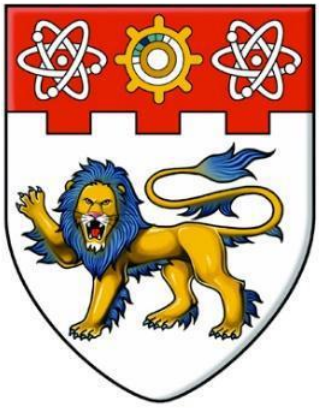


NANYANG TECHNOLOGICAL UNIVERSITY

SCSE21-0534
IoT - Development of a Home Automation System 3

Submitted by: Ooi Kok Yih

Supervisor: A/P Lau Chiew Tong

Examiner: A/P Tan Rui

School of Computer Science and Engineering

AY 2021/2022

Nanyang Technological University

SCSE21-0534
IoT – Development of a Home Automation System 3

Submitted by: Ooi Kok Yih (U1921262E)

A Final Year Project submitted to the School of Computer Science and Engineering,

Nanyang Technological University

In Partial Fulfilment of the Requirements for the Degree of Bachelor of Engineering

(Computer Engineering)

Abstract

Advancement in technologies has made IoT more practical and one of the most important technologies of the 21st century. It has gained worldwide attention from manufacturers, consumers, research institutes, and developer communities. IoT has the potential to revolutionise industries such as transportation, healthcare, power and energy, and home automation.

Home automation has gained popularity due to the benefits it brings such as security, energy efficiency, and better quality of life. The development of faster and cheaper microprocessors has enabled users to effortlessly install, monitor, and control home appliances remotely over the internet.

This project aims to develop a smart home system by integrating home appliances with ESP32 microprocessors and various third-party APIs.

Acknowledgments

The author would like to express his sincere and utmost gratitude to Associate Professor Lau Chiew Tong for his guidance and support despite his busy schedule. He has provided useful tips for the project and also gave the liberty for the author to come up with different ideas for the project. It was an honour to work under him on this project.

The author would also like to show his appreciation to his examiner Associate Professor Tan Rui for his support and time to examine the project.

Lastly, the author would like to extend his thanks to his classmates for their unwavering support and the School of Computer Science and Engineering for incorporating the Final Year Project into the curriculum and providing the knowledge to see this project through.

Table of Contents

Abstract 2
Acknowledgments 3
Table of Contents 4
List of Figures & Illustrations 6
List of Tables 7
List of Abbreviations 8
1. Introduction 9
1.1 Background 9
1.2 Previous Research 10
1.3 Objective and Scope 10
2. Literature Review 11
2.1 Home automation products available in the market 11
2.1.1 Samsung Smarthings 11
2.1.2 Google Home 12
2.1.3 Smart Devices – August Smart Lock Pro 12
3. Project Overview 13
3.1 Hardware 13
3.2 Software 14
3.2.1 Blynk 14
3.2.2 Google Vision API 19
3.2.3 Telegram Bot API 20
4. Design & Implementation 21
4.1 Room Simulation 21
4.1.1 Circuit Diagram 21
4.1.2 Flowchart Diagram 22
4.2 Door Simulation 23
4.2.1 Circuit Diagram 23
4.2.2 Flowchart Diagram 24
4.2.3 RFID Reader 25
4.2.4 ESP32-CAM Facial Recognition 25
4.3 Monitoring System with Object Detection 27
4.3.1 Circuit Diagram 27
4.3.2 Flowchart Diagram 28

4.3.3 NodeJS Server for Google Vision API 29
4.3.4 Notification system with Telegram Bot API 32
4.4 Touch Screen Remote Controller 33
4.4.1 Circuit Diagram 33
4.4.2 Flowchart Diagram 34
4.4.4 Converting Image to XBM format 36
4.4.5 Blynk Cloud HTTP API 38
5. Testing and Results 40
5.1 Room Simulation 40
5.1.1 Controlling Light and Fan on Blynk App 40
5.1.2 Using Motion Sensor to control Light and Fan 40
5.1.3 Reading Rain Sensor on Blynk App 41
5.2.1 Opening Door on Blynk App 42
5.2.2 Opening Door with RFID Reader 42
5.2.3 Opening Door with Facial Recognition 43
5.3 Monitoring System with Object Detection 44
5.3.1 Google Vision API result 44
5.3.2 Telegram Bot Notification 45
5.4 Touch Screen Remote Controller 46
6. Conclusion & Future Recommendation 47
7. References 48
Appendix A – ESP32-WROOM-32 Pinout Diagram 50
Appendix B - ESP32-CAM AI-Thinker Pinout Diagram 50
Appendix C – Features of Telegram Bot API 51
Appendix D – Blynk functions to Read and Write data to Server 52
Appendix E – Telegram Bot Send Message Function 53

List of Figures & Illustrations

Figure 1: How Google Home operates [9] 12
Figure 2: August Smart Lock Pro [10] 12
Figure 3: Blynk working principle [11] 14
Figure 4: Creating a template on Blynk 15
Figure 5: Creating DataStream for hardware 15
Figure 6: Customising Web Dashboard in Blynk 16
Figure 7: Assigning Datatype to Blynk Widget 16
Figure 8: Customising Mobile Dashboard in Blynk App 17
Figure 9: Importing Blynk Library to hardware 17
Figure 10: Setting up Blynk for hardware 18
Figure 11: Linking hardware to Blynk Server and Blynk App 18
Figure 12: Google Vision API image processing flow process [13] 19
Figure 13: Telegram Bot API on Home Automation [14] 20
Figure 14: Room Simulation Circuit Diagram 21
Figure 15: Room Simulation Flowchart Diagram 22
Figure 16: Door Simulation Circuit Diagram 23
Figure 17: Door Simulation Flowchart Diagram 24
Figure 18: Registering RFID Token into program 25
Figure 19: Logic Process of RFID Reader 25
Figure 20: Web server hosting for livestream of ESP32-CAM 25
Figure 21: ESP32-CAM Web server user interface 26
Figure 22:Object Detection Camera Circuit Diagram 27
Figure 23: Object Detection Camera Flowchart Diagram 28
Figure 24: Image transfer process from Camera to-and-fro Google Vision API 29
Figure 25: Posting image to server using HTTP 29
Figure 26: Setting Google Vision API Key path 30
Figure 27: NodeJS server listen for HTTP Post 30
Figure 28: Image saved in file path declared by user 30
Figure 29: Function to call Google Vision API and receive results 31
Figure 30: Getting Bot and User ID for Telegram Bot set up [15] 32
Figure 31: Reading Json file and sending message to Telegram 32
Figure 32: Touch Screen Remote Controller Circuit Diagram 33

Figure 33: Touch Screen Remote Controller Flowchart Diagram 34
Figure 34: ILI9341 LCD Display coordinates layout 35
Figure 35: drawXBitmap function for ILI9341 LCD display 35
Figure 36: Example of online image converter to XBM format [17] 36
Figure 37: Example of an image file in XBM format 36
Figure 38: ringMeter function for sensor reading display 37
Figure 39: Example of sensor reading display using ring meter by Bodmer [18] 37
Figure 40: Diagram of multiple ESP32s communication to Blynk Cloud 38
Figure 41: HTTP request to get data values using Blynk HTTP API [19] 39
Figure 42: HTTP request to update data values using Blynk HTTP API [20] 39
Figure 43: Picture of Light and Fan control via App 40
Figure 44: Picture of using Motion Sensor to on Light and Fan 40
Figure 45: Picture of Blynk App reading Rain Sensor 41
Figure 46: Picture of Door control via Blynk App 42
Figure 47: Picture of unlocking Door with RFID Reader 42
Figure 48: Picture of using Facial Recognition to Open Door 43
Figure 49: Picture of Google Vision API returning result with confidence score 44
Figure 50: Picture of desired object detected along with notification on Telegram 45
Figure 51: Picture of Remote Controller Sending and Receiving data with Blynk 46

List of Tables

Table 1: List of hardware components 13
Table 2: Parameters of the HTTP request 39

List of Abbreviations

IoT	Internet of Things
GPIO	General Purpose Input Output
LED	Light Emitting Diode
PIR	Passive Infrared Sensor
RFID	Radio-Frequency Identification
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
APP	Application
API	Application Programming Interface
AI	Artificial Intelligence
UID	Unique Identifier
IP	Internet Protocol
LCD	Liquid Crystal Display
XBM	X Bitmap

1. Introduction

1.1 Background

The Internet of Things (IoT), widely referred to as the Internet of Everything, is a new paradigm envisioned as a cluster network of machines and devices capable of communicating with each other. With this capability, the IoT has started an era where man and machine merge. In recent years, studies have shown that these IoTs are present in numerous applications ranging from simple machinery sensors to complex organ implants, which also emphasize its potential in versatility and popularity [1]. Thus, IoT has attracted vast attention from a broad spectrum of industries and was estimated an increase of 25 billion units from 2009 to 2020, ultimately gaining momentum in adopting this technology in various fields [2].

One such field is Smart Home Automation, where technologies are utilized to automate home devices through the features of IoT. For instance, automatically switching on a heater when the temperature is low and shutting the windows whenever rain is detected. It is a trending concept where according to Digital Market Outlook, the number of smart home devices in the market is forecasted to increase from 223 million in 2020 to 478 million by 2025 [3]. Well-known consumer electronics companies have introduced their version of smart home devices and systems such as SmartThings from Samsung and Apple HomeKit from Apple to compete in this thriving market. However, despite the vast amount of smart home products available in the market, consumers still face a dilemma when purchasing a product that satisfies their requirements. This happened because most of these products were designed as a closed ecosystem concerning their brand to retain customer loyalty. Thus, arises the issue of compatibility where consumers are forced to stick to one brand to ensure that their smart home devices function in harmony.

1.2 Previous Research

A recent study acknowledged this compatibility problem in existing smart home ecosystems [4]. However, the approach proposed in this research requires multiple types of software and hardware to operate. In addition, users have to configure the software manually whenever they wish to add a new device into their ecosystem. This task requires technical and programming knowledge, which causes inconvenience for consumers. In summary, there is still no practical way to unify devices of different brands in existing smart home systems. As technology advances, new smart home devices will be invented, which will inevitably lead to more compatibility issues. Hence, there needs to be a solution that supports heterogeneous smart home devices while maintaining user-friendliness.

1.3 Objective and Scope

This project aims to develop an IoT smart home control system capable of accommodating any type of smart home device with ease. The project will focus on areas such as interoperability between devices of different manufacturers with the use of third-party platforms and microcontrollers with Wi-Fi capabilities. The system will be developed using the C++ programming language in Arduino Integrated Development Environment (IDE), a software for microcontroller programming. The microcontroller ESP32 will be used in this project due to its aptness for high-speed computing and reliable Wi-Fi connection.

2. Literature Review

A Literature review was conducted to acquire the necessary knowledge to carry out the project. It is vital to understand the types of home automation systems available in the consumer market today. This review also helps to gain insight and inspiration in implementing the design of the project.

2.1 Home automation products available in the market

The global market for home automation products has grown by 10.3% year over year in the third quarter of 2021 [5]. This increase in demand has led more consumer electronics companies to design and launch their version of an “ideal” home automation system and products. Some designs are implemented in a way that only works within a designated ecosystem, such as SmartThings. Whereas others propose a different approach of adapting their device to function on any popular smart home ecosystems such as Alexa and Google Home.

2.1.1 Samsung Smarthings

SmartThings is an American home automation company founded in 2012 and was bought over by Samsung in 2014 as a move by the Korean company to gain a foothold into smart homes [6]. SmartThings comes with a mobile application and requires a hub to work. The hub is a physical device that looks like a router and can control all compatible smart home devices connected to it. SmartThings is universal and can work with any compatible devices outside of Samsung. However, the functionality and features between Samsung and non-Samsung devices may differ. For instance, the SmartTag by Samsung was designed to work only with Samsung Galaxy devices [7].

