle and heating bed of Prusa i3 MK3S+

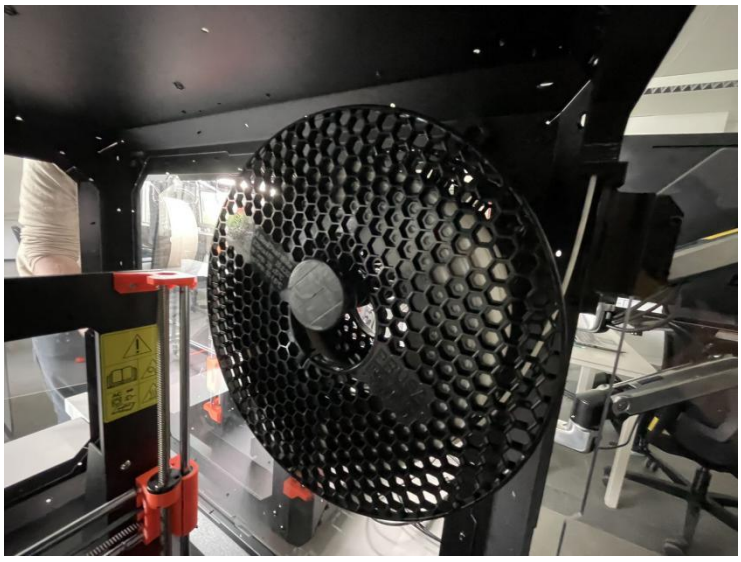

Figure 3.7: Filament of Prusa i3 MK3S+

Figure 3.8: LCD display on Prusa i3 MK3S+

3.2.2.2. Standard steps for using a 3d printer

In order to complete a printing task, especially without external monitoring software, users have to follow a series of operations. Figure 3.9 shows the standardized workflow for 3D printing. Users should first use a 3D modeling software, such as Autodesk Fusion 360, to create a 3D model file, such as an STL file, corresponding to the target object. After that, the generated file is supposed to be imported into a slicing software, such as Prusa Slicer, to generate a specific file called G-code file. A G-code file represents the movement of the nozzle during a specific printing task. Ultimately, the G-code file is written to an SD card, and the SD card will be inserted into the 3D printer. Users can start the task by using the display on the printer. After the printing task is selected for execution, the 3D printer will be warmed up. The default value of preheating is 215 degrees for the nozzle and 60 degrees for the work bed. When both of them are preheated to the preset value, the 3D printer will automatically enter the calibrating stage. During the calibrating process, the nozzle will sample a total of 9 points, which are the top right corner, the top middle point, the top left corner, the right middle point, the center point, the left middle point, the bottom right corner, the bottom middle point, and the bottom left corner. These nine points determine the final movement range of the nozzle. The machine will start the printing task according to the selected G-code file when both the warm-up and calibrating stages have completed. A standard cube whose side length is 1 centimeter takes about 13 to 15 minutes to print.

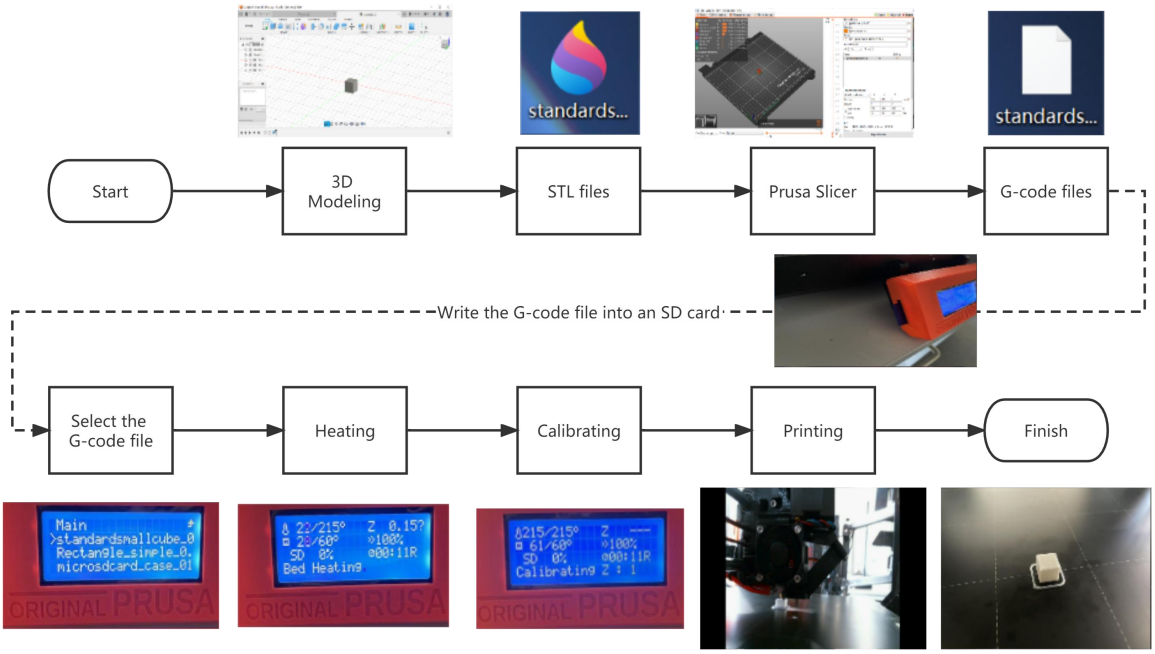

Figure 3.9: Standardized workflow for 3D printing

3.2.3. Raspberry Pi

In order to get information from the 3D printer, a Raspberry Pi 400 is used in this project. Raspberry Pi has the characteristics of super portable, powerful functions and strong compatibility. Different from the regular models of Raspberry Pi, Raspberry Pi 400 has been completed redesigned. The main board and the keyboard are integrated and it has an on/off button. It has a 1.8GHz ARM Cortex-A72 CPU and a 4GB LPDDR4-3200 RAM. Besides, it has a 40 Pin GPIO interface which can be used to implement some specific functions of sensors and actuators, a micro SD card slot for burning operating systems, dual micro HDMI ports for 4K resolution display output, a Type-C power port, dual USB 3.0 ports providing faster transmission rate, a USB2.0 port, and a Gigabit Ethernet port ensuring the fluency of the network. Figure 3.10 shows the Raspberry Pi 400 in the lab.

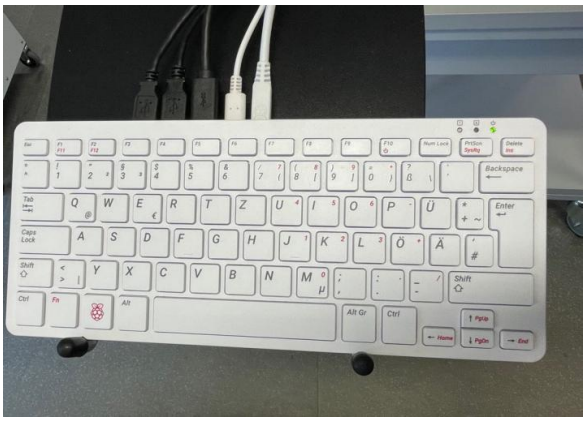

Figure 3.10: Raspberry Pi 400

3.2.4. Camera

In order to monitor the status of the 3D printer directly and dynamically, a Logitech C615 Mobile Webcam is installed nearby the nozzle. It supports USB2.0 and can capture up to 1080p videos. With a 74-degree FOV(Field of View), it can meet the needs of supervised shooting for 3D printing tasks. Besides, in order to install the camera, the 3D model of a camera mount was downloaded, sliced, printed and installed. It allows the camera to follow the movement of the nozzle, which effectively avoids occlusion. Figure 3.11 shows the Logitech C615 Mobile Webcam installed nearby the nozzle with the help of a camera mount.

Figure 3.11: Logitech C615 Mobile Webcam

3.2.5. MQTT broker

Due to the MQTT protocol, a broker is needed to handle all the subscriptions and publications from devices. In this project, FUJITSU CLIENT COMPUTING LIMITED ESPRIMO D7012 plays the role of broker. This device has 16GB of RAM, a processor of 12th Gen Intel Core™ i7-12700 x 20, a Graphics processor of Mesa Intel UHD Graphics 770 (ADL-S GT1), and a disk capacity of 512GB. As a PC host, its performance is much stronger than other embedded devices such as Raspberry Pi. So it can meet the hardware requirements of an MQTT broker. Figure 3.12 shows the hardware of MQTT broker.

Figure 3.12: MQTT broker

3.3. Operating systems

3.3.1. OctoPi in Raspberry Pi

OctoPi is essentially based on the Linux operating system. The biggest difference between Linux and Windows is that Linux usually uses commands to control the behavior of the operating system. For example, 'ls' will list all the files in the current path, and 'cd' will navigate to a specific folder. With these commands, programmers can be more convenient to add, delete, modify, or query data. Besides, the Linux-based operating system also makes it easier to code. It only needs to compile the written code using specific commands, then it's ready to execute, which has the characteristics of lightweight and high efficiency. Beyond that, OctoPrint can detect the port number used by the 3D printer, read and analyze all the data including the video stream of the camera via USB from the 3D printer, and finally transmit all the data to its corresponding integration software: OctoPrint.

To be able to successfully burn this operating system into the raspberry Pi 400, Raspberry Pi Imager is used as the auxiliary software. Figure 3.13 shows OctoPi in the Raspberry Pi Imager. After successfully writing the operating system and booting up, it is found that the version number is 1.0.0 and the native OctoPi has no GUI. In order to facilitate the visual display of the data on the industrial screen, GUI needs to be downloaded. This is done by first turning on the WiFi service to ensure that the Raspberry Pi is connected to the Internet, then entering the following command in the terminal: 'sudo /home/pi/scripts/install-desktop'. After the

installation, reboot the Raspberry Pi and the OctoPi GUI will open successfully. Figure 3.14 shows the desktop of OctoPi.

But this GUI has a serious drawback. It is very slow. In practice, the GUI refresh rate on the industrial display was less than 30 frames per second and was insensitive to touch. Using the 'top' command, the current top CPU hogging processes on Linux popped up and it was found that one of them, called mutter, was using 50% of the CPU. This process is the default window manager of this operating system, which takes up a large amount of resources. If the window manager was changed from mutter to openbox, the refresh rate and sensitivity would be greatly increased, so that it could be used normally.

Figure 3.13: Burn the OctoPi operating system into SD card with the Raspberry Pi Imager

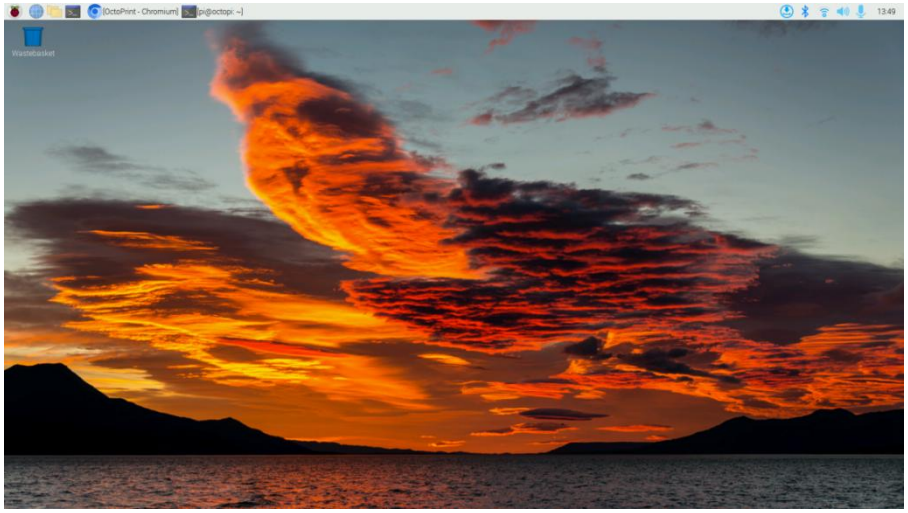

Figure 3.14: Desktop of OctoPi

3.3.2. Ubuntu in MQTT broker

Similarly, the Ubuntu operating system is a Debian-based Linux operating system, which is free, easy to install, easy to use, and has a strong community for maintenance. It is a mature operating system, including a full set of multimedia application software tools, Bluetooth, network application tools, printer drivers, and built-in Linux terminal server function. Thus, it is very suitable to install in a server or a broker. It runs more stable than Windows operating system thanks to its strong compatibility. In addition, it has a user-friendly GUI, which means whether developers or users can carry out barrier-free use. The version of Ubuntu in this project is 22.04.2 LTS, 64-bit. Figure 3.15 shows the desktop of Ubuntu.

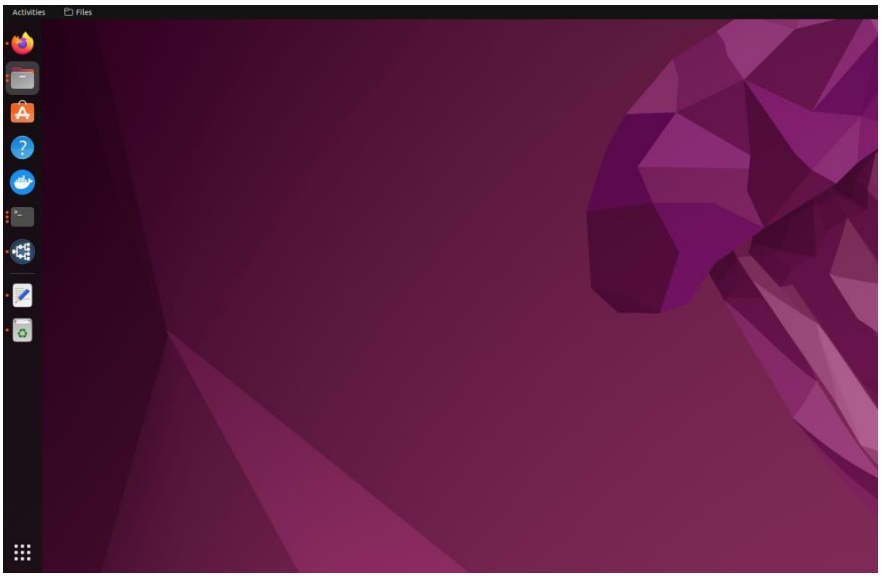

Figure 3.15: Desktop of Ubuntu

3.4. Software

3.4.1. Overall structure of software

Figure 3.16 illustrates the overall structure of software. The Raspberry Pi gets data from the 3D printer in real time through HTTP GET requests. Conversely, Raspberry Pi can also send relevant commands to the 3D printer by using HTTP POST requests. The camera sends its video stream data directly to the Raspberry Pi. When the acquired data is integrated and processed in the Raspberry Pi, the Raspberry Pi will use 'mosquitto_pub' command to specify a specific topic through the MQTT protocol and send the processed information to the MQTT broker. Similarly, the MQTT broker can also control the Raspberry Pi and the 3D printer through HTTP POST requests. On the other hand, other devices in the smart factory can obtain the data in the specific topic published by Raspberry Pi in real time by using the

command 'mosquitto_sub'. This whole system not only ensures that other devices in the smart factory can monitor the status of the 3D printer in real time, but also gives users the opportunity to control the status and task scheduling of the 3D printer by using specific programs directly or indirectly.