Ultimately, SmartThings can be categorised as a partially closed ecosystem as third-party compatible devices can still function, albeit with limited services and support for them. This restriction may be seen as a way for the tech giant company to retain its competitiveness in the market by encouraging users to consider their products for additional benefits.

2.1.2 Google Home

Google Home is a smart speaker which comes with Google Assistant functionality. Users can converse with the speaker to issue commands to control compatible products such as lighting, appliances and fixtures, entertainment systems, etc. Unlike Samsung SmartThings, most smart home devices work natively with Google Home, while only a small portion of devices require third-party software to act as a bridge. In 2018, Google announced that Google Assistant can now control over 5000 distinct devices [8].

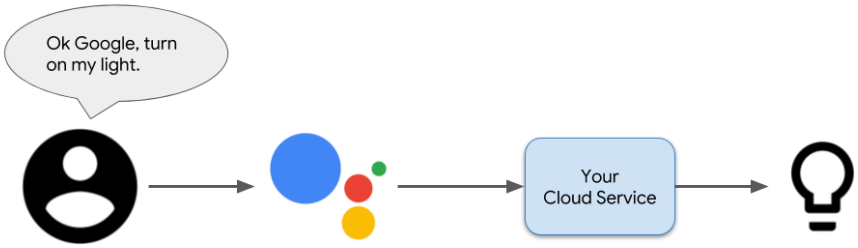


Figure 1: How Google Home operates [9]

As Google home is compatible with a wide variety of smart devices available in the market, it can be considered as an open ecosystem. Comparing both types of ecosystems, it was observed that the control of smart home devices was generally done through a cloud server.

2.1.3 Smart Devices – August Smart Lock Pro

The August Smart Lock Pro is one of the smart devices available in the market where it was designed to adapt itself into an open ecosystem such as Google Home and Alexa.

Figure 2: August Smart Lock Pro [10]

Smart home devices available in the market tend to be off-the-shelf solutions and are usually marked up to a high cost. There needs to be an alternative solution such as transforming non-smart home devices into smart home devices.

3. Project Overview

3.1 Hardware

The main component used in this project was the ESP32 microprocessor. The ESP32, created and developed by Espressif Systems, features a dual-core and ultra-low power co-processor of 160MHZ to 240MHZ. The chip also comes with WiFi capabilities which are suitable for IoT communication. Last but not least, the amount of GPIOs available and its low cost makes this microprocessor chip an efficient solution for this project. The types of ESP32s used in this project were ESP32-WROOM-32 and ESP32-CAM AI-Thinker. For the pinout of the two boards, refer to **Appendix A** and **B** respectively.

The other components that were used in this project are listed in the table below:

Table 1: List of hardware components

1.	RC522 RFID Kit
2.	12VDC Solenoid Door Lock
3.	Single Channel 5V Relay Modules
4.	3x 18650 Battery Holder
5.	18650 Batteries
6.	5V L9110 Fan Motor Module
7.	AM312 PIR Motion Sensor
8.	Rain Drop Sensor Module
9.	2.8” ILI9341 240x320 TFT SPI LCD Display Touch Panel
10.	FT232RL USB-TTL Converter
11.	LED Light Bulb

Some of the components used in this project such as the 12VDC Solenoid Door Lock, L9110 Fan Motor Module, and LED Light Bulb were mainly used to simulate door and room conditions respectively. The sensor modules such as the AM312 PIR Motion Sensor, Rain Drop Sensor, RC522 RFID Kit, and ILI9341 TFT SPI LCD Display Touch Panel function as inputs to ESP32 via the GPIO pins for home automation features. Lastly, FT232RL USB-TTL Converter was used to program ESP32-CAM AI-Thinker as the board does not have any port available for direct connection to the computer.

3.2 Software

3.2.1 Blynk

Blynk was utilized in this project to serve as a medium for data transfer and control between the hardware and smartphone. It is an IoT platform for iOS and Android devices that features highly customisable graphical interfaces for controlling and monitoring hardware. Coupled with its ease of usage and setting up, as well as being well-documented, Blynk is a suitable and reliable choice for this project.

Blynk comes with three major components in the platform:

1. Blynk App - The application is used to create a graphical interface by compiling and providing appropriate addresses and widgets.

2. Blynk Server – Acts as a bridge for all communications between hardware and smartphones

3. Blynk Libraries – provides the necessary functions for hardware to communicate with the server and also to process incoming and outcoming commands.

The diagram below illustrates how Blynk operates to transfer data:

Figure 3: Blynk working principle [11]

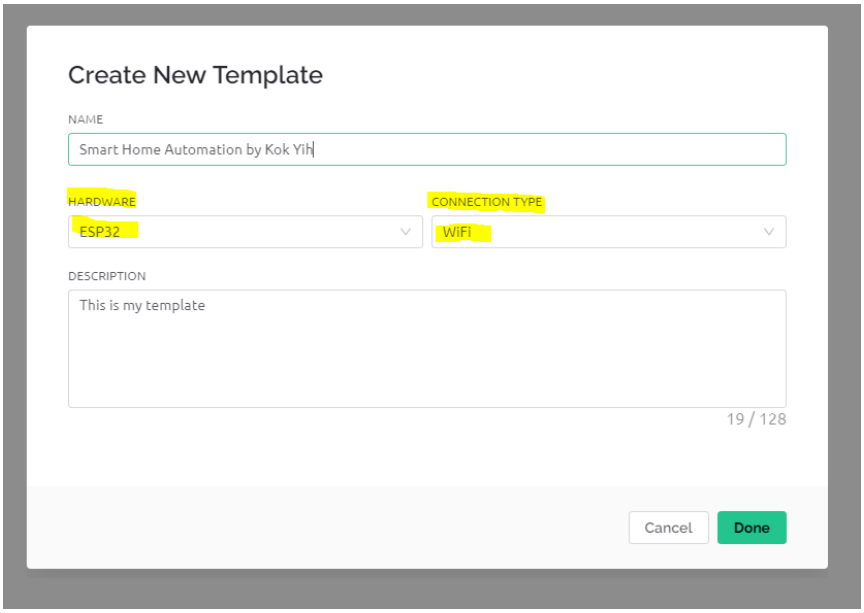


Figure 4: Creating a template on Blynk

To set up Blynk, a template has to be created on their website followed by configuring the hardware type to ESP32 and setting the connection type to be Wi-Fi.

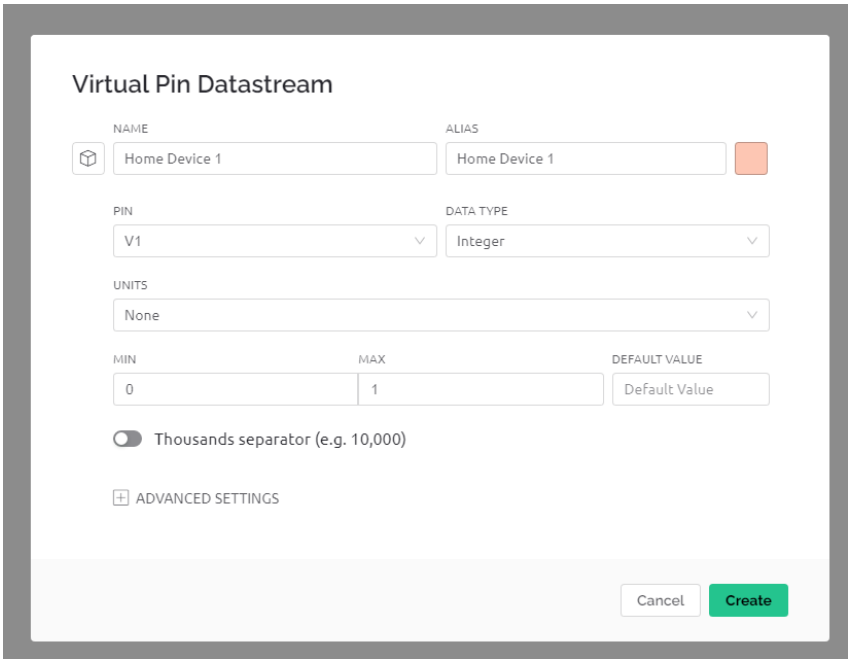


Figure 5: Creating DataStream for hardware

Next, create a DataStream and configure its pin, data type, and any other values that suites the respective hardware input or output data. The pin number in this case does not have to be exactly similar to the desired hardware GPIO pin to be controlled.

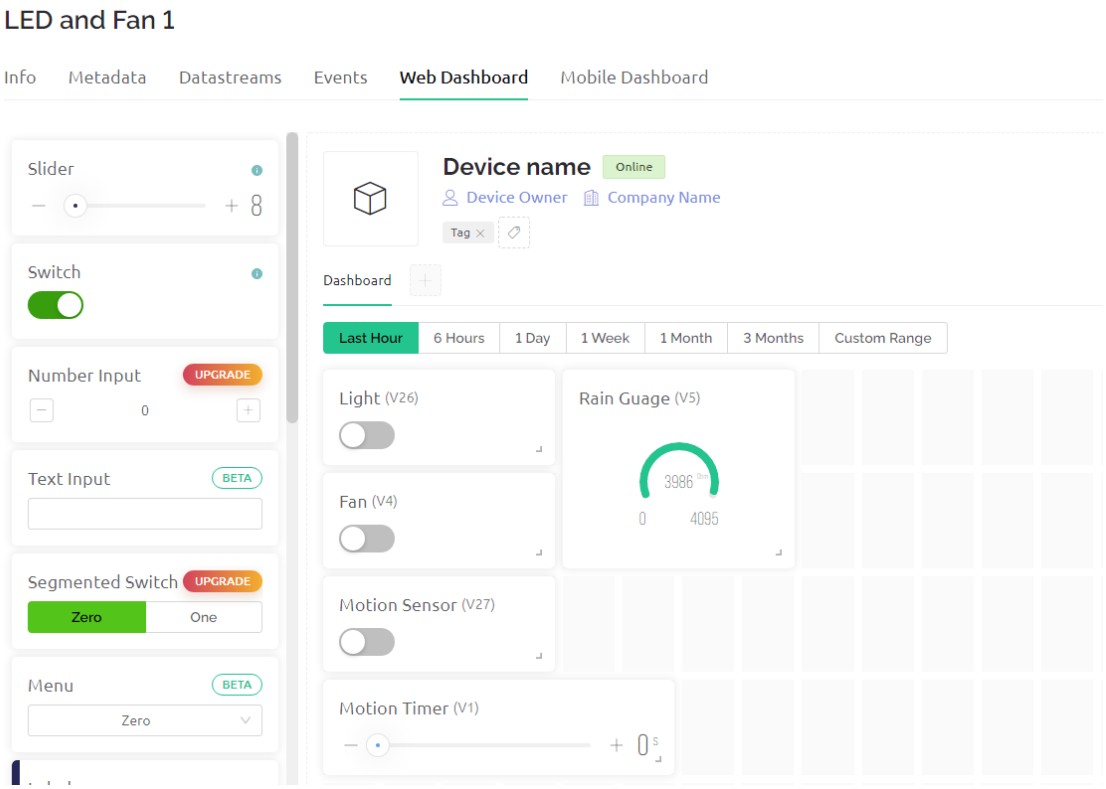


Figure 6: Customising Web Dashboard in Blynk

Then, create and customise the widgets on the Web Dashboard. These widgets will act as an interface to monitor or control the hardware.