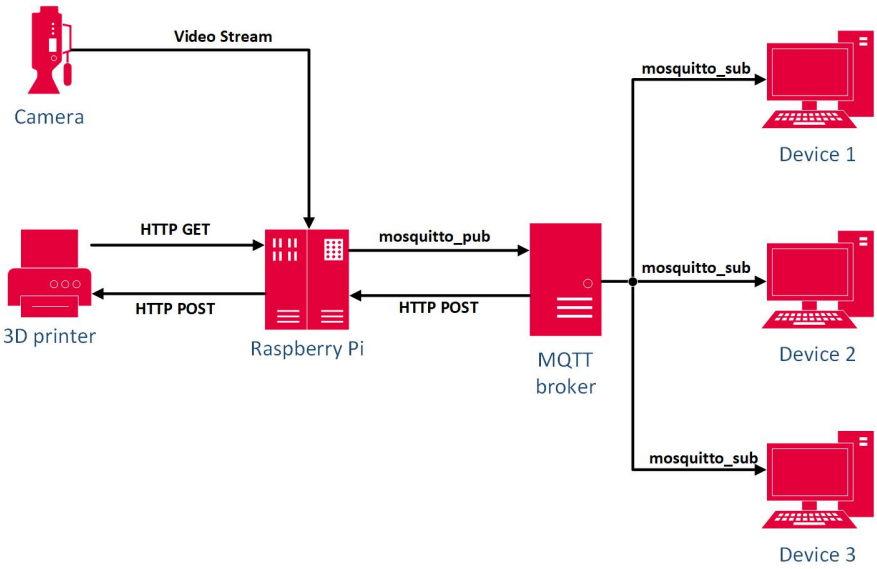

Figure 3.16: Overall structure of software

3.4.2. Software in Raspberry Pi

3.4.2.1. OctoPrint

OctoPrint runs on top of the OctoPi operating system. The version of OctoPrint is 1.8.7 in this project. If the 3D printer and the camera is connected correctly, enter the local IP address: 127.0.0.1 in the browser and then the GUI of OctoPrint pops up. The GUI of OctoPrint can be divided into the following sections. The top column contains settings, switch buttons, notifications, and user information. The status of the current 3D printer connection is located on the upper left side, including the printing progress, the file name of the current printing job, and the pause and cancellation of the printing job. All the 3D printed files in the SD card are on the lower left side. The user can create folders to manage all the files, or download the files from the SD card and add them to the printing job sequence. The right side of the interface is composed of multiple components, including temperature panel interface, control interface, terminal and visual printing interface. In the temperature panel interface, user can intuitively see the preset temperature value and the actual temperature value of the nozzle and the bed. They can be modified it in this interface. Figure 3.17 shows the GUI of OctoPrint with temperature panel interface on the right side. In the control interface, the user can see the real-time monitoring video stream of the printer, and below the monitoring interface there are buttons to control the movement of the bed, as well as buttons to control the nozzle moving up and down. The user can change how far the corresponding widget moves with each click of

the button by changing the value below. In addition, there is a switch to control the cooling fan and some other functions. Figure 3.18 shows the GUI of OctoPrint with control interface on the right side.

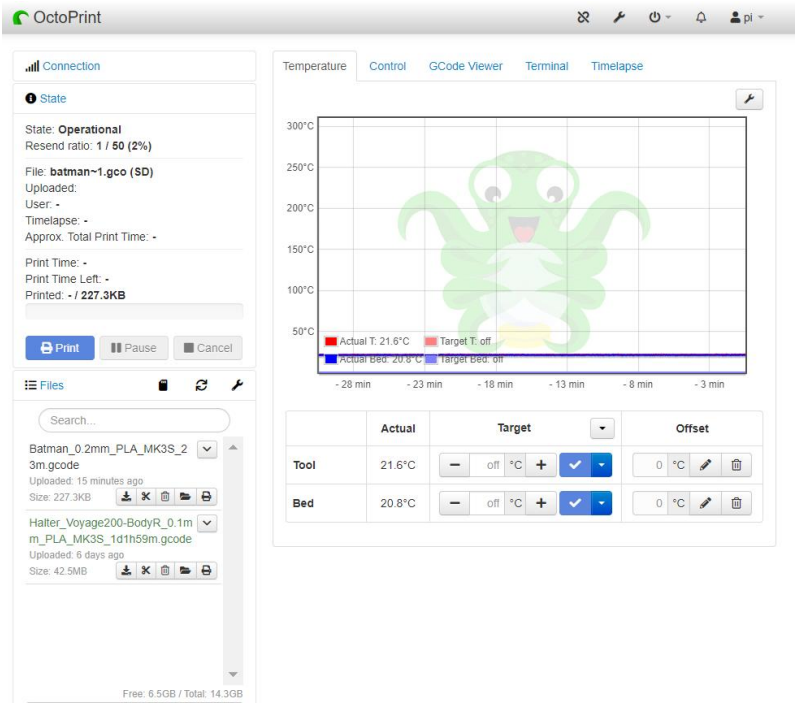

Figure 3.17: GUI of OctoPrint with temperature panel interface on the right side

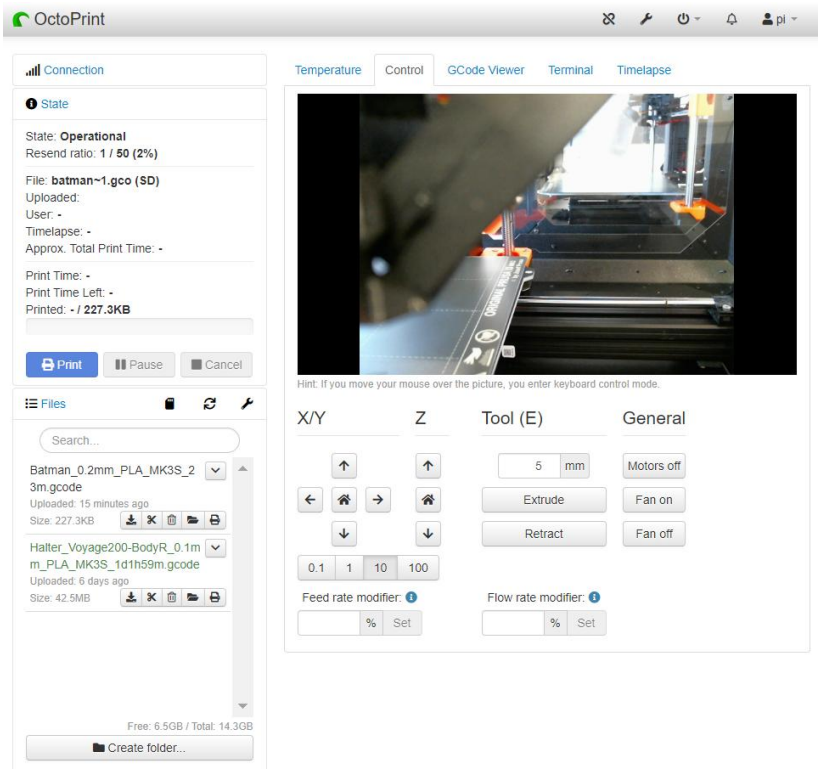

Figure 3.18: GUI of OctoPrint with control interface on the right side

3.4.2.2. REST API

In order to facilitate developers to carry out secondary development, OctoPrint also provides a series of APIs. An API key is required if those APIs is to be used. Figure 3.19 shows how to generate an API key. The user needs to enter a username and an application identifier. OctoPrint will automatically generate an API key. The way to use this API key is to attach the API key value in the HTTP requests to gain access to the internal data or control of the 3D printer.

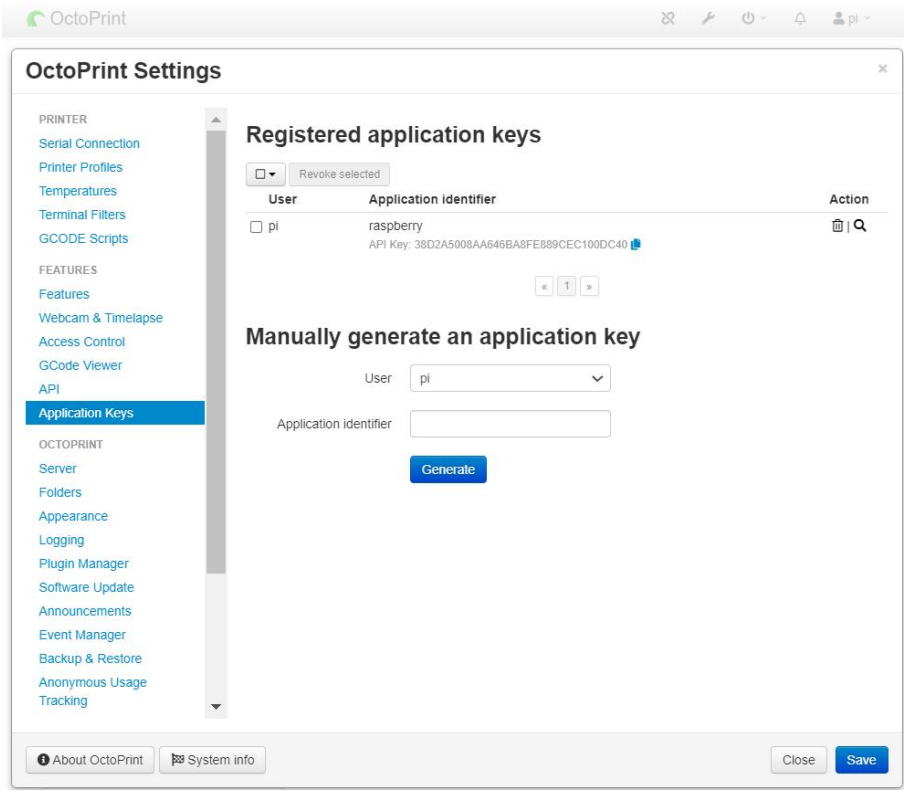

Figure 3.19: Generate an application key

After generating the API key, user needs to know about the REST API. REST API is an architectural style of API designed over HTTP. Users can obtain more information about the API on this website: https://docs.octoprint.org/en/master/API/index.html. In this project, the information of 3D printer is obtained through HTTP GET request, and the obtained results are returned as JSON files. This scheme provides great flexibility and compatibility, and facilitates the subsequent reading and processing of data. Figure 3.20 shows the list of REST API in OctoPrint. Figure 3.21 shows the API for retrieving all printer profiles. It is simply referred to as the following command: 'GET /api/printer'.

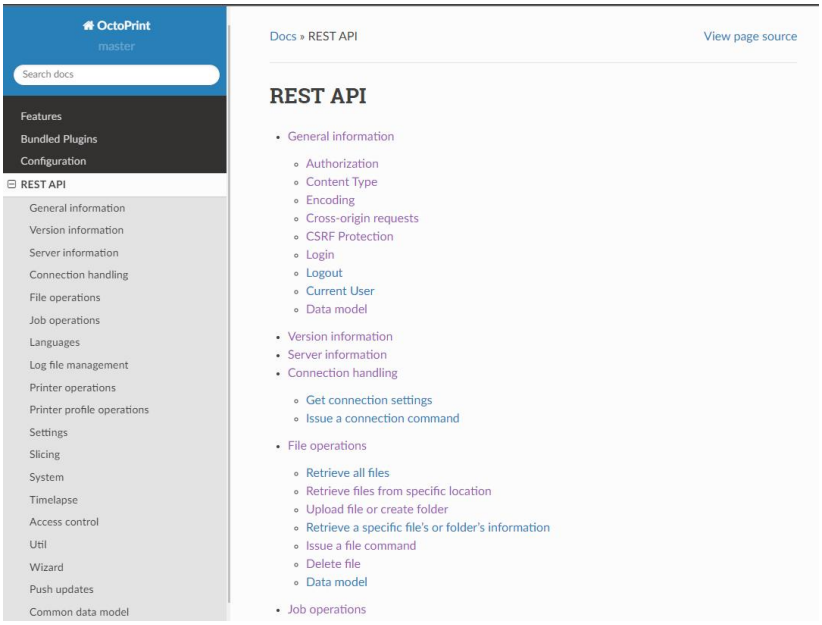

Figure 3.20: REST API list

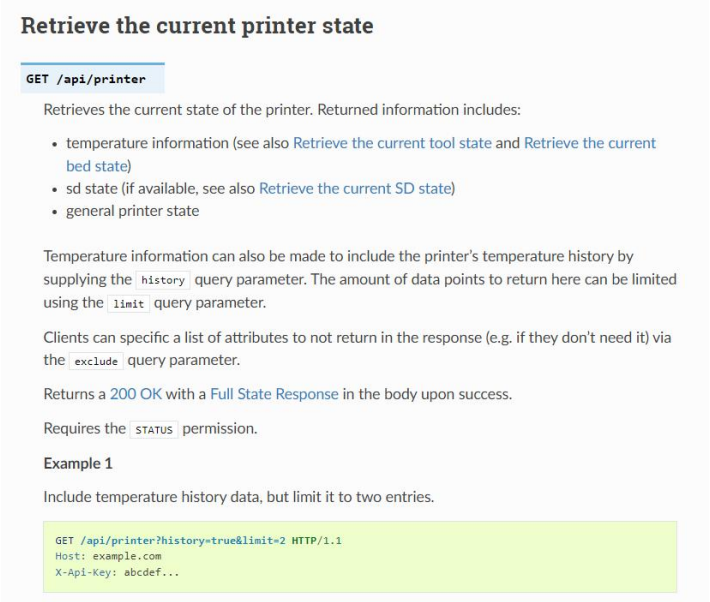

Figure 3.21: API for retrieving the current printer state

To be able to execute an HTTP request in Raspberry Pi, curl is introduced. Curl is a common command-line tool used to make requests to an external server. Here's an example of using curl command in Figure 3.22. -X specifies the type of the request, -H specifies the content type to be JSON. 127.0.0.1 represents the target server address, in this case, the local machine. /api/printer is the path provided by the REST API, and apikey represents the API Key generated earlier. If this command is run in the terminal, a JSON string will pop up, as shown in Figure 3.23.

```
curl -X GET -H 'Content-Type':'application/JSON' "127.0.0.1/api/printer?apikey=38D2A5008AA646BA8FE889CEC100DC40"
```