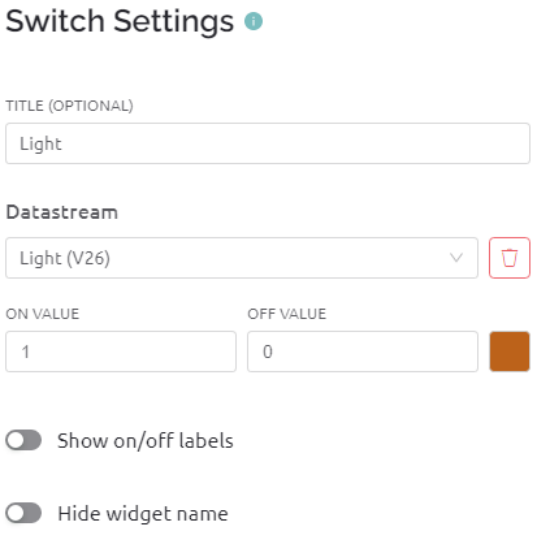


Figure 7: Assigning Datatype to Blynk Widget

After that, assign each widget to their datatype and the Web Dashboard is now able to communicate with the hardware through interacting with these widgets. Effectively, this Web Dashboard sets up the hardware controls to be accessible by computer.

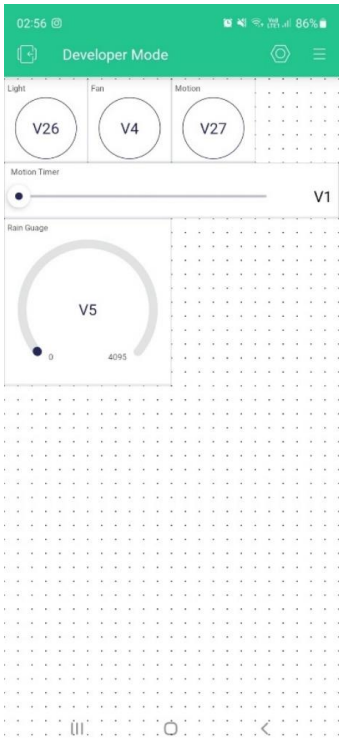


Figure 8: Customising Mobile Dashboard in Blynk App

Finally, replicate the design from the Web Dashboard to the Mobile Dashboard and assign each datatype to their respective widget. The smartphone is now able to communicate and synchronise with the Web Dashboard over Blynk Server.

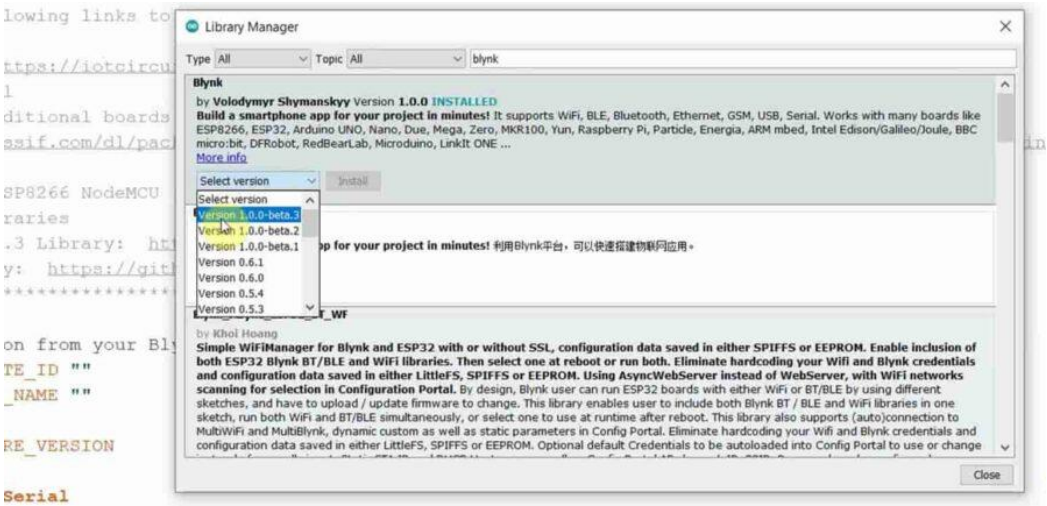


Figure 9: Importing Blynk Library to hardware

To use Blynk functions in the hardware, the Blynk library has to be installed and imported into Arduino IDE.

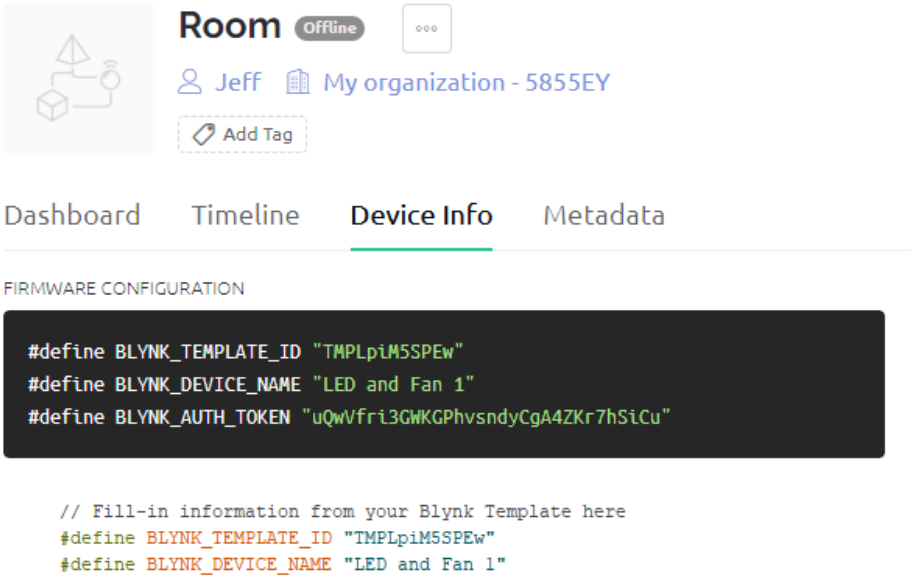


Figure 10: Setting up Blynk for hardware

The Blynk template ID and device name, which can be retrieved from the template info, are then updated into the hardware code. The template ID and device name will be used by the hardware to authorise and authenticate itself to the Blynk server for data exchange.

Figure 11: Linking hardware to Blynk Server and Blynk App

After the hardware code is compiled and uploaded, the final step is to link the hardware to Blynk Server using Blynk App over Wi-Fi connection.

3.2.2 Google Vision API

The Google Vision API allows developers to easily integrate vision detection features within applications, including image labelling, face, and landmark detection, optical character recognition (OCR), and tagging of explicit content [12].

The diagram below illustrates how Google Vision API process and identify images:

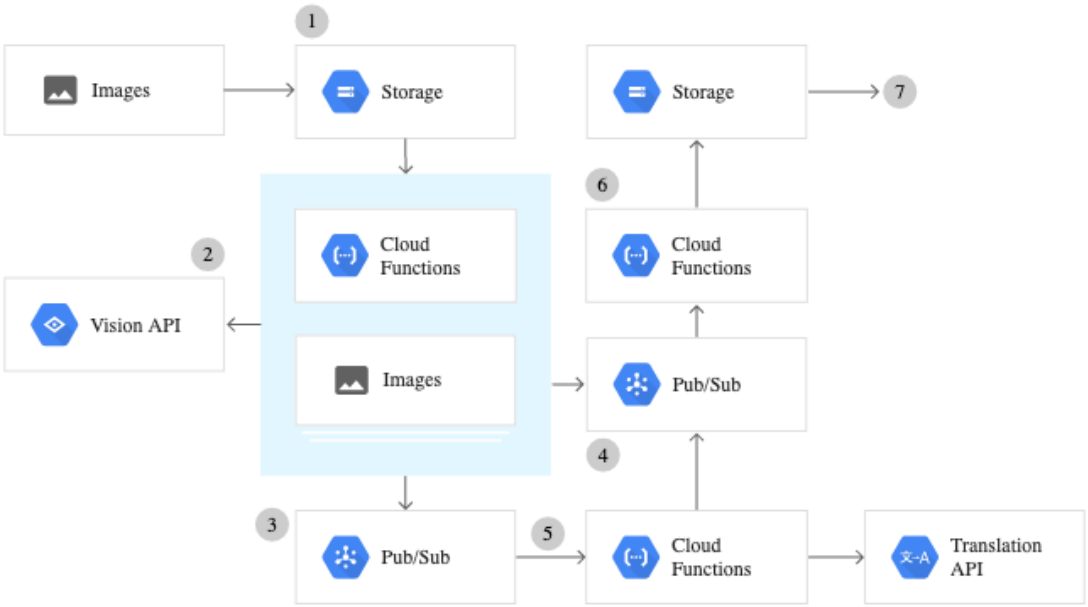


Figure 12: Google Vision API image processing flow process [13]

When a picture is taken and sent to the Google Vision API, the image is processed remotely on Google Cloud. The result is then produced in JSON format file and returned as an output to a function.

In this project, Google Vision API was used on ESP32-CAM to develop an AI-integrated home monitoring system. The image is taken by ESP32-CAM and sent to Google Vision API using Wi-Fi for object detection. The results were then used to notify and alert events happening at home such as parcel delivery, water leakage, fire outbreak, etc.

3.2.3 Telegram Bot API

The Telegram Bot API is an HTTP-based interface for creating bots for Telegram. Users can interact with the bot by sending messages, commands, and inline requests. The bot can be configured to send customised notifications and news, integrate with third-party online services (Gmail Bot, GithubBot, etc.), and many more. Refer to **Appendix C** for a list of Telegram Bot features.

The diagram below represents Telegram Bot API functionality in home automation:

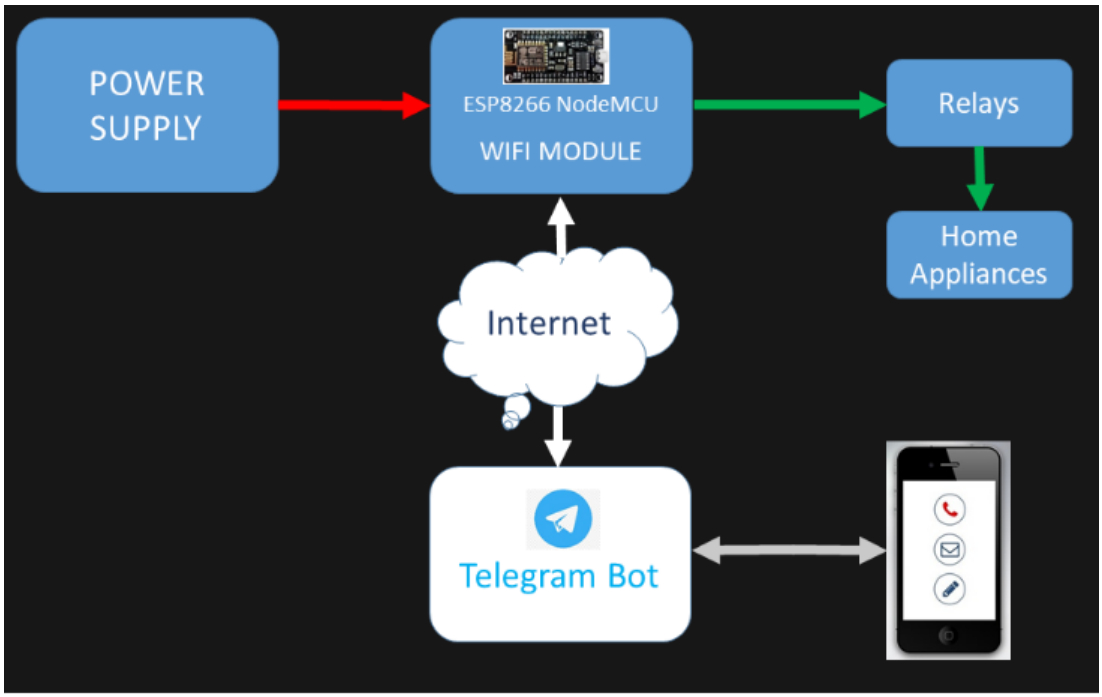


Figure 13: Telegram Bot API on Home Automation [14]

In this project, the Telegram Bot API was for sending notification alerts and displaying image results processed by Google Vision API to the user.

4. Design & Implementation

4.1 Room Simulation

The Room Simulation aims to control household electronics and receive sensor readings remotely via smartphone. This is done using ESP32 as the controller and Blynk as the server for graphical interface and data communication.

4.1.1 Circuit Diagram

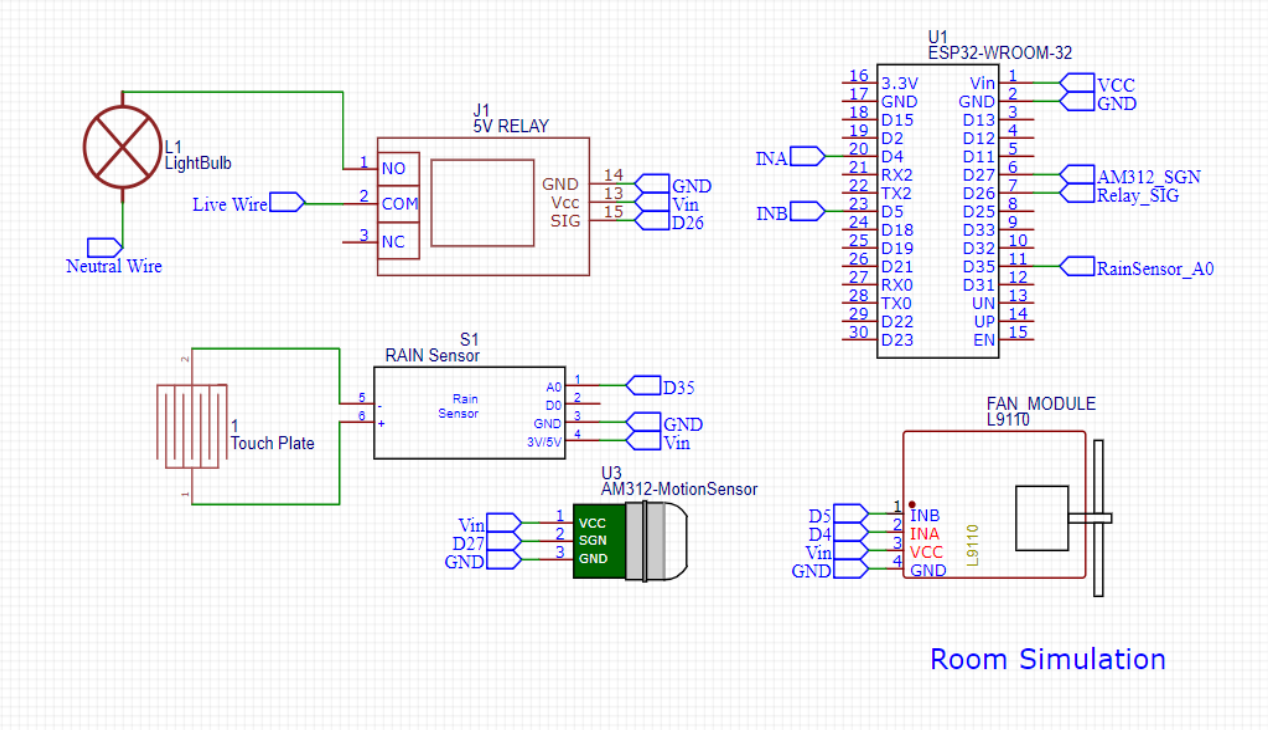


Figure 14: Room Simulation Circuit Diagram

In this design, light and fan were chosen as the household appliances to be controlled using a 5V single channel relay. The fan in this design was replaced with a fan module to substitute an actual ceiling fan. A rain sensor was used for the simulation of reading sensor data and a motion sensor was used for an alternate means of automation.

4.1.2 Flowchart Diagram

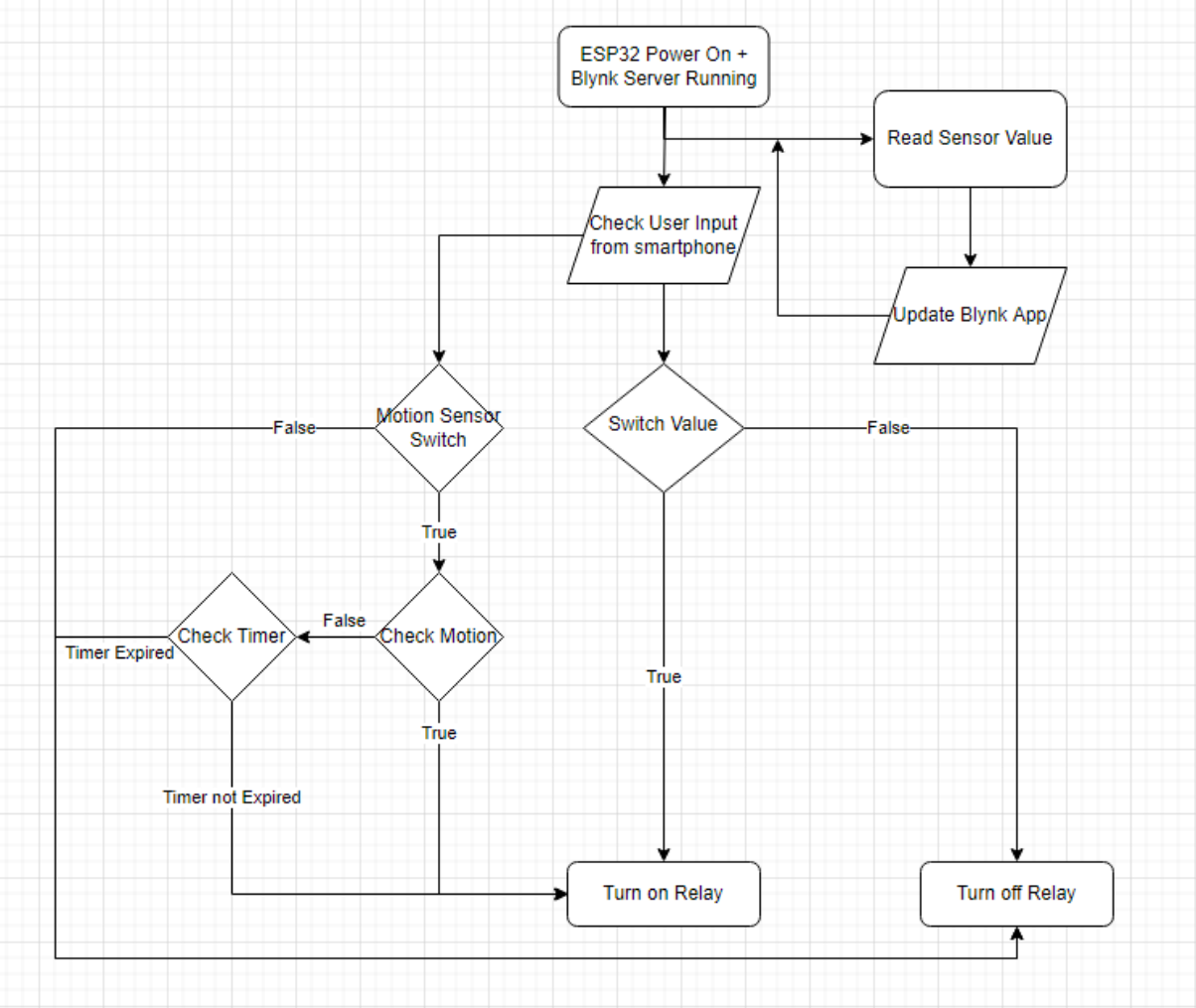


Figure 15: Room Simulation Flowchart Diagram

Upon boot up, ESP32 will establish a connection to the Blynk server via Wi-Fi connection. Blynk will then update the latest user input to the microprocessor using the imported Blynk Library function called *Blynk_Write(PinNumber)*. Based on the assigned pin number, the respective hardware device will be enabled. For instance, referring to Figure 14: Room Simulation Circuit Diagram, if pin 26 is high, the relay circuit will be closed, thus enabling the light bulb to power on.

For reading and updating sensor data to the Blynk server, a different function called *Blynk.virtualWrite(PinNumber, SensorValue)* was called. For examples of the Blynk function used, refer to **Appendix D**.

4.2 Door Simulation

The Door Simulation aims to replace traditional bulky keys with a more compact and convenient method of premises access while retaining home security. Similar to the Room Simulation design, ESP32 was used as the logic controller and Blynk as the server for the data hub. The ESP32-CAM was added to provide a more intelligent mode of access using facial recognition.

4.2.1 Circuit Diagram

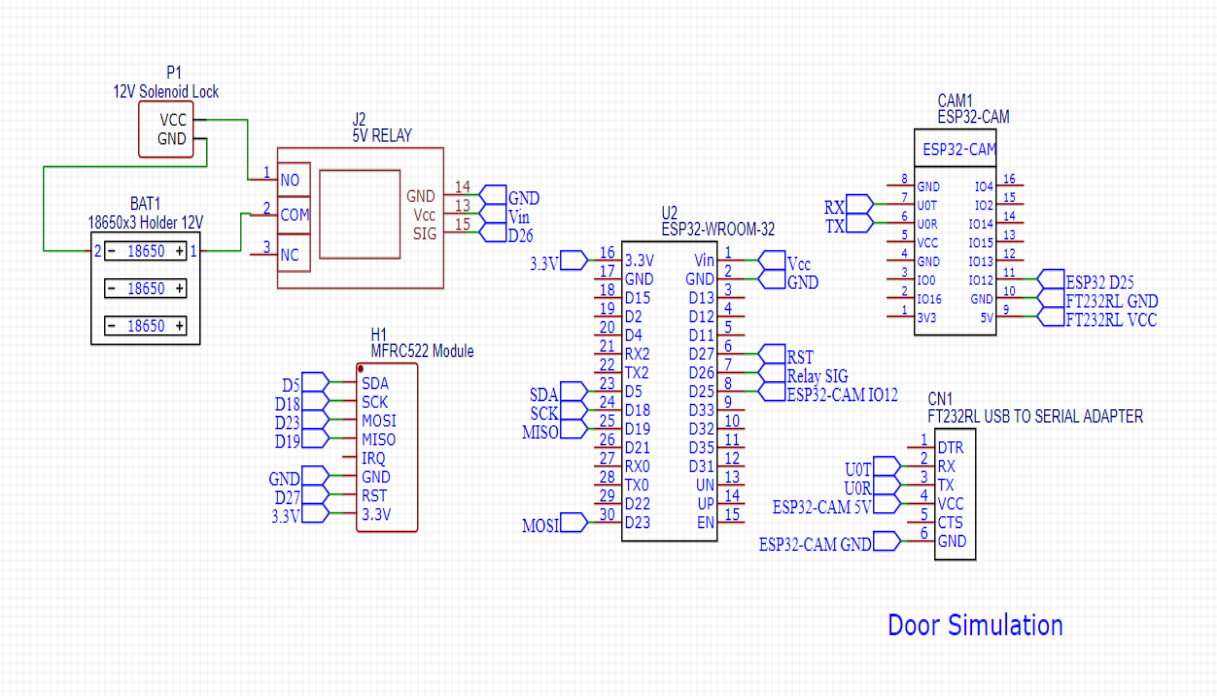


Figure 16: Door Simulation Circuit Diagram

In this design, a 12V DC solenoid lock was used to represent the door mechanism. It is powered by three 18650 batteries and controlled using a 5V single channel relay. Instead of keys, the RC522 RFID was selected as the mode of access via physical material.

The FT232RL USB-TTL Converter was connected to the ESP32-CAM for power and software configuration as the camera does not have a port available for direct USB connection.