Figure 3.22: Example of a curl command

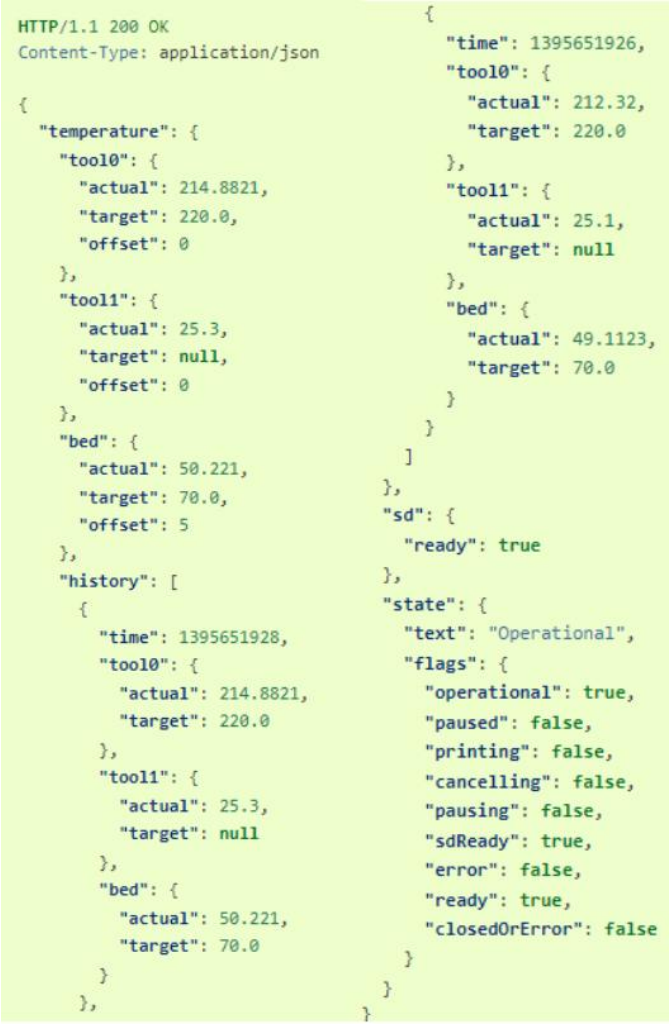

Figure 3.23: Result in JSON format

3.4.2.3. JSON to line protocol converter

After receiving the string of a JSON file, it is difficult to process the JSON file itself further. So the attribute values need to be stored as strings in order to be processed and used. In addition, since this project subsequently requires all data to be written into influxDB, the data format needs to comply with line protocol. There are some Apps that can do this, like Telegraf. But after the actual test, Telegraph is extremely demanding on the environment configuration, and the use of external software will cause the whole system to become redundant. So in the end, a program using Jansson library and C language is written to solve this problem. Figure 3.24 shows the flowchart of the program.

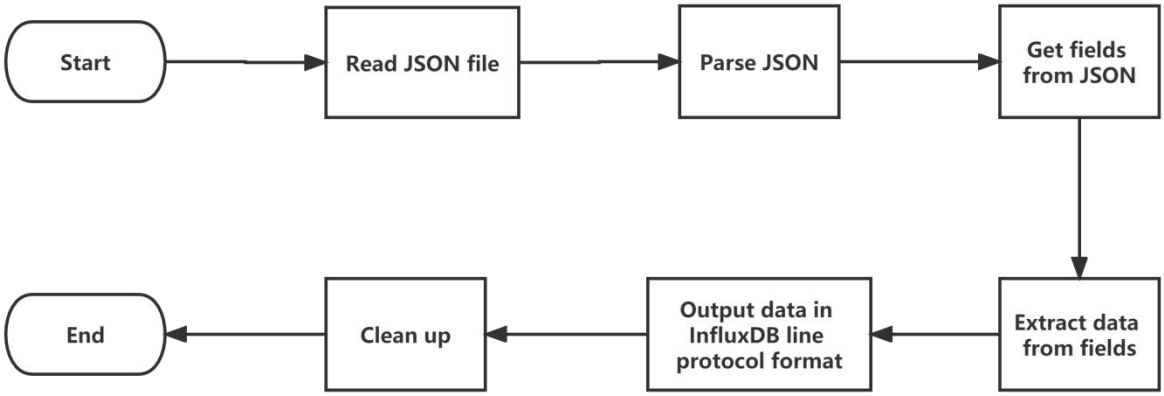

Figure 3.24: Flowchart of converter

First, the program reads the data from the JSON file and stores it as a string. Second, it calls the json_loads function, which parses the string into a root node that Jansson library can understand. Then, it calls the json_object_get function. This function takes two inputs, the root node as the first input and the property name from which we want to retrieve as the second input. It returns a child node that can be parsed by Jansson library. For example, for the JSON file in Figure 3.23, we can use these three strings: ‘sd’, ‘state’ and ‘temperature’ as the second input of the function. After the first retrieval, we obtain three child nodes, namely the sd node, state node and temperature node. Under the sd node, we need to get attribute value of the ready node. So for the second retrieval, we need to call the json_object_get function again, placing the sd node as the first input and the ‘ready’ string as the second input, which returns the ready node. To get the boolean value of the ready node, we need to call the json_boolean_value function. Finally, the return value is written to an integer variable. The same principle holds for all other values, except that the final node property value is obtained using json_string_value for strings and json_real_value for floating-point numbers.

Once all the values are obtained, we need to convert them to line protocol for writing to influxDB and visualizing the data on the dashboard of influxDB. Figure 3.25 shows the format of line protocol. The first variable is measurement, which is the name of the measurement. In this project, it is set to a, which means it is printer a in the lab. The second variable is tag set, which is used to declare the name of the variable being measured so far, such as name=state_text. The third variable is field set, which represents the measurement itself, such as value=24.90000, and the last variable is timestamp, which is not required. So it is not specified in this project, and instead the system time is used.

```
myMeasurement,tag1=value1,tag2=value2 fieldKey="fieldValue" 1556813561098000000
```

Figure 3.25: Format of line protocol

Since measurement and tag set are constant, it is only necessary to put the obtained values from JSON file to the field set. Now the line protocol is completely generated. Source code of converter is shown in Figure 3.26.

```
//Step2_1: Read JSON from file
FILE *fp;
fp = fopen("get.json", "r");
char buffer[1024];
int len = fread(buffer, 1, 1023, fp); // read up to 1023 bytes
buffer[len] = '\0'; // add null terminator
fclose(fp);

//Step2_2: Parse JSON
json_error_t error;
json_t *root = json_loads(buffer, 0, &error);
if(!root) {
    fprintf(stderr, "Error parsing JSON: %s\n", error.text);
    exit(1);
}

//Step2_3: Get fields from JSON
json_t *sd = json_object_get(root, "sd");
json_t *state = json_object_get(root, "state");
json_t *temperature = json_object_get(root, "temperature");

//Step2_4: Extract data from fields
int sd_ready = json_boolean_value(json_object_get(sd, "ready"));
const char *text = json_string_value(json_object_get(state, "text"));
double bed_actual = json_real_value(json_object_get(json_object_get(temperature, "bed"), "actual"));
double tool0_actual = json_real_value(json_object_get(json_object_get(temperature, "tool0"), "actual"));

//Step2_5: Output data in InfluxDB line protocol format
char sd_ready_result[1024]={'\0'};
char state_text_result[1024]={'\0'};
char bed_actual_result[1024]={'\0'};
char tool0_actual_result[1024]={'\0'};

sprintf(sd_ready_result,"a,name=sd_ready value=%d", sd_ready);
sprintf(state_text_result,"a,name=state_text value=%s", text);
sprintf(tool0_actual_result,"a,name=tool0_actual value=%f", tool0_actual);
sprintf(bed_actual_result,"a,name=bed_actual value=%f", bed_actual);

printf("%s\n",sd_ready_result);
printf("%s\n",state_text_result);
printf("%s\n",tool0_actual_result);
printf("%s\n",bed_actual_result);

//Step2_6: Clean up
json_decref(root);
```

Figure 3.26: Source code of converter

3.4.2.4. Mosquitto

In order to realize the protocol of MQTT, Mosquitto is introduced in this project, with the version of 2.0.11. It is open source, lightweight, and fully functional, so it's very suitable for embedded development. It provides a C language library, as well as the terminal command, including 'mosquitto_sub' and 'mosquitto_pub'. The following two pictures demonstrate the use of 'mosquitto_sub' and 'mosquitto_pub' in the terminal respectively(Figure3.27 and Figure3.28). -h specifies the target IP address, -t specifies the topic of the subscription. -m specifies the message to publish. After executing the 'mosquitto_sub' command, data from the subscribed topic will pop up in the terminal. On the contrary, after executing the 'mosquitto_pub' command, data behind '-m' will be published to the target topic.

```
mosquitto_sub -h [IP address] -t [topic]
```

Figure 3.27: Usage of 'mosquitto_sub'

```
mosquitto_pub -h [IP address] -t [topic] -m [message]
```

Figure 3.28: Usage of 'mosquitto_pub'

3.4.3. Software in MQTT broker

3.4.3.1. MQTT Explorer

MQTT explorer can intuitively manage all topics under the MQTT protocol. Figure 3.29 shows the login screen of MQTT explorer, where users can select the connection name, protocol to use, host name, port number, username, and password. When it comes to network security, users also have the option of using valid certificates or encryption. After clicking the connect button, users will enter the main interface of MQTT explorer, as shown in Figure 3.30. On the left side of the main interface, topics under various directories are displayed. After selecting the target topic, users can see the messages published by other devices. On the right side of the interface, users can see the information published under the current topic, as well as its corresponding time information and history. For the same topic, users can also manually publish in the right side of the interface, which supports RAW, XML or JSON format. In this case, other devices that subscribe to the topic will receive the information.

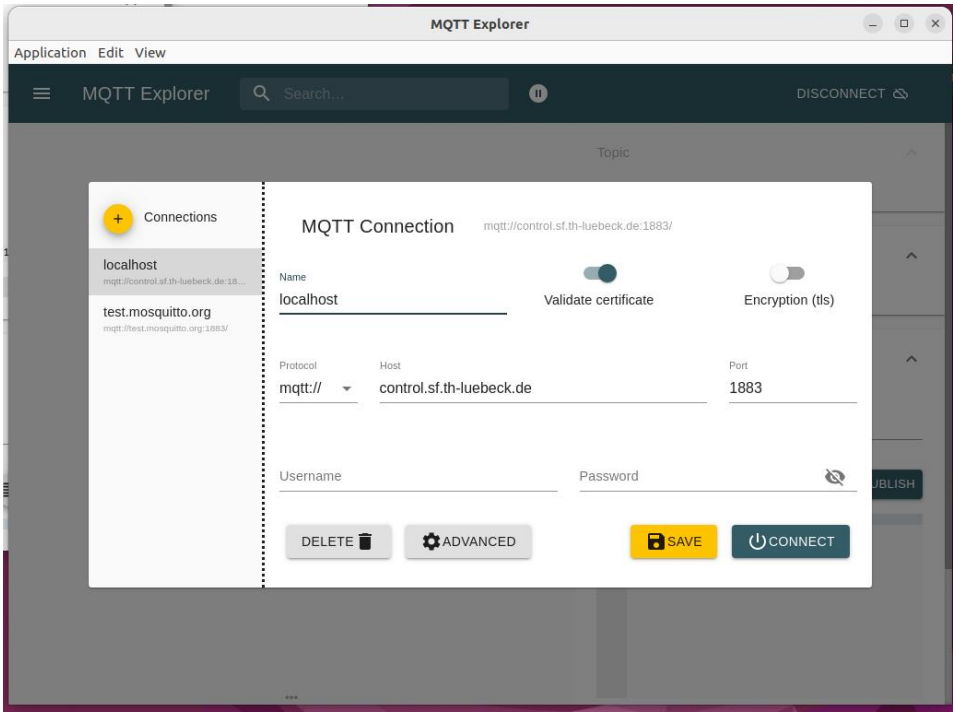

Figure 3.29: The login screen of MQTT Explorer

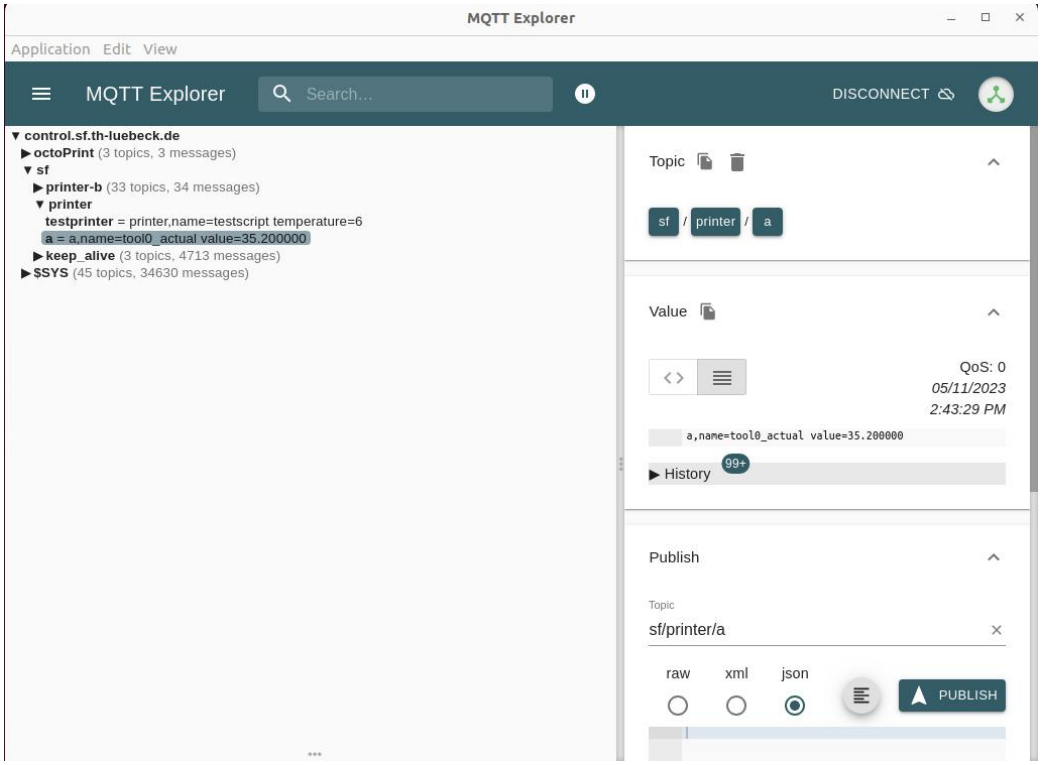

Figure 3.30: GUI of MQTT Explorer

3.4.3.2. InfluxDB Data Explorer

After successfully logging in to influxDB, the Data Explorer interface appears, as shown in Figure 3.31. If the message is sent through the MQTT protocol and conforms to the line protocol format, the name of topic can be seen at the bottom of the interface. For example, for this project, it is sf/printer/a. When the corresponding topic is selected, the internal attribute value will appear. Select the desired property values and click submit once on the right, and all the property values will be displayed as graph in the top half of the Data Explorer. In addition, the format of data presentation can also be chosen, such as tables, etc.