4.2.2 Flowchart Diagram

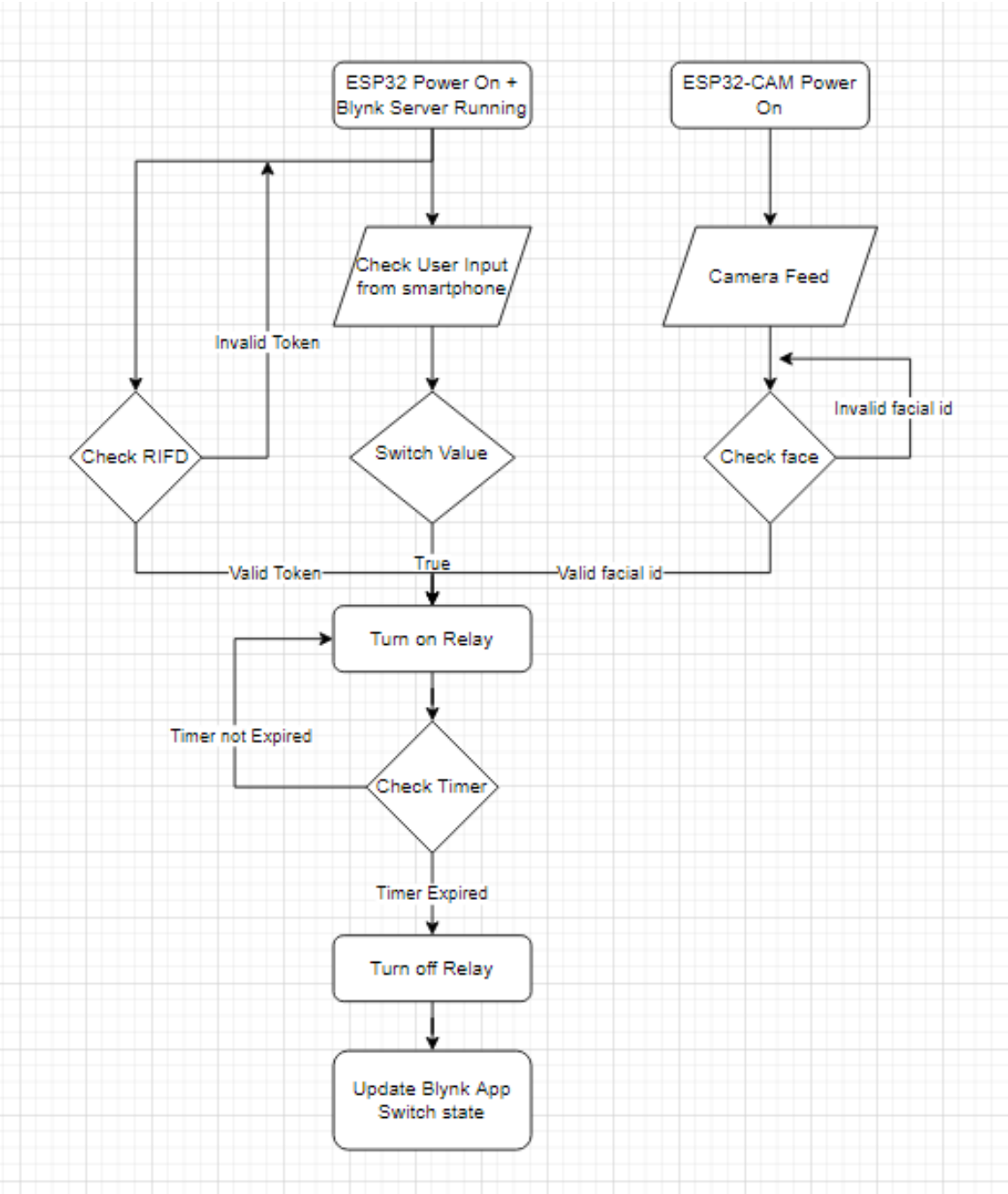


Figure 17: Door Simulation Flowchart Diagram

The controlling of door lock mechanism using Blynk on smartphone followed the design from Room Simulation where the imported Blynk library function, *Blynk_Write(door)* was used to control the state of the relay module for the lock. When the lock timer runs out, the door returns back to locked state and *Blynk.virtualWrite(door, value)* function is called to update the Blynk Server.

4.2.3 RFID Reader

```
byte keyTagUID[4] = {0x13, 0xB9, 0xFD, 0x16};//13 B9 FD 16
```

Figure 18: Registering RFID Token into program

For the RFID token to be detected as valid, the key tag UID of the token has to be manually registered into an array of the code before compiling.

```
if ((rfid.uid.uidByte[0] == keyTagUID[0] &&
    rfid.uid.uidByte[1] == keyTagUID[1] &&
    rfid.uid.uidByte[2] == keyTagUID[2] &&
    rfid.uid.uidByte[3] == keyTagUID[3])) {
  Serial.println("Access is granted");
  door=1;
  Blynk.virtualWrite(V5, "1");
  digitalWrite(RELAY_PIN, LOW);  // unlock the
  delay(5000);
  digitalWrite(RELAY_PIN, HIGH); // lock the d
  Blynk.virtualWrite(V5, "0");
  door =0;
```

Figure 19: Logic Process of RFID Reader

When the RFID reader detects a token, the reader will check the array of registered UID for verification. If the UID is valid, the Blynk server will be updated, and the door unlocks for 5 seconds before returning back to lock state.

4.2.4 ESP32-CAM Facial Recognition

The ESP32-CAM AI-Thinker board comes with a built-in feature to perform facial recognition.

```
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.1.131' to connect
```

Figure 20: Web server hosting for livestream of ESP32-CAM

Upon booting, the ESP32-CAM will generate an IP address to provide a live feed of what the camera is capturing.

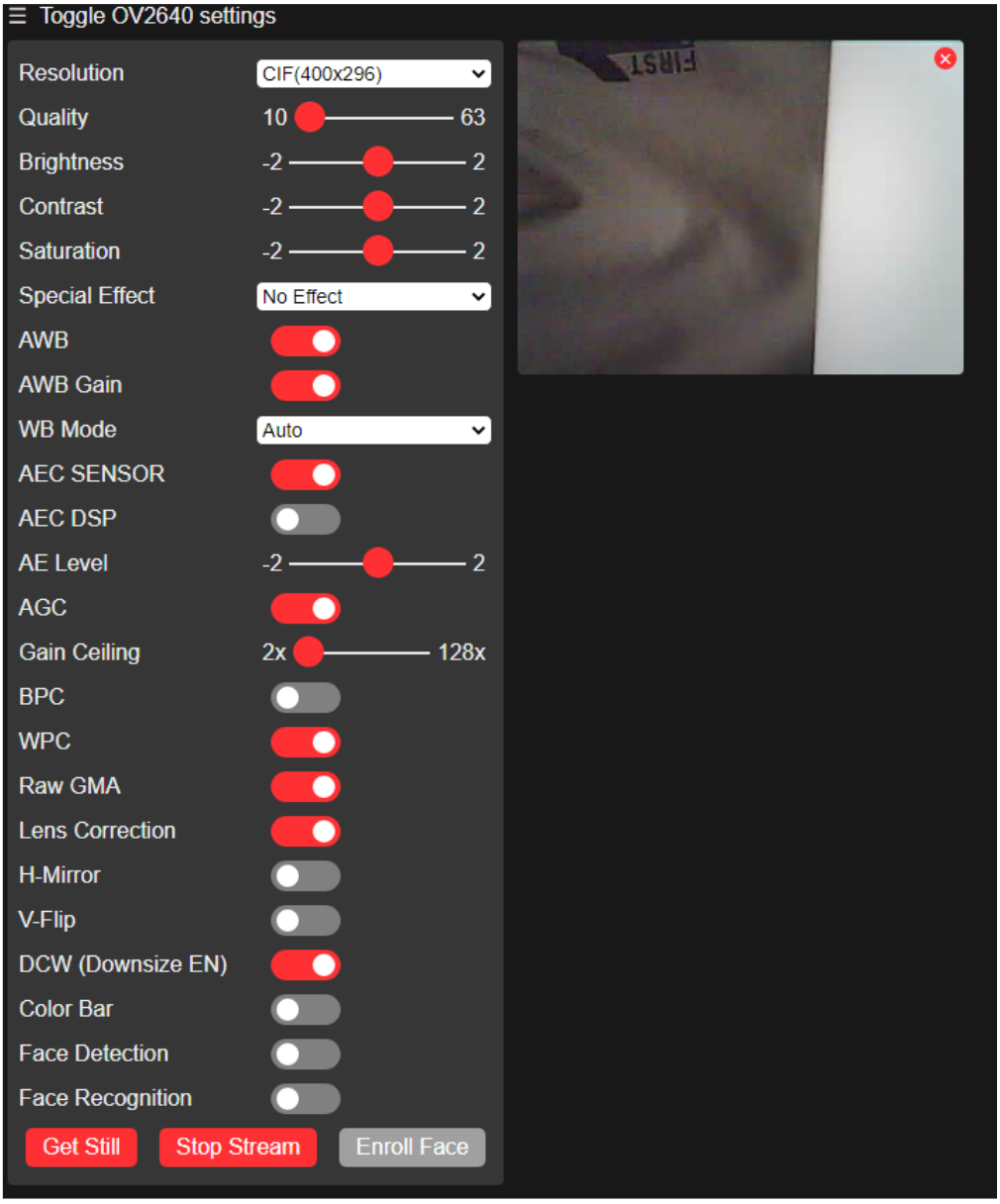


Figure 21: ESP32-CAM Web server user interface

Upon accessing the web server, the Face Detection and Face Recognition feature of the ESP32-CAM can be enabled on the hosted web server to perform facial recognition. When a face is detected, the camera will capture the face image matrix and run a face recognition function to compare it against a list of enrolled face image matrix to look for a match. When a match is found, the function will return a Boolean value True, otherwise False. The returned value is then used to control the state of the door lock.

4.3 Monitoring System with Object Detection

In traditional a surveillance system, the camera is actively recording the entire time regardless of an event being present or not. This not only requires a large amount of storage data but power consumption too. This monitoring system uses Google Vision API to detect desired objects around the home. The camera works passively and will only capture an event based on sensors being triggered. Ultimately, this implementation avoids the need for large data storage space and also reduces power consumption.

4.3.1 Circuit Diagram

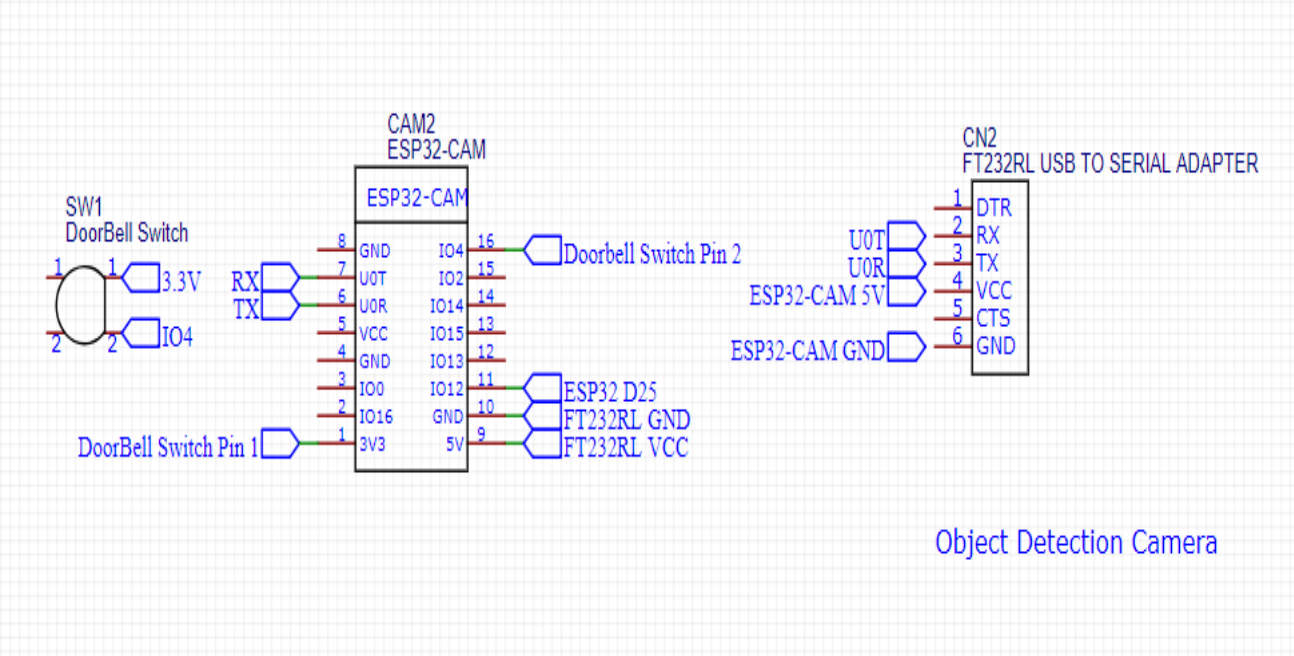


Figure 22:Object Detection Camera Circuit Diagram

In this design, a pushbutton was used as the input to represent an event of a doorbell being pressed when there is a delivery. Similar to the door simulation design, a FT232RL USB-TTL Converter was needed to power and transfer data to the ESP32-CAM.

4.3.2 Flowchart Diagram

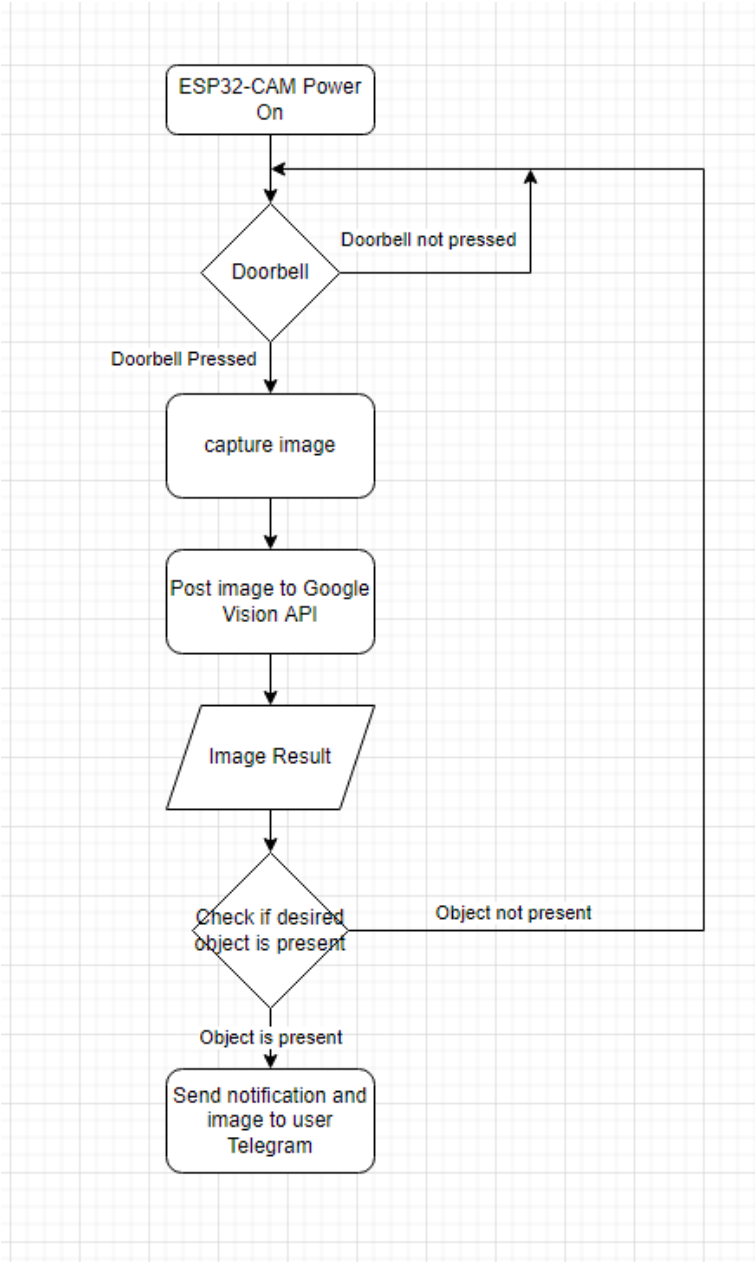


Figure 23: Object Detection Camera Flowchart Diagram

When the doorbell is pressed, the ESP32-CAM camera will capture an image and send it to a NodeJS server hosted by a computer. This NodeJS server will then post the image to Google Vision API for processing. The API then return the results as an array of labels to the server and from the server, these labels are sent to the ESP32-CAM. The labels are then checked to see if it contains the desired object to be detected (“box” or “parcel” in this case). Once detected, ESP32-CAM sends a notification to Telegram via a Telegram Bot API.

4.3.3 NodeJS Server for Google Vision API

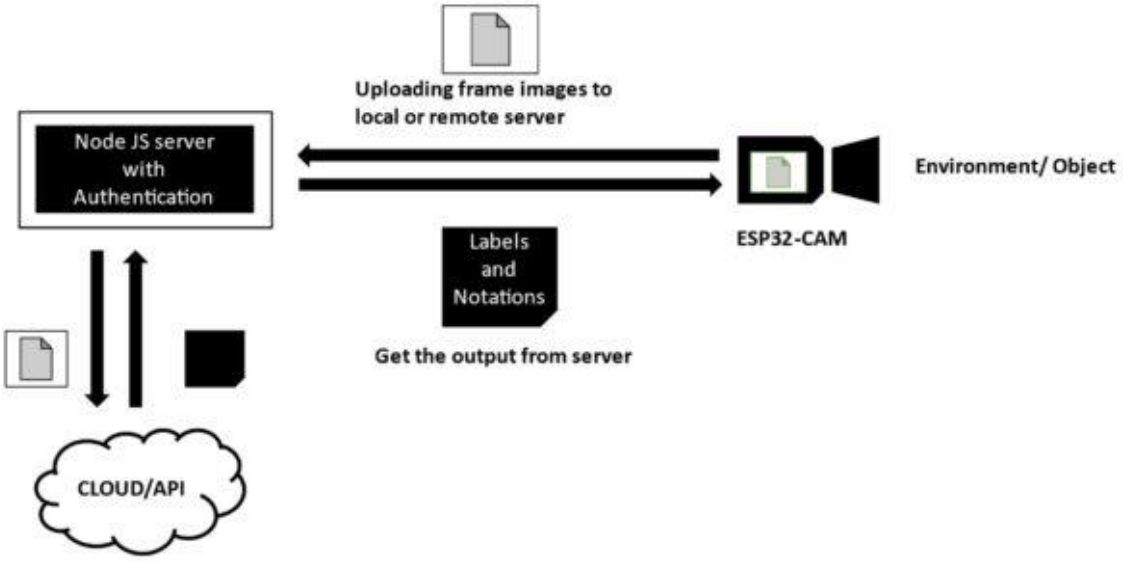


Figure 24: Image transfer process from Camera to-and-fro Google Vision API

The **ArduinoJson** and **HTTPClient** libraries are downloaded and imported into Arduino IDE code to enable the camera to communicate with the NodeJS server. Figure 24 above illustrates the flow process of the image capture by the camera.

```
void postingImage(camera_fb_t *fb){
  HTTPClient client;

  client.begin("http://192.168.1.20:8888/imageUpdate"); //
  //client.begin("http://10.27.140.142:8888/imageUpdate");
  client.addHeader("Content-Type", "image/jpeg");
  int httpResponseCode = client.POST(fb->buf, fb->len);
  if(httpResponseCode == 200){
    String response = client.getString();
    parsingResult(response, fb);
```

Figure 25: Posting image to server using HTTP

As shown in Figure 25, the image captured by the camera is sent to the NodeJS server using HTTP post. The IP address in this function is the IP address of the computer hosting the server. After sending the image to the server, the function will then attempt to request the server for the results in a form of a String datatype.

```
C:\Users\ooiko\Desktop\objectdetection\server>set GOOGLE_APPLICATION_CREDENTIALS=C:\Users\ooik
o\Desktop\objectdetection\key.json

C:\Users\ooiko\Desktop\objectdetection\server>node server.js
Listening at 8888
```

Figure 26: Setting Google Vision API Key path

Shown in Figure 26, before hosting the NodeJS server, the path of the Google Vision API Key has to be set to enable the server to find and authenticate itself to Google Vision API. The server is then ready to run and starts listening for a post.

```
server.on('request', (request, response)=>{
    if(request.method == 'POST' && request.url === "/imageUpdate"){

        var ImageFile = fs.createWriteStream(filePath, {encoding: 'utf8'});
        request.on('data', function(data){
            ImageFile.write(data);
        });

        request.on('end',async function(){
            ImageFile.end();
            const labels = await labelAPI();
            response.writeHead(200, {'Content-Type' : 'application/json'});
            response.end(JSON.stringify(labels));
        });
```

Figure 27: NodeJS server listen for HTTP Post

With reference to Figure 27, the server will listen for HTTP post from ESP32-CAM. The received image data will be saved in a file path declared by the user as shown in Figure 28.

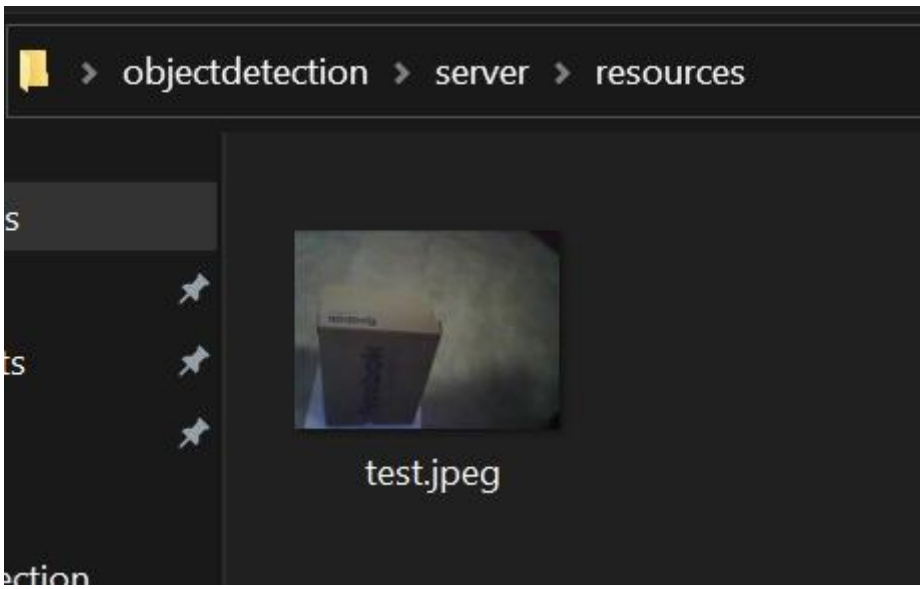


Figure 28: Image saved in file path declared by user

```
async function labelAPI() {
  var o = [];
  // Imports the Google Cloud client library
  const vision = require('@google-cloud/vision');

  // Creates a client
  const client = new vision.ImageAnnotatorClient();

  // Performs label detection on the image file
  const [result] = await client.labelDetection(filePath);
  const labels = result.labelAnnotations;

  labels.forEach(label => {
    o.push({description: label.description, score: label.score});
  });
  return o;
}
```

Figure 29: Function to call Google Vision API and receive results

After receiving the image, the server calls the function *labelAPI()* to run Google Vision API. The image will be processed in Google Cloud and the results returned are the object name identified and the confidence score.