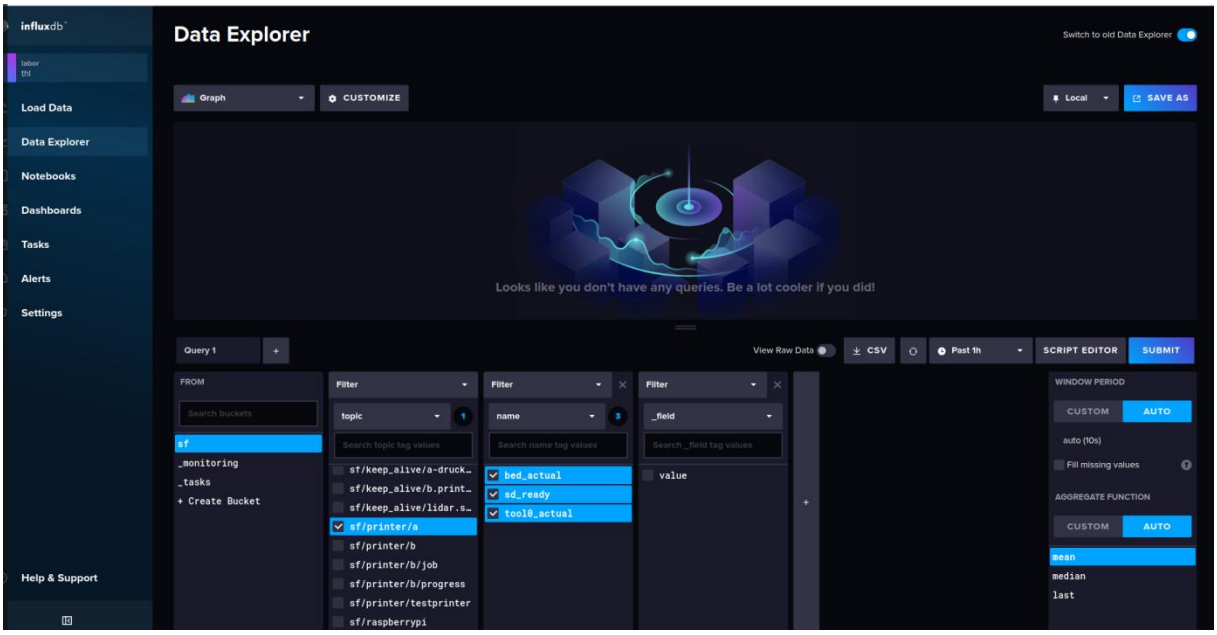

Figure 3.31: GUI of influxDB

3.4.4. Details of the information transfer

3.4.4.1. 3D printer to Raspberry Pi

Since the 3D printer cannot send messages to the Raspberry Pi autonomously, an indirect way is needed to get messages from the 3D printer. As mentioned in Section 3.4.2.2, REST API can be used to get information about 3D printer. Therefore, a program based on this will be introduced to solve this problem. Figure 3.32 shows the schematic diagram of the program and figure 3.33 shows the source code of the program. The program uses curl to send an HTTP request to the machine, and the 3D printer will return a JSON format string of the current state of the 3D printer. By using the character '>' behind the command, it is possible to write the feedback to a specific file, like get.json, which is more convenient for processing.

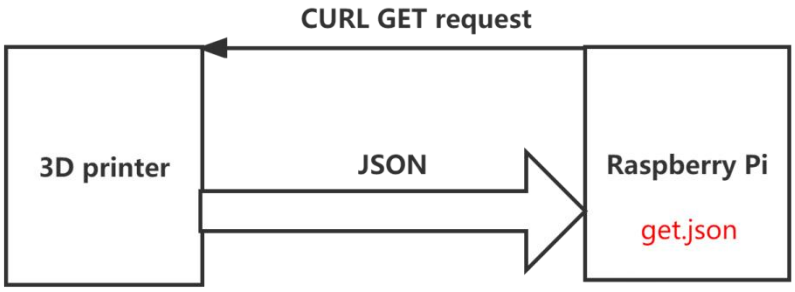

Figure 3.32: Schematic diagram of data transfer from 3D printer to Raspberry Pi

```
//Step1: Get json from printer and write the result to get.json
    char finalGetCommand[1024]={'\0'};
    char localIP[] = "curl -X GET -H 'Content-Type':'application/json' \"127.0.0.1";
    char middle[] = "?apikey=";

    strcat(finalGetCommand,localIP);
    strcat(finalGetCommand,getCommand);
    strcat(finalGetCommand,middle);
    strcat(finalGetCommand,apiKey);
    strcat(finalGetCommand,"\" > get.json");

   // printf("finalcommand = %s\n",finalGetCommand);
    system(finalGetCommand);
```

Figure 3.33: Source code of data transfer from 3D printer to Raspberry Pi

3.4.4.2. Raspberry Pi to MQTT broker

After retrieving the get.json file, as explained in Section 3.4.2.3, using the JSON to line protocol converter will successfully produces the following four pieces of data in the JSON file: sd_ready, state_text, tool0_actual and bed_actual converted to line protocol format. The transformed result is sent to topic: ‘sf/printer/a’ in MQTT broker through 'mosquitto_pub' command as mentioned in Section 3.4.2.4. Figure 3.34 shows the schematic diagram of the program and figure 3.35 shows the source code.

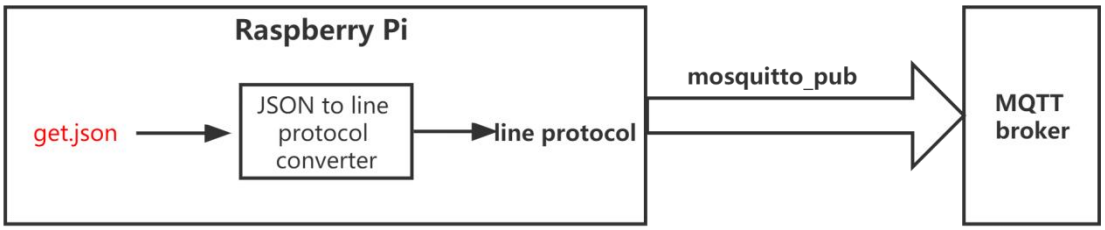

Figure 3.34: Schematic diagram of data transfer from Raspberry Pi to MQTT broker

```
//Step3: Send the line protocol to the broker via MQTT

char mqtt_sd_ready_result[1024]={'\0'};
char mqtt_state_text_result[1024]={'\0'};
char mqtt_bed_actual_result[1024]={'\0'};
char mqtt_tool0_actual_result[1024]={'\0'};

sprintf(mqtt_sd_ready_result,"mosquitto_pub -h 172.31.128.141 -t sf/printer/a -m \"%s\"",sd_ready_result);
sprintf(mqtt_state_text_result,"mosquitto_pub -h 172.31.128.141 -t sf/printer/a -m \"%s\"",state_text_result);
sprintf(mqtt_bed_actual_result,"mosquitto_pub -h 172.31.128.141 -t sf/printer/a -m \"%s\"",bed_actual_result);
sprintf(mqtt_tool0_actual_result,"mosquitto_pub -h 172.31.128.141 -t sf/printer/a -m \"%s\"",tool0_actual_result);

printf("%s\n",mqtt_sd_ready_result);
printf("%s\n",mqtt_state_text_result);
printf("%s\n",mqtt_bed_actual_result);
printf("%s\n",mqtt_tool0_actual_result);

system(mqtt_sd_ready_result);
system(mqtt_state_text_result);
system(mqtt_bed_actual_result);
system(mqtt_tool0_actual_result);
```

Figure 3.35: Schematic diagram of sending line protocol format string to MQTT broker via 'mosquitto_pub' command

3.4.4.3. MQTT broker to Raspberry Pi

In principle, sending messages to Raspberry Pi by MQTT broker should be done by using 'mosquitto_sub' command in Raspberry Pi. But because of the limitation of mosquitto, 'mosquitto_sub' can only subscribe to the string format data in the related topic. If we need to remotely send a large file to a 3D printer, strings are clearly not acceptable. Instead, curl HTTP post requests are used to send strings and files to the IP address of Raspberry Pi remotely. Figure 3.36 shows the schematic diagram of the program. Figure 3.37 shows an example of sending a file by curl.

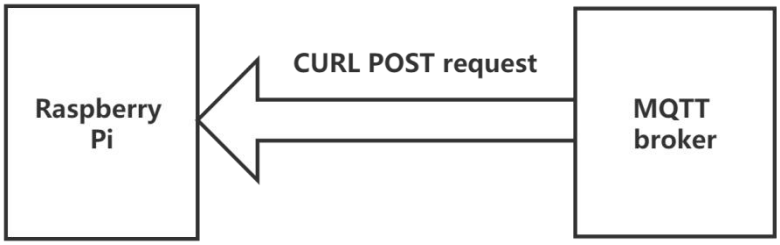

Figure 3.36: Schematic diagram of data transfer from MQTT broker to Raspberry Pi

```
curl -X POST -F 'file=@./Batman_0.2mm_PLA_MK3S_23m.gcode' http://172.31.128.136/api/files/sdcard?apikey=38D2A5008AA646BA8FE889CEC100DC40
```

Figure 3.37: Command to send a G-code file via HTTP request to Raspberry Pi

3.4.4.4. Raspberry Pi to 3D printer

It is relatively easy to send a request directly from the Raspberry Pi to the 3D printer. Users just need to enter the corresponding curl command in the terminal according to the REST API provided on the official website, with the local IP address: 127.0.0.1. Figure 3.38 shows the schematic diagram of the process. Here is an example of sending a command from Raspberry Pi to start the printing job of the 'batman~1.gco' file in the SD card of the 3D printer (Figure 3.39):

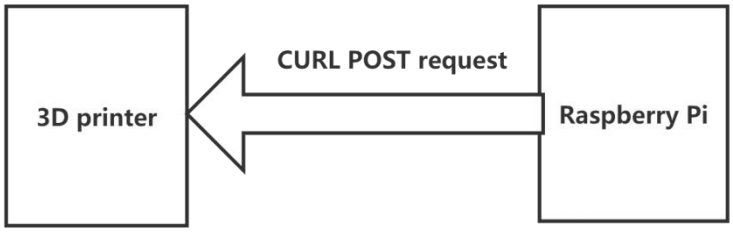

Figure 3.38: Schematic diagram of data transfer from Raspberry Pi to 3D printer

```
curl -X POST -H 'Content-type':'application/json' -d '{"command":"select","print":true}' "127.0.0.1/api/files/sdcard/batman~1.gco?apikey=38D2A5008AA646BA8FE889CEC100DC40"
```

Figure 3.39: Command to start the printing job of the 'batman~1.gco' file in the SD card of the 3D printer

3.4.4.5. MQTT broker and other devices

Other devices can also subscribe to the content of related topics through the 'mosquitto_sub' command. All users need to know is the IP address of the MQTT broker and the topic name. Figure 3.40 gives an example of using the 'mosquitto_sub'. As long as the MQTT broker is running, multiple devices are able to have stable subscriptions at the same time. Figure 3.41 shows the schematic diagram of data transfer from MQTT broker to other devices.

```
mosquitto_sub -h 172.31.128.141 -t sf/printer/a
```

Figure 3.40: Format of a 'mosquitto_sub' command

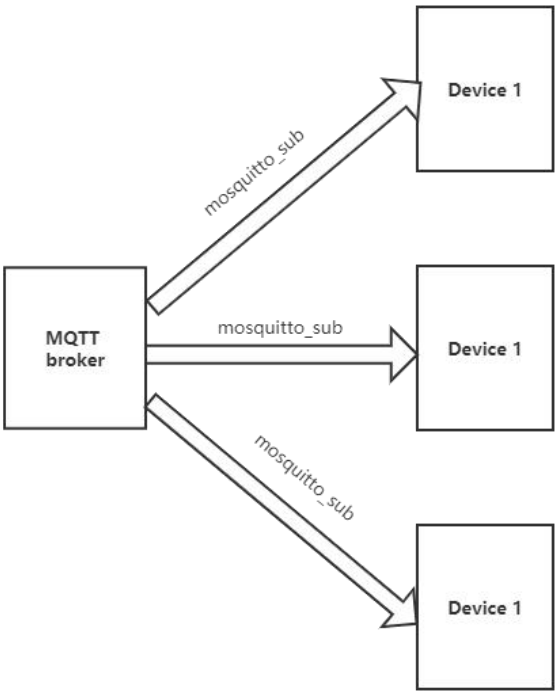

Figure 3.41: Schematic diagram of data transfer from MQTT broker to other devices

3.5. Other work

In order to make this project play a better role in the smart factory laboratory, some additional work has also been completed. A program was designed for the automation of the whole process: obtaining the 3D printer's status, converting the information format and publishing, as well as its self-starting program. In order to implement the remote transfer of files and control of the 3D printer's working state, a remote controller was also designed. With these additional work, the whole system can run stably without supervision. Documentation is also very important. This facilitates later maintenance and further development.

3.5.1. Automation program of obtaining and sending status of 3D printer

A C program was introduced to automate the whole process. Figure 3.42 is the flowchart of the program. As mentioned in Section 3.4.4.1 and 3.4.4.2, in order to obtain and send the status of 3D printer, the following steps need to be completed. The first step is send HTTP POST request to the printer, get JSON from the printer and write the result to get.json. The second step is to convert json to influxDB line protocol. The third step is to send the line protocol to the broker via MQTT. After 2 seconds of sleeping, the program goes back to step 1. Figure 3.43 shows the source code of main function. 'getUrl' function includes the source code of the three steps separately in Section 3.4.4.1, Section 3.4.2.3 and Section 3.4.4.2. It can also specify the path of API, which provides a lot of convenience for later development.