4.3.4 Notification system with Telegram Bot API

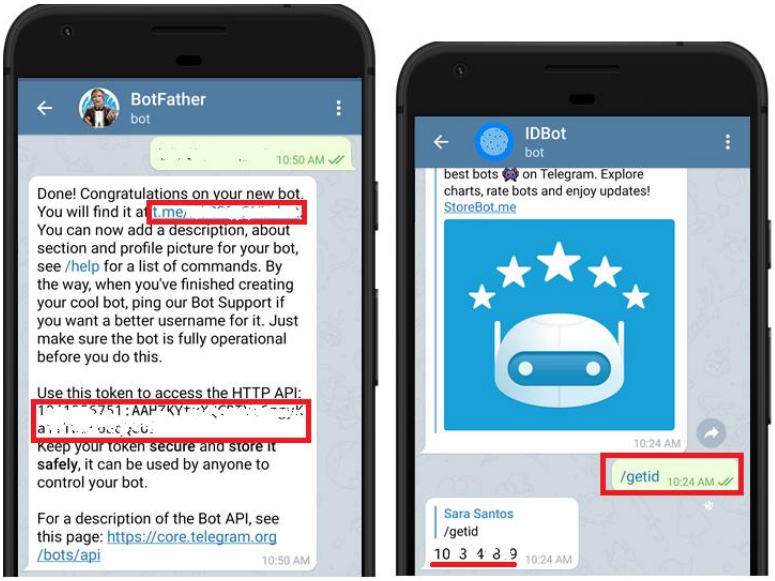


Figure 30: Getting Bot and User ID for Telegram Bot set up [15]

To set up a Telegram Bot, a bot token and telegram chat ID of the recipient has to be declared. Both bot and user credentials can be retrieved from the Telegram APP itself by interacting with BotFather and IDBot respectively as shown in Figure 30.

```
DynamicJsonDocument doc(1024);
deserializeJson(doc, response);
JsonArray array = doc.as<JsonArray>();

for(JsonVariant v : array){
  JsonObject object = v.as<JsonObject>();
  const char* description = object["description"];
  float score = object["score"];
  String label = "";
  //String String(description)
  label += description;
  if (label=="Parcel" || label == "Box")
  {
    Serial.println("Delivery detected");
    bot.sendMessage(CHAT_ID, "Delivery detected!!", "");
```

Figure 31: Reading Json file and sending message to Telegram

The previously fetched results from NodeJS server can be accessed on ESP32-CAM using the ArduinoJson library. If “box” or “parcel” is present, a notification message is sent using the sendMessage function available from UniversalTelegramBot library. Refer to **Appendix E** for documentation of the function.

4.4 Touch Screen Remote Controller

The touch screen remote controller allows interaction of home devices without having to open up the Blynk APP. This was done using a screen capable of sensing touch input and Blynk Cloud HTTP API.

4.4.1 Circuit Diagram

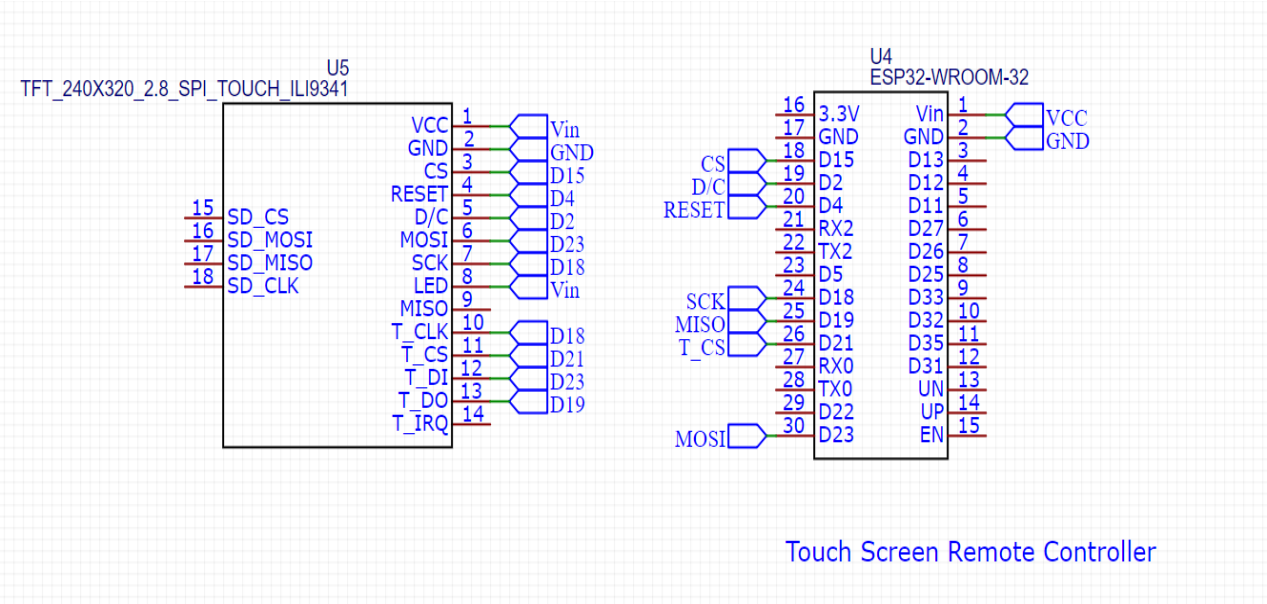


Figure 32: Touch Screen Remote Controller Circuit Diagram

In this design, a 2.8” ILI9341 240x320 TFT SPI LCD Display Touch Panel was used to sense user touch input as well as to display graphical user interface. The ESP32 powers the LCD and the display logic of the user interface on the front end. At the backend, the ESP32 will communicate with the Blynk Server to receive and send data.

4.4.2 Flowchart Diagram

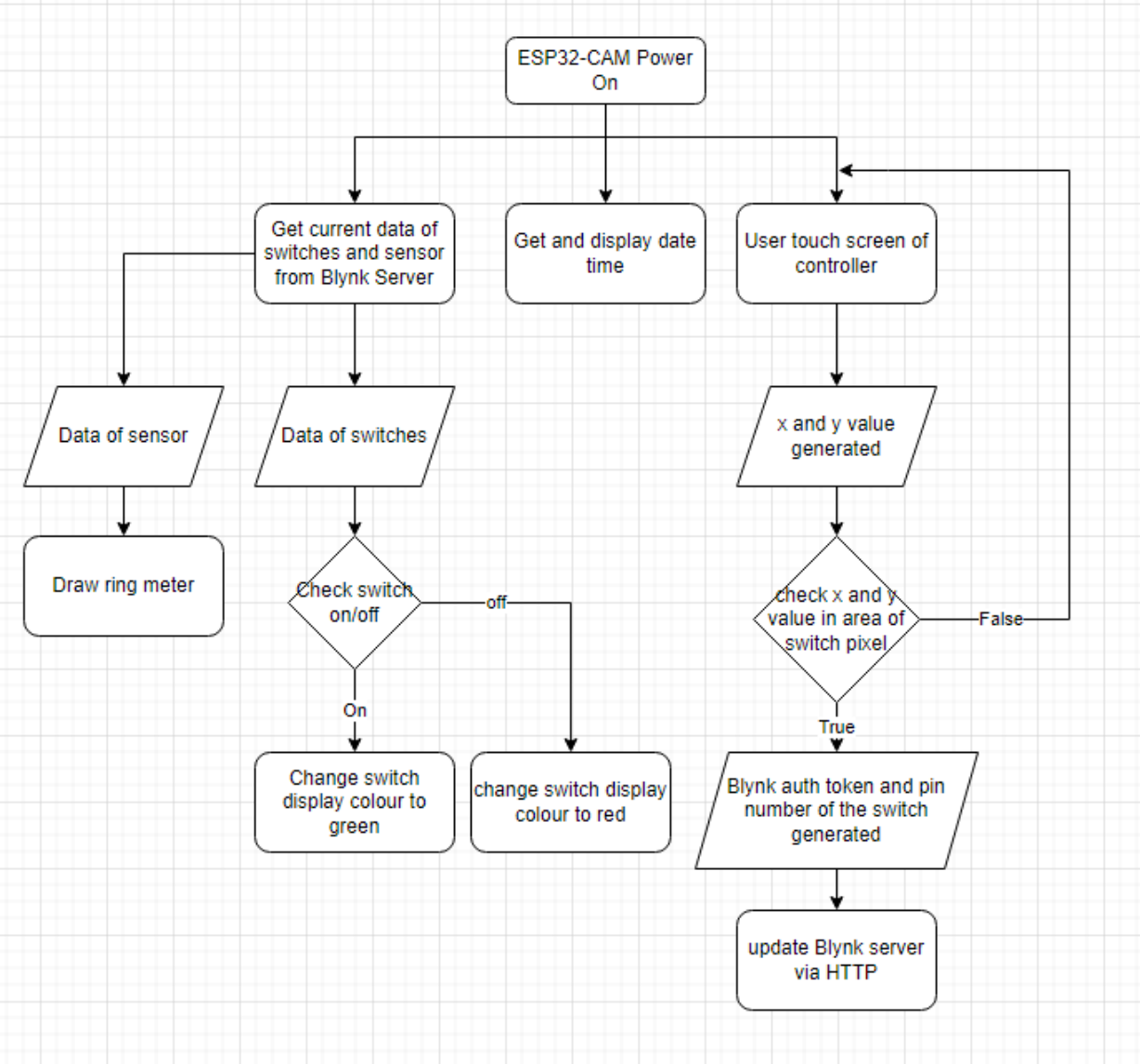


Figure 33: Touch Screen Remote Controller Flowchart Diagram

Whenever the LCD display senses a touch input, the program will check to see if the x and y coordinates of the input reside in the area of any switches. If it falls inside one of the switches, that respective switch will toggle its current state and generates a Blynk authentication token, pin number, and a value. These data values generated were then used to communicate to the Blynk server for synchronisation of that switch.

While performing the previous task, the remote controller is also actively seeking the latest switch values from the Blynk server to maintain an accurate display.

4.4.3 User Interface Design on ILI9341 LCD Display

Figure 34: ILI9341 LCD Display coordinates layout

With reference to Figure 34 above, the dimension of the LCD display is 320 by 240 pixels. The x and y-coordinates represent the position of each pixel on the display.

To draw an image on the LCD display, the TFT_eSPI and SPI library has to be imported into the Arduino code. The TFT_eSPI library was created by Bodmer specifically for 32-bit processors such as the ESP32 to control the ILI9341 LCD Display [16].

```
drawXBitmap(int16_t x, int16_t y, const uint8_t *bitmap, int16_t w, int16_t h, uint16_t fgcolor),
drawXBitmap(int16_t x, int16_t y, const uint8_t *bitmap, int16_t w, int16_t h, uint16_t fgcolor, uint16_t bgcolor),
```

Figure 35: drawXBitmap function for ILI9341 LCD display

The drawXbitmap function is called to draw images onto the display. As shown in Figure 35, “x” and “y” input arguments are the starting coordinates to draw from. The argument “*bitmap” contains the image to be drawn in XBM format. The “w” and “h” arguments are there to set the width and height of the image. “fgcolor” is the foreground colour while “bgcolor” is the background colour of the image. In this project, the drawXbitmap function was used to draw button images onto the LCD display.

4.4.4 Converting Image to XBM format

To draw an image on the LCD display, the image has to be converted into XBM format first using a converter. This image format converter can be found online. An example of a converter is shown in Figure 36 below.