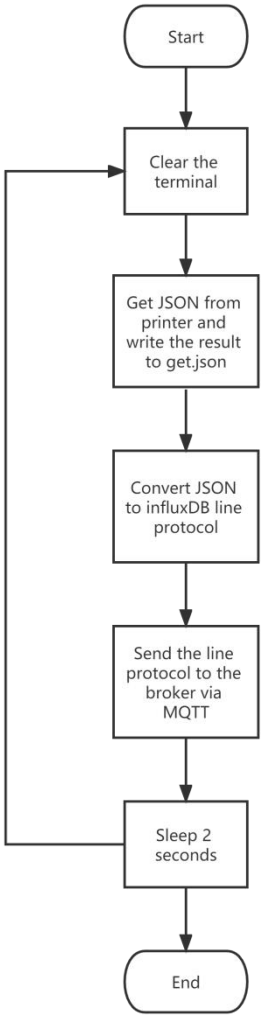

Figure 3.42: Flowchart of the automation program

```
int main(int argc, char *argv[])
{
    while(1){
        system("clear");
            getUrl("/api/printer");
        sleep(2);
    }
}
```

Figure 3.43: Source code of main function

3.5.2. Self-starting program

After designing the program for automation, there is still a problem: every time the Raspberry Pi is restarted manually or with a power outage, the code that was running before the restart will no longer run. To solve this problem, a script file was written. Figure 3.44 shows the contents of the script. The script is very simple: sleep for ten seconds, then execute the

automation program. The reason why sleeping for ten seconds is significant is that when the OctoPi operating system starts up, it needs to go through a series of hardware self-tests as well as the default software program opening and port connection, such as OctoPrint. If users try to open the automation program in the meantime, the curl HTTP request will not be answered, resulting in an error. Therefore, sleeping for 10 seconds ensures that all hardware and software is ready to run the automation program.

```
#!/bin/bash

sleep 10
/home/pi/Smart3DPrinter/Smart3DPrinter
```

Figure 3.44: Source code of main function

After writing this script, as illustrated in figure 3.45, a file called testboot.service needs to be created in the following path: /usr/lib/systemd/system, whose content is shown in Figure 3.46. The script ‘start.sh’ is supposed to be filled in ‘execstart’, and finally ‘systemctl enable’ command is executed in the terminal to realize the boot of the script automatically. Figure 3.47 shows the start of the testboot.service.

```
pi@octopi:~ $ sudo nano /usr/lib/systemd/system/testboot.service
```

Figure 3.45: Edit testboot.service

```
[Unit]
Description=testboot
[Service]
Type=oneshot
ExecStart=/home/pi/start.sh
[Install]
WantedBy=multi-user.target
```

Figure 3.46: Content of testboot.service

```
pi@octopi:~ $ sudo systemctl enable testboot.service
Created symlink /etc/systemd/system/multi-user.target.wants/testboot.service → /lib/systemd/system/testboot.service.
```

Figure 3.47: Start the testboot.service

3.5.3. Remote controller

As mentioned in Section 3.4.4.3, HTTP post requests are used to send files and data from the MQTT broker to Raspberry Pi. But executing curl command in the terminal of MQTT broker directly is not perfect to achieve the effect, because after executing the curl command, it is not possible to confirm whether the file transfer was successful. On top of that, using the curl command remotely is complicated due to different APIs are required to start, pause, resume, or cancel a printing job. So a program needs to be designed to solve this problem. Figure 3.48 shows the flowchart and the source code of the first three steps of the program. The previous three steps check if the user has executed the program in the correct format like in Figure 3.49. If so, the program will automatically concatenate the user's input into a curl command and execute it in the terminal. If not, the program will be automatically exited.

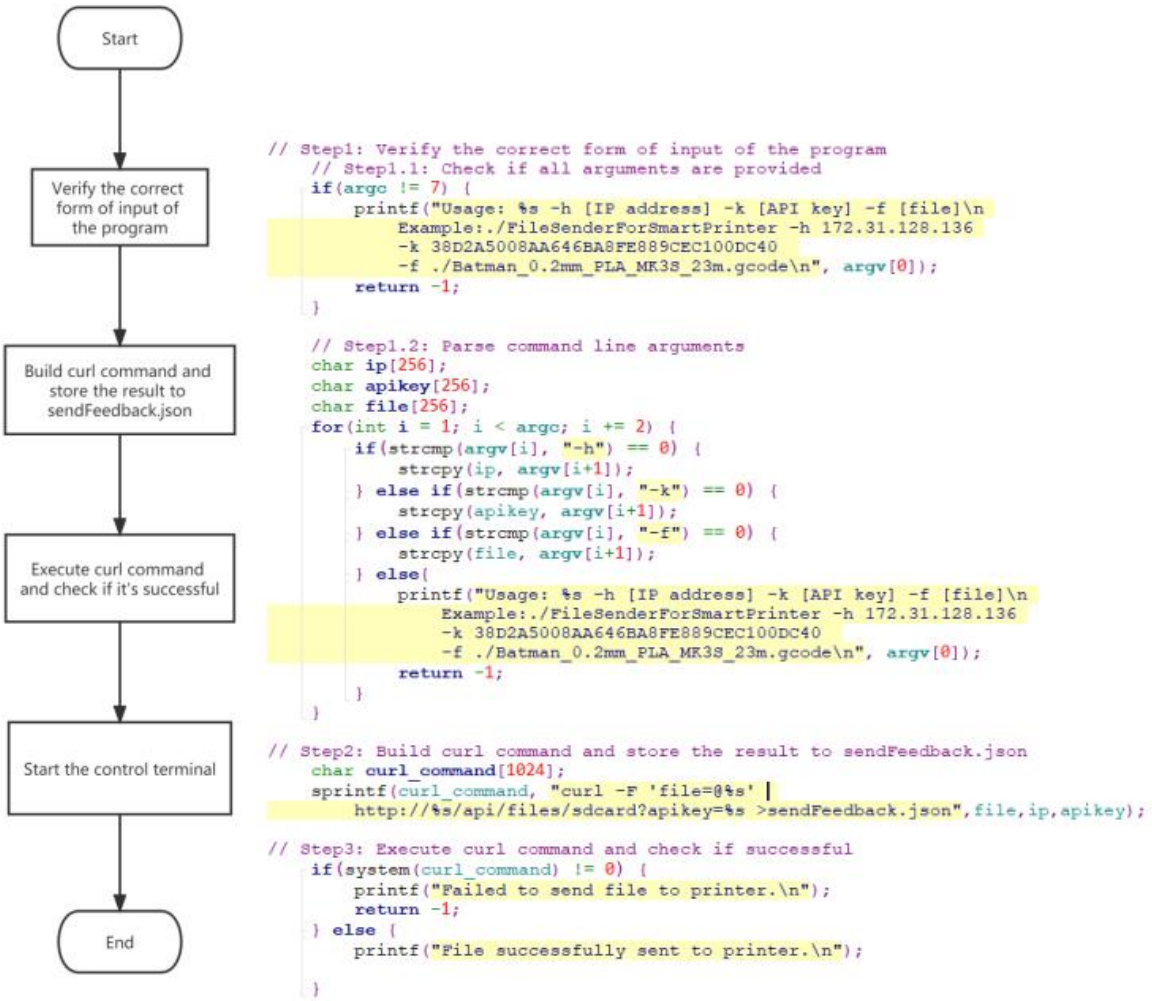

Figure 3.48: Flowchart and part of the source code of the remote controller program

command -h [IP address] -k [API key] -f [file]

Figure 3.49: The correct format for executing the program

The following figure shows the flowchart for the control terminal and part of the source code. The program first absorbs a string from the user and evaluates it. If the string 'resend' is received, the program will send the file the user entered again. If the user enters 'quit', the entire program exits. If the user enters 'cancel', 'pause', or 'resume', then the program will execute the corresponding HTTP request. If the user executes other command, the program will tell the user all the commands it support. Figure 3.50 shows the flowchart and part of the source code of the terminal.

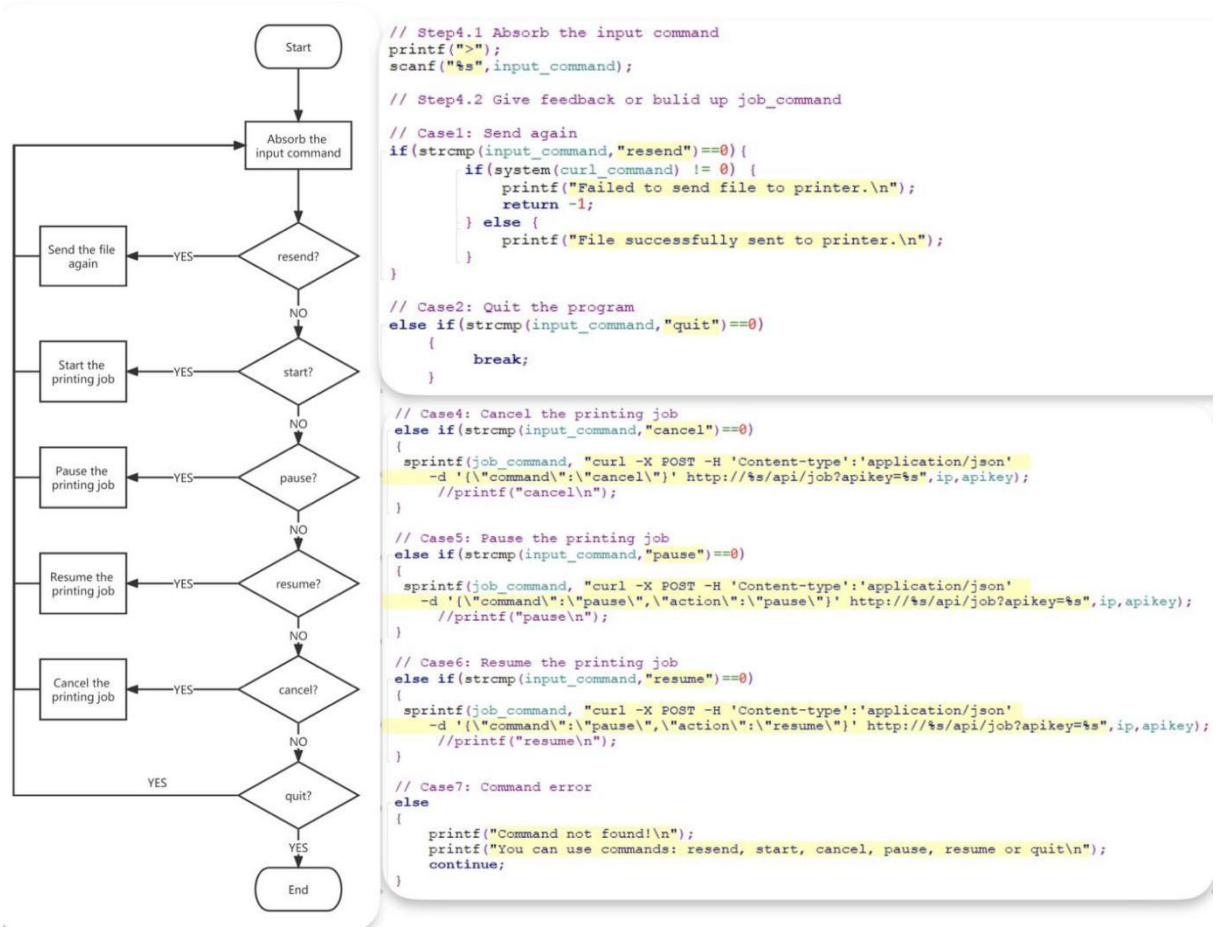

Figure 3.50: Flowchart and part of the source code of the controller terminal

The most complicated part is to start the printing job. Due to the limitation of OctoPrint, every file sent over HTTP has its name changed to a combination of its first six characters, followed by a tilde '~', and a number. For example, file 'Batman_0.2mm_PLA_MK3S_23m.G-code' will become 'BATMAN~1.GCO'. So if user tries to execute the original filename, it will return an error that the file does not exist. Thus, the program need to read the feedback from the HTTP post request, figure out the filename that OctoPrint accepts, and send the correct command to start the printing job. Figure 3.51 shows the flowchart and the source code of the 'start' command.


```
// Case3: Start the printing job
else if(strcmp(input_command,"start")==0)
    {
         printf("start\n");

     // 1. Get the target resource
        // Read JSON from file
        json_error_t error;
        json_t *root = json_load_file("sendFeedback.json", 0, &error);
        if(!root) {
            fprintf(stderr, "Error parsing JSON: %s\n", error.text);
            return 1;
        }

         // Get fields from JSON
         json_t *files = json_object_get(root, "files");
         json_t *local = json_object_get(json_object_get(files, "local"), "refs");
         json_t *sdcard = json_object_get(json_object_get(files, "sdcard"), "refs");

         // Extract data from fields
         const char *local_resource = json_string_value(json_object_get(local, "resource"));
         const char *sdcard_resource = json_string_value(json_object_get(sdcard, "resource"));

         // Output data
         printf("Local resource: %s\n", local_resource);
         printf("SD card resource: %s\n", sdcard_resource);

     // 2. Build up the job_command
     sprintf(job_command, "curl -X POST -H 'Content-type':'application/json'
     -d '{\"command\":\"select\",\"print\":true}' %s?apikey=%s",sdcard_resource,apikey);

     // 3. Clean up
     json_decref(root);

    // printf("start command:%s\n",job_command);

     } « end if strcmp(input_command,... »
```

Figure 3.51: Flowchart and the source code of the 'start' command

After the job_command is successfully built up, the program will finally execute it and validate whether the file transfer is success. Figure 3.52 shows the source code of it.

```
// Step4.3 Execute job_command and validate
if(system(job_command) != 0) {
    printf("Failed to send command to printer.\n");
} else {
    printf("Command successfully sent to printer.\n");
}
```

Figure 3.52: Flowchart and the source code of the 'start' command

3.5.4. Documentation

At this point, all the parts of the project have been covered. To make it more clear for users to use the automation program as well as remote controller, a README.md has also been included in the project. The content of the file is shown in Figure 3.53. This file tells how to compile and run the program, which is convenient for users to use this program normally after secondary development.

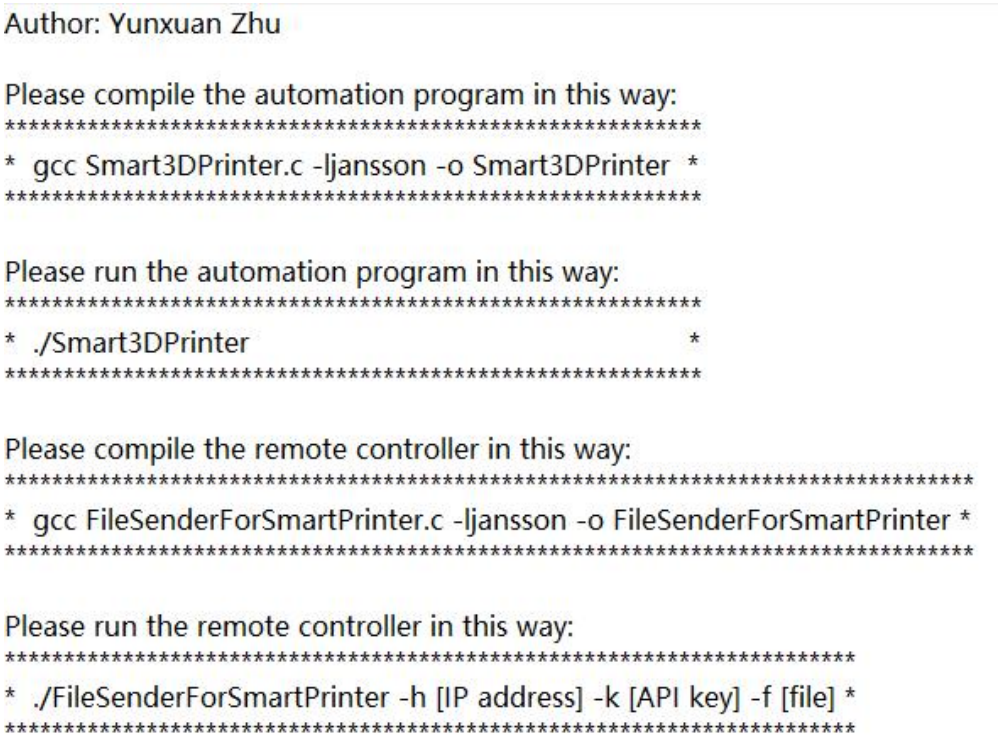

Figure 3.53: The content of the README.md

4. Evaluation

In order to test the performance of the system, this section will be divided into three sections for evaluation. The first section verifies by the feedback of the software in MQTT broker, the section section is to verify the rendering effect in other devices, and the final section is to verify the feedback from OctoPrint by executing the remote controller. Also, the existing problems are attached.

4.1. Results in MQTT broker

4.1.1. MQTT Explorer

As shown in figure 4.1, all the status information about the 3d printer is successfully displayed in the topic of 'sf/printer/a' in MQTT Explorer. In order to avoid confusion caused by too many topics in the subsequent process of subscribing for other devices, all information is integrated in a same topic, but this does not affect the timeliness of all groups of data.

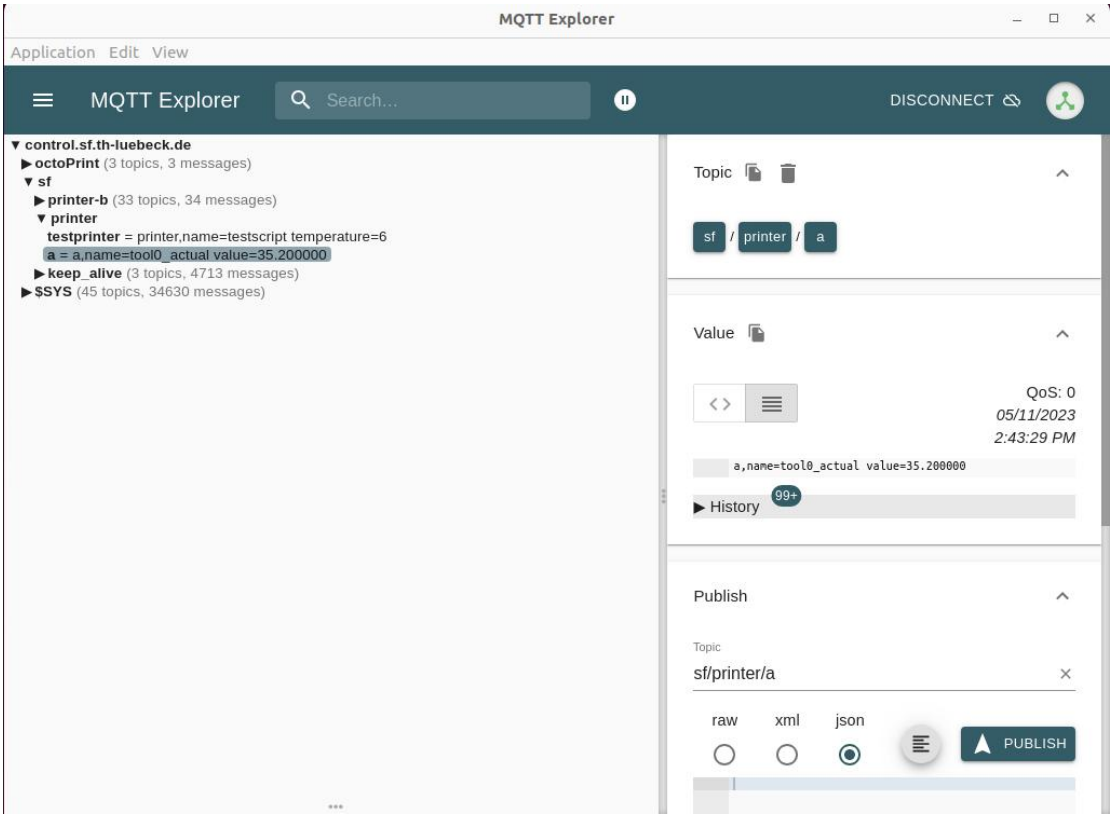

Figure 4.1: Information displayed in MQTT Explorer

4.1.2. InfluxDB Data Explorer

After ensuring that the MQTT Explorer can collect the messages from the Raspberry Pi, the user will find that all the line protocol format data will be displayed at the bottom of the interface in influxDB Data Explorer. When the ‘submit’ button is clicked, the selected data will be displayed as a graph in the middle part of the screen, just like the one in Figure 4.2. However, because the printer's state is a string, it doesn't show up in the Data Explorer.

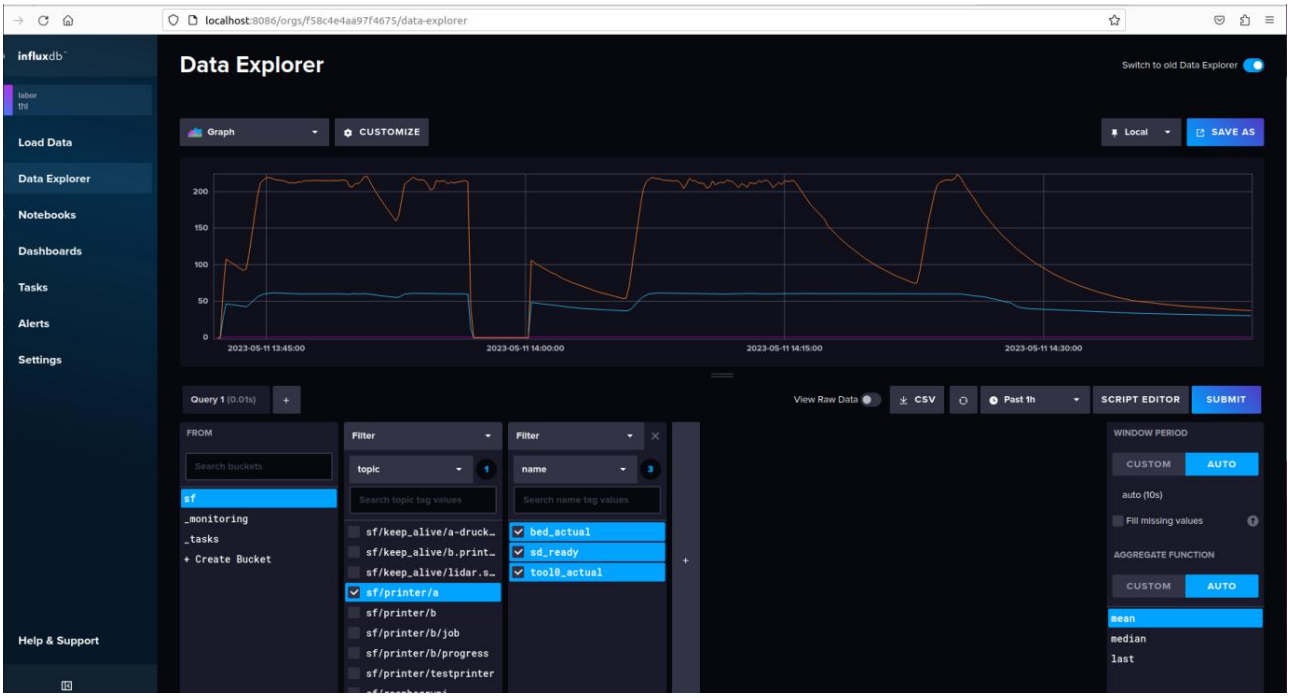

Figure 4.2: Generated graph after submission

4.2. Results in other devices

4.2.1. Display data in Google Glass Enterprise 2

This is a program finished by Chenyu Han. Figure 4.3 shows the Google Glass Enterprise 2. The project uses Google Glass Enterprise 2 to scan a QR code attached to the device, and the result is the corresponding device name. Based on this, messages from different topics subscribed on MQTT Explorer are accepted according to the device name and finally displayed on the screen of the glasses. Figure 4.4 shows all the data presented in the glasses interface simulator on the computer, including the current state of the printer, the temperature of the nozzle, the temperature of the print bed, and the status of the SD card.

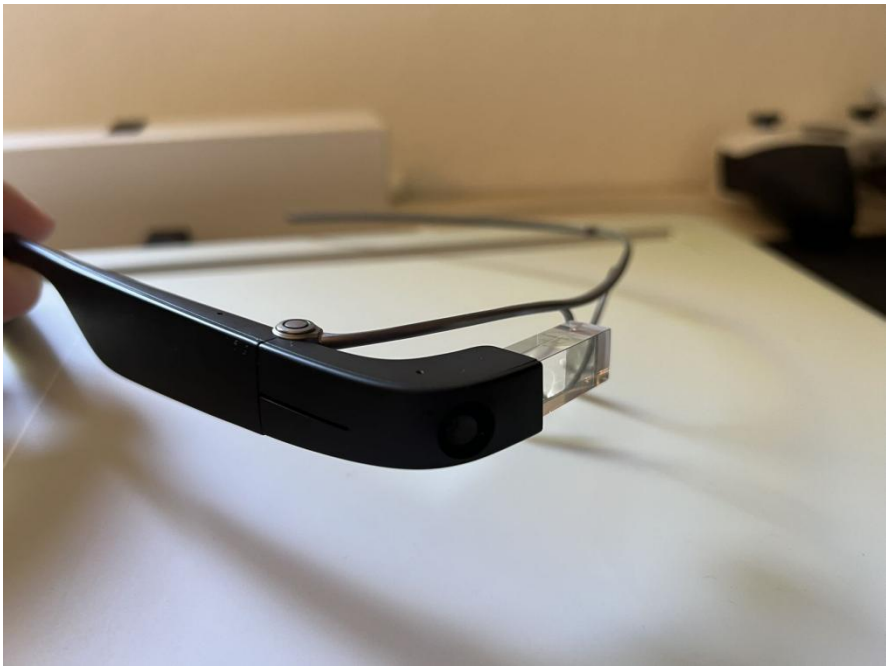

Figure 4.3: Google Glass Enterprise 2

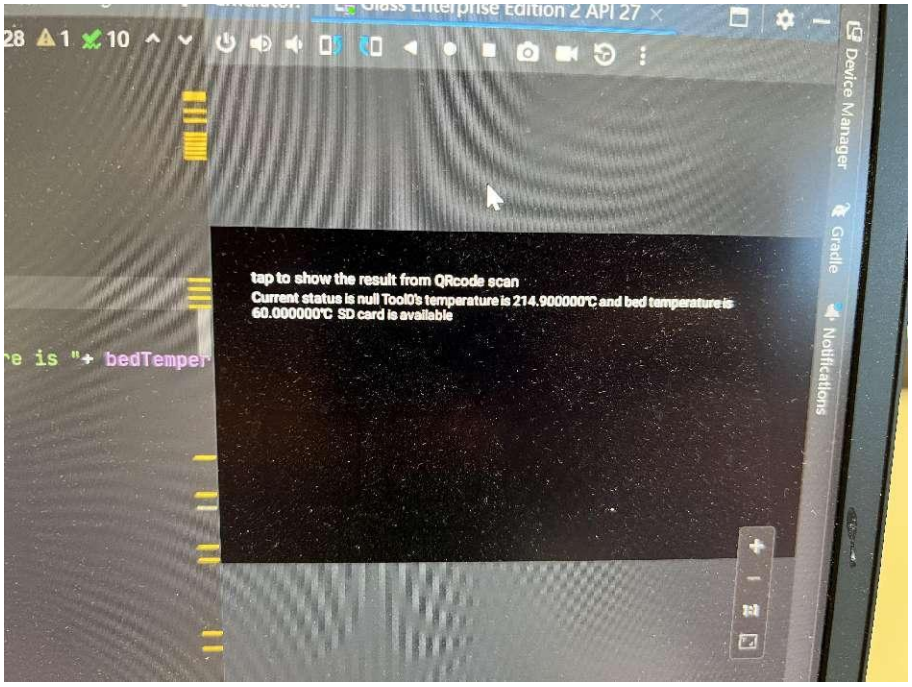

Figure 4.4: Results displayed in Google Glass Enterprise 2

4.2.2. Display data in Vive Pro 2

This is a program finished by Yiren Wang. Figure 4.5 shows the Vive Pro 2. It consists of a VR goggles and two controllers. With the controllers and goggles, the user can enter the virtual world. The project models a virtual smart factory through unity. In this smart factory, there are two virtual 3D printers. The printer number for this project is a. By subscribing the topic ‘sf/printer/a’ in MQTT Explorer, Vive Pro 2 will have access to all the information about the 3d printer. the information will appear simply by using the controller to point at the target printer and pull the trigger off. The result is shown in Figure 4.6.

Figure 4.5: Vive Pro 2

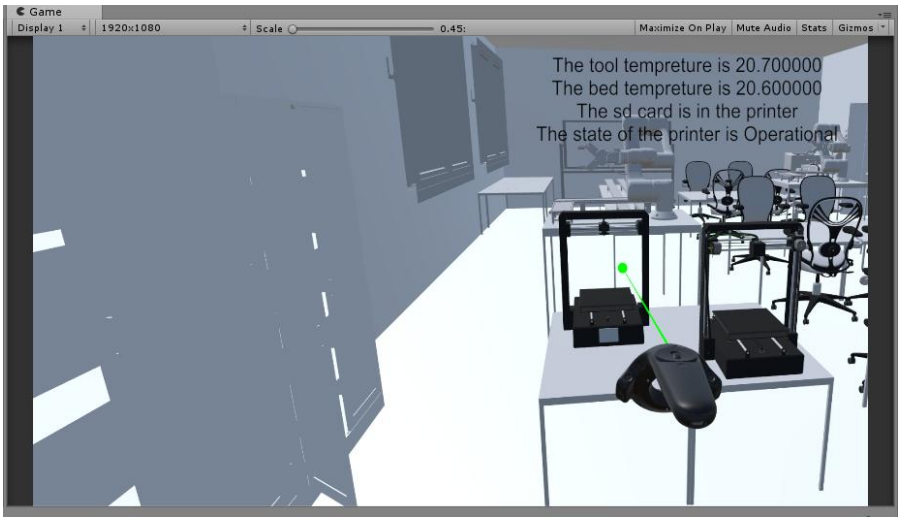

Figure 4.6: Results displayed in Vive Pro 2

4.3. Send files and control the 3D printer remotely

4.3.1. Preparation

Similar to the standard steps of using a 3D printer, the G-code file needs to be generated before it can be transferred remotely. The figure 4.7 below shows the 3D modeling process, slicing process and exporting G-code file process for a 2cm cube. After successfully exporting the G-code file, the user needs to copy the corresponding G-code file to the MQTT broker using a removable storage device. As you can see in the figure 4.8, the corresponding G-code file has been successfully copied into the project directory, so it's ready for the remote transfer.

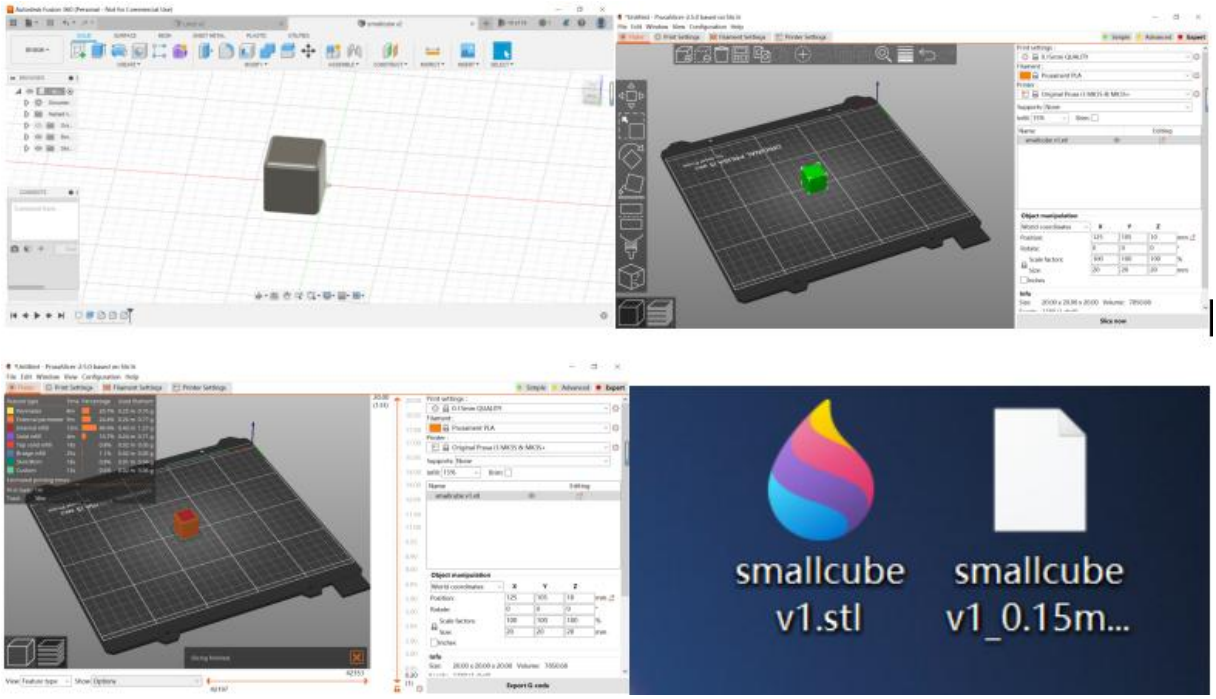

Figure 4.7: Preparation for generating G-code file

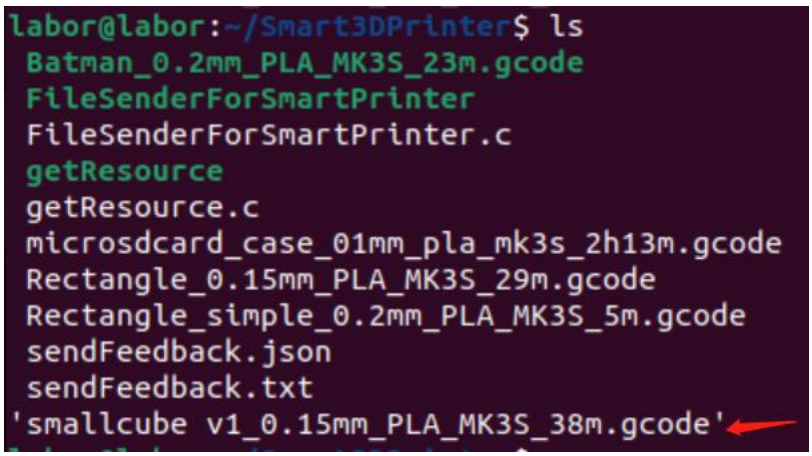

Figure 4.8: The G-code file copied in the broker

4.3.2. Start the program

Following READ.me, the program is run with the IP address of the Raspberry Pi, the API key, and the path of the G-code file. As illustrated in figure 4.9, the terminal shows that the file has been successfully sent to the printer. To verify the result, OctoPrint on the Raspberry Pi is open, and the state interface shows that the stat has turned from ‘Operational’ to ‘Sending file to SD’. This means the success transfer of the file. The state interface of OctoPrint is shown in figure 4.10.

```
labor@labor:~/Smart3DPrinter$ ./FileSenderForSmartPrinter -h 172.31.128.136 -k 38D2A5008AA646BA8FE889CEC100DC40
 -f ./smallcube\ v1_0.15mm_PLA_MK3S_38m.gcode
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1112k  100   541  100 1112k   1104  2269k --:--:-- --:--:-- --:--:-- 2270k
File successfully sent to printer.
>
```

Figure 4.9: Terminal message after starting the program

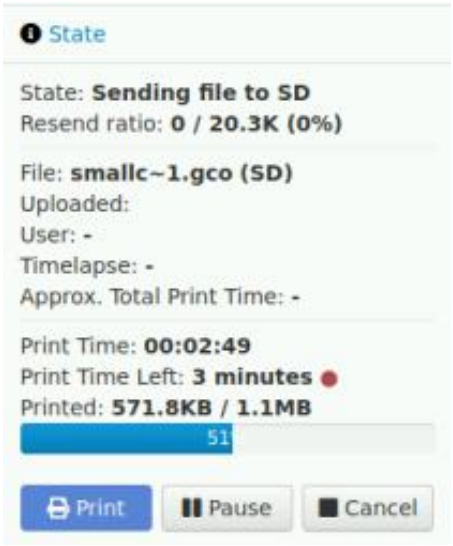

Figure 4.10: The state interface of OctoPrint after starting the program

4.3.3. Resend the file

To test the remote controller's re-sending functionality, the users can enter the string 'resend' into the terminal. Soon the message 'File successfully sent to printer' will appear. Figure 4.11 shows the terminal messages after entering ‘resend’. To verify the result, Octoprint is supposed to be opened again. It can be found in Figure 4.12 that the ‘state’ in the state interface will display as ‘sending file to SD’ again, which proves the feasibility of resend.

```
>resend
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  174k  100   557  100  173k    312  99587  0:00:01  0:00:01 --:--:-- 99890
File successfully sent to printer.
```

Figure 4.11: Terminal messages after entering ‘resend’

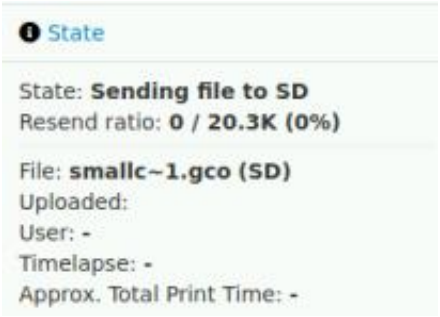

Figure 4.12: The state interface of OctoPrint after entering the 'resend'

4.3.4. Start the job

In order to test the remote controller's functionality of starting the printing job, the users can enter the string 'start' into the terminal. Figure 4.13 shows the reaction of the terminal, OctoPrint and the video stream. A string with the name of the file stored on the SD card is returned and printed to the terminal, as well as the message 'File successfully sent to printer'. Back to OctoPrint, it can be found that a printing job has been started. Corresponding to the video stream, it can be seen that the nozzle has started working, and the printing job which is going to execute is a 2cm cube.

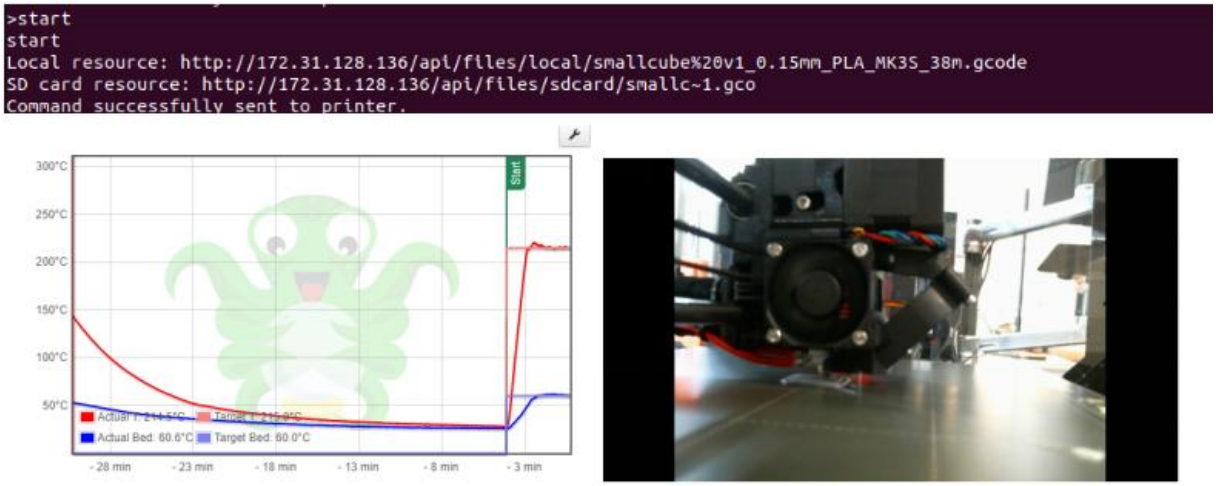

Figure 4.13: Results after entering 'start' in the terminal

4.3.5. Pause the job

To test the remote controller's functionality of pausing the printing job, the users can enter the string 'pause' into the terminal. Figure 4.14 shows the reaction of the terminal, OctoPrint and the video stream. The message 'File successfully sent to printer' is returned and printed to the terminal. Back to OctoPrint, it can be found that the printing job has been paused on the graph. The video stream shows that the nozzle has moved back to its original position, which can verify that the printing job has been paused.

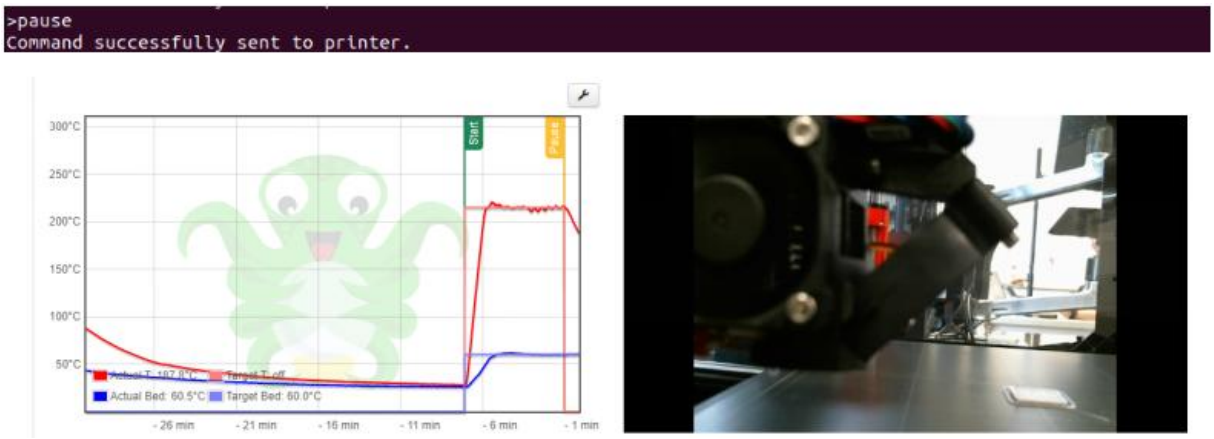

Figure 4.14: Results after entering 'pause' in the terminal

4.3.6. Resume the job

To test the remote controller's functionality of resuming the printing job, the users can enter the string 'resume' into the terminal. Figure 4.15 shows the reaction of the terminal, OctoPrint and the video stream. The message 'File successfully sent to printer' is returned and printed to the terminal. Back to OctoPrint, it can be found that the previous printing job has been resumed on the graph. The video stream shows that the nozzle moves back to its working area, which can verify that the printing job has been resumed.

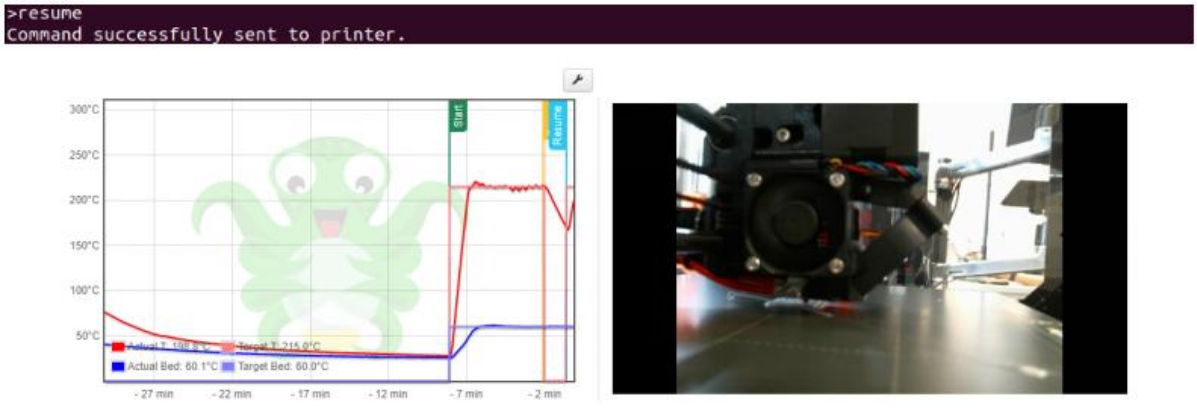

Figure 4.15: Results after entering 'resume' in the terminal

4.3.7. Cancel the job

To test the remote controller's functionality of canceling the printing job, the users can enter the string 'cancel' into the terminal. Figure 4.16 shows the reaction of the terminal, OctoPrint and the video stream. The message 'File successfully sent to printer' is returned and printed to the terminal. Back to OctoPrint, it can be found that the previous printing job has been canceled on the graph. The video stream shows that the nozzle moves to the position above the product and raise its height, which can verify that the printing job has been canceled.

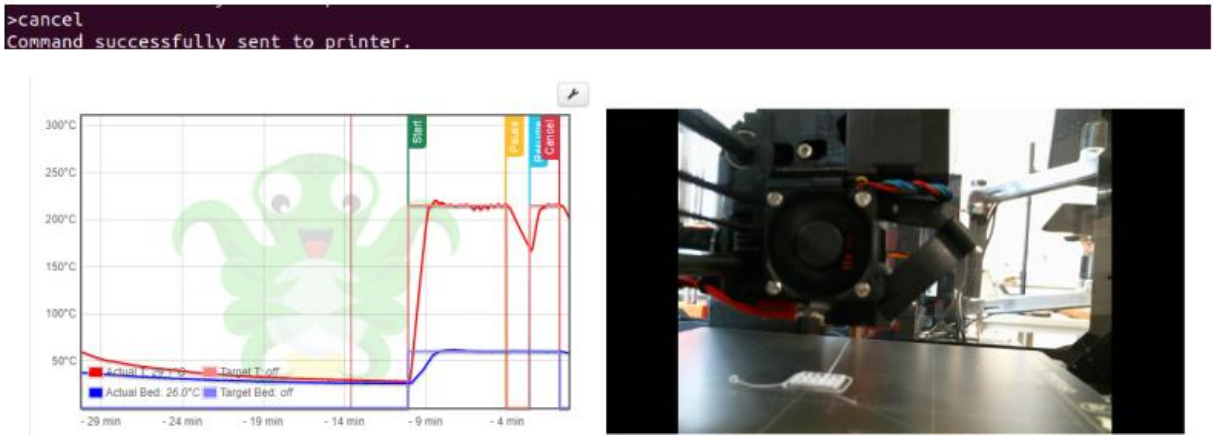

Figure 4.16: Results after entering ‘cancel’ in the terminal

4.3.8. Quit the program

To quit the program, the users can enter the string 'quit' into the terminal. Figure 4.17 shows that the terminal of the remote controller has exited and returned to the default terminal of the Linux system.

Figure 4.17: Terminal messages after entering ‘quit’ in the terminal

4.4. Remaining problems

Although the above results show that all aspects of the system have been implemented, but in the process of testing, many problems are still found. This section will point out those problems, as well as functional imperfections, so as to further improve the usability of this system in the future development.

4.4.1. Slow file transmission speed

After executing the sending command, the users can see the real-time progress bar of the file receive in the OctoPrint interface. As is shown in figure 4.18, a 1.1 MB file takes about 5 minutes to transfer in total, which means that the transfer rate is only 3.67kBps. This transfer speed is not tolerable for 3D printed files, since the size of a 3D printed file is generally larger than 1MB. It doesn't make sense to use a program of remote transmission at that speed, because it may take more time for transmitting rather than printing. So in the future development, slow transmission speed is a very urgent problem to solve.

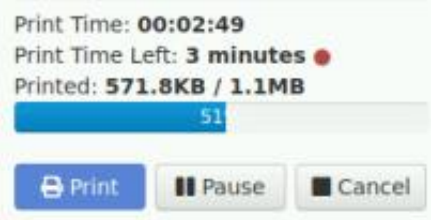

Figure 4.18: The state interface of OctoPrint after starting the program

4.4.2. 'Send-start' problem

After the user starts the remote controller program, the file entered by the user will be automatically sent to the 3D printer, but this takes some time. The remote controller program does not integrate a progress bar for sending files. The terminal in figure 4.19 below only shows the progress of writing the JSON file string returned by OctoPrint to the local file on MQTT broker after sending the file. Therefore, the actual progress of file transfer is unknown to the MQTT broker. Therefore, if the user enters 'start' in the terminal at this time, the program will report an error, because the corresponding target file has not been written to the SD card, and there will be a communication error in OctoPrint, which may even cause the 3D printer to be disconnected. Figure 4.20 shows the communication error in OctoPrint.

```
labor@labor:~/Smart3DPrinter$ ./FileSenderForSmartPrinter -h 172.31.128.136 -k 38D2A5008AA646BA8FE889CEC100DC40
 -f ./standardsmallcube_0.15mm_PLA_MK3S_11m.gcode
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  174k  100   557  100  173k   1581   492k --:--:-- --:--:-- --:--:--  493k
File successfully sent to printer.
>start
start
Local resource: http://172.31.128.136/api/files/local/standardsmallcube_0.15mm_PLA_MK3S_11m.gcode
SD card resource: http://172.31.128.136/api/files/sdcard/standa~3.gco
{"error":"The requested URL was not found on the server. If you entered the URL manually please check your spel
ling and try again."}
Command successfully sent to printer.
>
```

Figure 4.19: Results in the terminal if entering 'start' before the file is completely transmitted

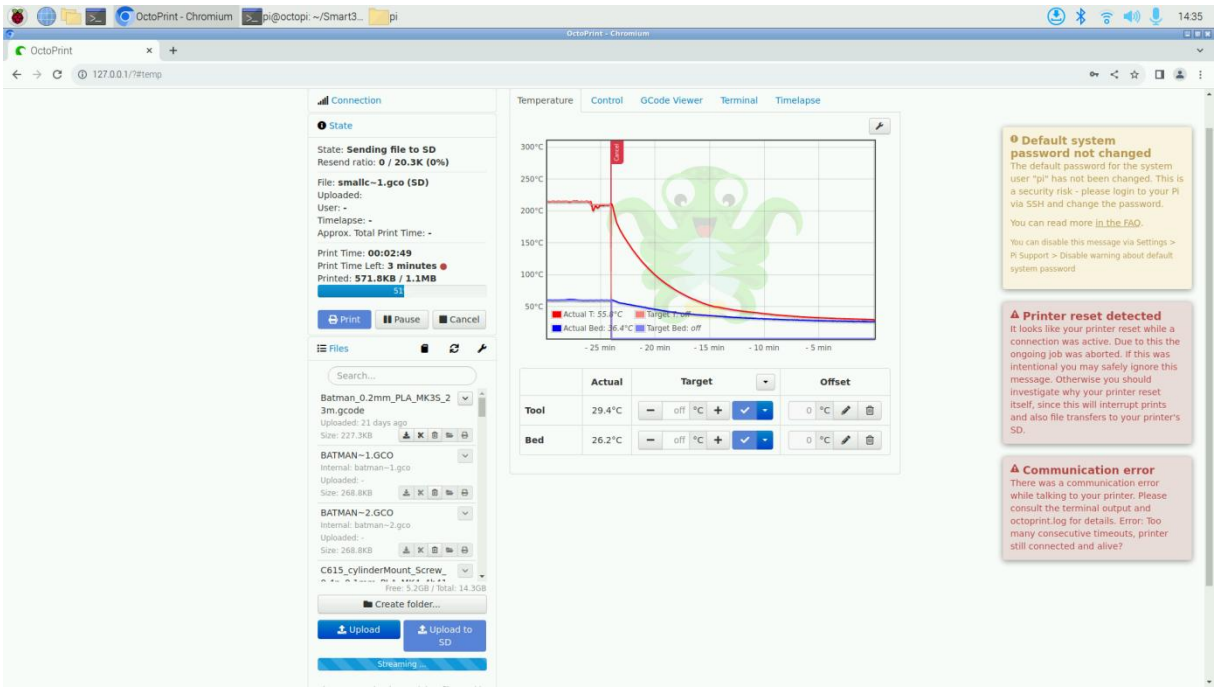

Figure 4.20: Communication error in OctoPrint

4.4.3. Video stream transmission

With OctoPrint, users can see the video stream on the industrial display from the camera mounted next to the 3D printer. But in this project, due to the limitations of MQTT protocol, it is very difficult to view this video stream remotely. For example, on the MQTT broker, the user only has access to the control terminal in the form of strings, and cannot see the visual video stream. So maybe the secondary development in the future can focus on using other protocols to transmit video streams in order to solve this problem.

4.4.4. Transfer G-code files across platforms

Although remote controller implements the function of remote transmission of G-code files, G-code files also need to be generated and transmitted across platforms in this project. This means that, for example, a user generates an STL file using a 3D modeling software in the Windows operating system, and after being sliced by a slicing software, the generated G-code file must be transferred to the Linux operating system through a removable storage device or other network protocol. This is very inconvenient to some extent. Therefore, in the secondary development, it should be considered that integrating the 3D modeling software and slicing software to the Linux system. Alternatively, another solution can be remotely operating the windows system on the Linux system to automatically complete the series of operations of 3D modeling, slicing, and generating G-code files, and transmitting the generated results to the Linux system.

5. Conclusion and outlook

In conclusion, this thesis has proposed a solution for sending the real-time information of a 3D printer to an MQTT broker in the smart factory laboratory, with the help of Raspberry Pi and MQTT protocol. In addition, a remote controller has been designed to send G-code files remotely from the MQTT broker, as well as controlling the printing job.

By developing a comprehensive solution, consisting of hardware components such as the 3D printer, Raspberry Pi, camera, and MQTT broker, along with the implementation of OctoPi and Ubuntu as the operating systems, the feasibility and potential of our proposed approach has been demonstrated. On that basis, the software structure presented and discussed in Raspberry Pi and MQTT broker ensures the stable information transfer process.

The evaluation of the solution revealed promising results, as observed in the MQTT broker and demonstrated through the data display on devices like Google Glass Enterprise 2 and Vive Pro 2. Moreover, the remote controller successfully transfer G-code files and start a feasible terminal to control the printing job of the 3D printer remotely.

While the proposed solution has shown significant advancements, there are still some remaining challenges that need to be addressed. These challenges include the slow file transmission speed, the "send-start" problem, video stream transmission for MQTT broker, and seamless transfer of G-code files across different platforms.

Looking ahead, further research and development can be undertaken to overcome these challenges and enhance the proposed solution. Future work should focus on increasing the file transmission speed, refining the system's reliability and user-friendliness, and exploring alternative approaches for video stream transmission. Additionally, efforts can be directed towards the cross-platform transfer of G-code files to ensure compatibility and ease of use. Overall, this thesis has laid the foundation for advancements of secondary development in the smart factory laboratory.

Acknowledgment

First of all, I want to thank all the professors, staff members and students who helped me directly or indirectly with this paper. One of my childhood dreams was to be able to learn about smart factory in college. So I would like to thank the Lübeck University of Applied Sciences for providing us with such a good opportunity and platform to complete our project. At the same time, I would like to thank Professor Pelka, who gave us this opportunity to enter the smart factory laboratory to carry out project development, as well as providing us with a lot of suggestions and tips when we met difficulties in the process. I also want to thank Mr. Kittelmann. Without him we would not have access to the laboratory. In addition, I would like to thank two of the students from ECUST, Chenyu Han and Yiren Wang. Their project names are respectively ‘Augmented Reality with Google Glass Enterprise’ and ‘Virtual Reality Environment with HTC Vive Pro 2’. In the process of completing the project, we helped each other and finally achieved the expected results.

Appendix - List of figures

References

[1] Kakade S, Mulay A, Patil S. IoT-based real-time online monitoring system for open ware FDM printers[J]. Materials Today: Proceedings, 2022, 67: 363-367.

[2] Lima D B C, da Silva Lima R M B, de Farias Medeiros D, et al. A performance evaluation of raspberry Pi zero W based gateway running MQTT broker for IoT[C]//2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE, 2019: 0076-0081.

[3] Abd-Elrahim A M, Abu-Assal A, Mohammad A A A A, et al. Design and Implementation of Raspberry Pi based Cell phone[C]//2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE). IEEE, 2021: 1-6.

[4] Saxena A, Tyagi M, Singh P. Digital Outing System Using RFID And Raspberry Pi With MQTT Protocol[C]//2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU). IEEE, 2018: 1-4.

[5] Al-Shahethi M G, Muneer A, Ghaleb E A A, et al. Real life monitoring of conveyor line speed using IoT and raspberry Pi[C]//2021 International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE). IEEE, 2021: 1-5.

[6] Hoque M M, Jony M M H, Hasan M M, et al. Design and implementation of an FDM based 3D printer[C]//2019 International conference on computer, communication, chemical, materials and electronic engineering (IC4ME2). IEEE, 2019: 1-5.

[7] General defination of Fused Deposition Modeling (FDM) [Online]. https://en.wikipedia.org/wiki/Fused_filament_fabrication#Fused_deposition_modeling

[8] Romero-Alva V, Alvarado-Diaz W, Roman-Gonzalez A. Design of a 3D printer and integrated supply system[C]//2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON). IEEE, 2018: 1-4.

[9] Bukhari S B H, Tanveer T, Abid A, et al. Design and Fabrication of Inexpensive Portable Polar 3D Printer[C]//2023 International Conference on Robotics and Automation in Industry (ICRAI). IEEE, 2023: 1-6.

[10] Hozdić E. Smart factory for industry 4.0: A review[J]. International Journal of Modern Manufacturing Technologies, 2015, 7(1): 28-35.

[11] Mabkhot M M, Al-Ahmari A M, Salah B, et al. Requirements of the smart factory system: A survey and perspective[J]. Machines, 2018, 6(2): 23.

[12] Osterrieder P, Budde L, Friedli T. The smart factory as a key construct of industry 4.0: A systematic literature review[J]. International Journal of Production Economics, 2020, 221: 107476.

[13] Yang K, Zhang B, Zhang J, et al. Design of Remote Control Inverter Based on MQTT Communication Protocol[C]//2021 IEEE International Conference on Mechatronics and Automation (ICMA). IEEE, 2021: 1374-1378.

[14] Yassein M B, Shatnawi M Q, Aljwarneh S, et al. Internet of Things: Survey and open issues of MQTT protocol[C]//2017 international conference on engineering & MIS (ICEMIS). Ieee, 2017: 1-6.