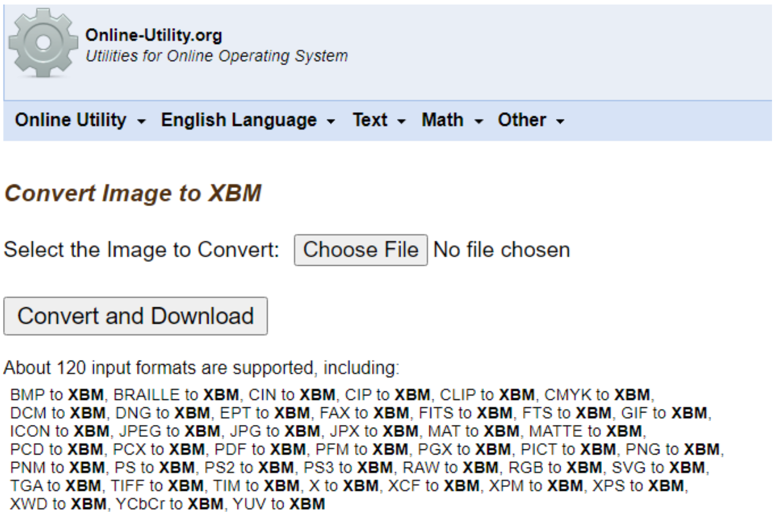


Figure 36: Example of online image converter to XBM format [17]

After converting, the image will now be represented in an array of hex as shown in Figure 37 below.

```
#define 1646763488763_width 50
#define 1646763488763_height 36
static char 1646763488763_bits[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xF0, 0x03, 0x1F, 0xF8, 0xF1, 0x03, 0x00, 0xF0, 0x87, 0x7F, 0xFC, 0xF1,
  0x07, 0x00, 0x30, 0xCE, 0x71, 0x8E, 0x73, 0x06, 0x00, 0x30, 0xDC, 0x60,
  0x06, 0x33, 0x06, 0x00, 0x30, 0xDC, 0xE0, 0x06, 0xF3, 0x03, 0x00, 0x70,
  0xDC, 0x60, 0x06, 0xF3, 0x03, 0x00, 0x30, 0xCC, 0x60, 0x06, 0x33, 0x07,
  0x00, 0x30, 0xCE, 0x71, 0x8E, 0x33, 0x06, 0x00, 0xF0, 0x87, 0x3F, 0xFE,
  0x31, 0x06, 0x00, 0xF0, 0x03, 0x1F, 0xF8, 0x30, 0x0E, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  };
```

Figure 37: Example of an image file in XBM format

This array is then declared in a header file as a reference for the program to draw.

As for displaying the sensor readings on the LCD display, the function made by Bodmer, ringMeter, was used.

```
int ringMeter(int value, int vmin, int vmax, int x, int y, int r, char *units, byte scheme)
```

Figure 38: ringMeter function for sensor reading display

With reference to Figure 38, “value” is the sensor value to be displayed and plotted. “vmin” and “vmax” specifies the minimum and maximum range of the value respectively. “x” and “y” specify the starting coordinates to draw from. “r” is the radius of the ring meter, “Units” to represent the unit of measurement and “scheme” to set the colour scheme of the ring meter.

Figure 39: Example of sensor reading display using ring meter by Bodmer [18]

4.4.5 Blynk Cloud HTTP API

The Blynk Cloud API was the main contributing factor to the idea of the remote controller design. Initially, Blynk only allows up to two devices to be connected for free and monthly subscription will be required if more devices were to be added. With the API, any device can communicate with the server indirectly through HTTP request as long as there is a valid Blynk authentication token.

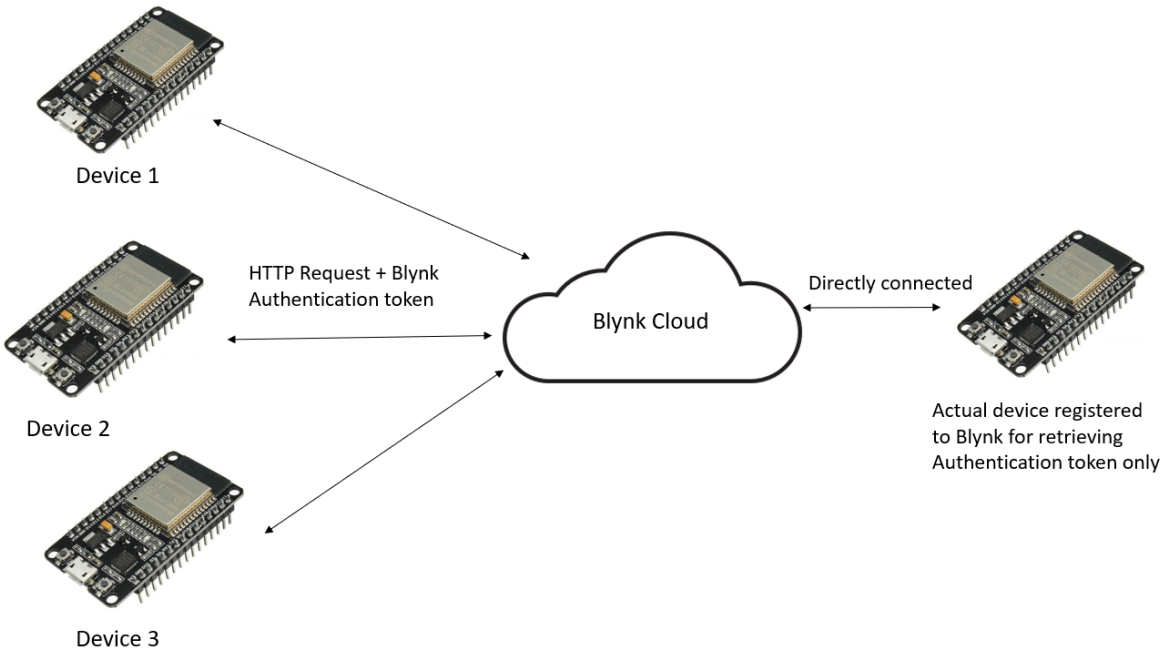


Figure 40: Diagram of multiple ESP32s communication to Blynk Cloud

Thus, as shown in Figure 40, only one template and ESP32 is needed to retrieve the necessary credentials for the initial setup. The other ESP32s will only need to make use of the original ESP32 authentication token to communicate with the Blynk server via HTTP request.

Using the Blynk Cloud HTTP API, to get data values from the Blynk server, the following HTTP request is made as shown in Figure 41 below.

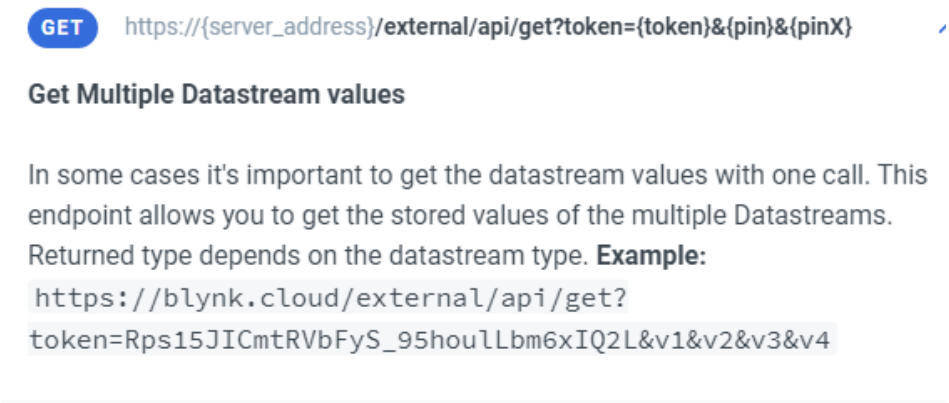


Figure 41: HTTP request to get data values using Blynk HTTP API [19]

To update values to the Blynk server, Figure 42 below shows the HTTP request to be made.

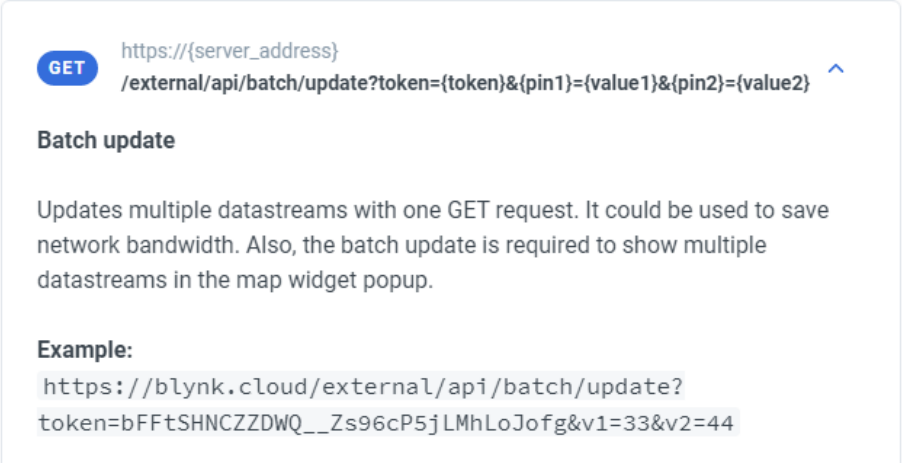


Figure 42: HTTP request to update data values using Blynk HTTP API [20]

The description of the parameters in the HTTP request is shown in Table 2 below:

Table 2: Parameters of the HTTP request

Parameters	Description	Data Format
(pin)	The virtual pin number	String
(token)	Blynk authentication token	String
(value)	The value of the data	int, double, String

5. Testing and Results

5.1 Room Simulation

5.1.1 Controlling Light and Fan on Blynk App

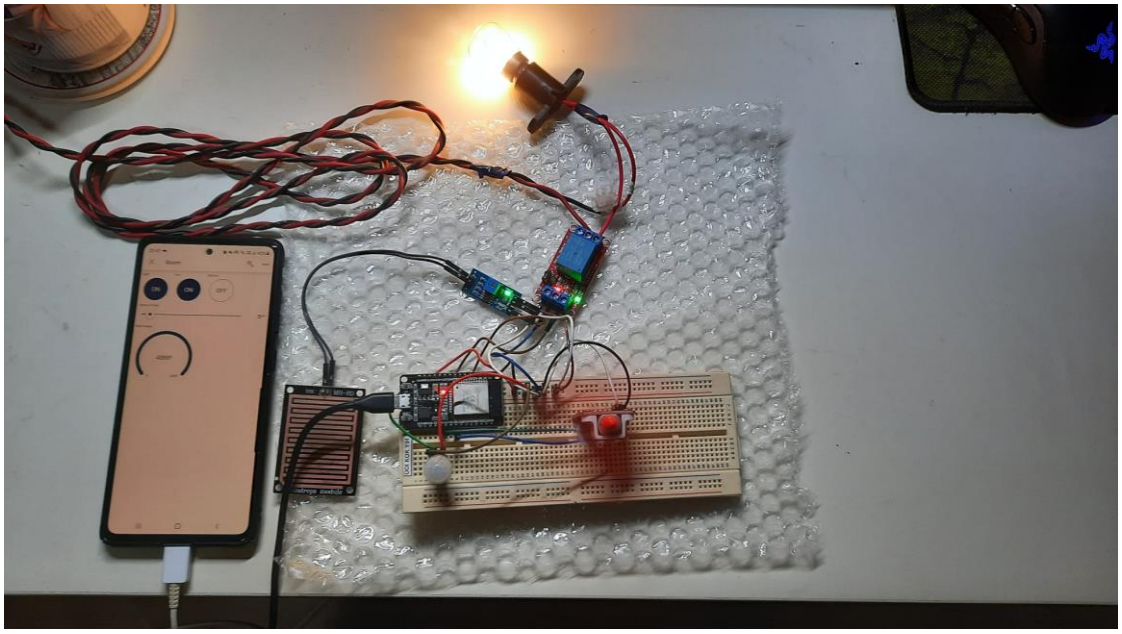


Figure 43: Picture of Light and Fan control via App

5.1.2 Using Motion Sensor to control Light and Fan

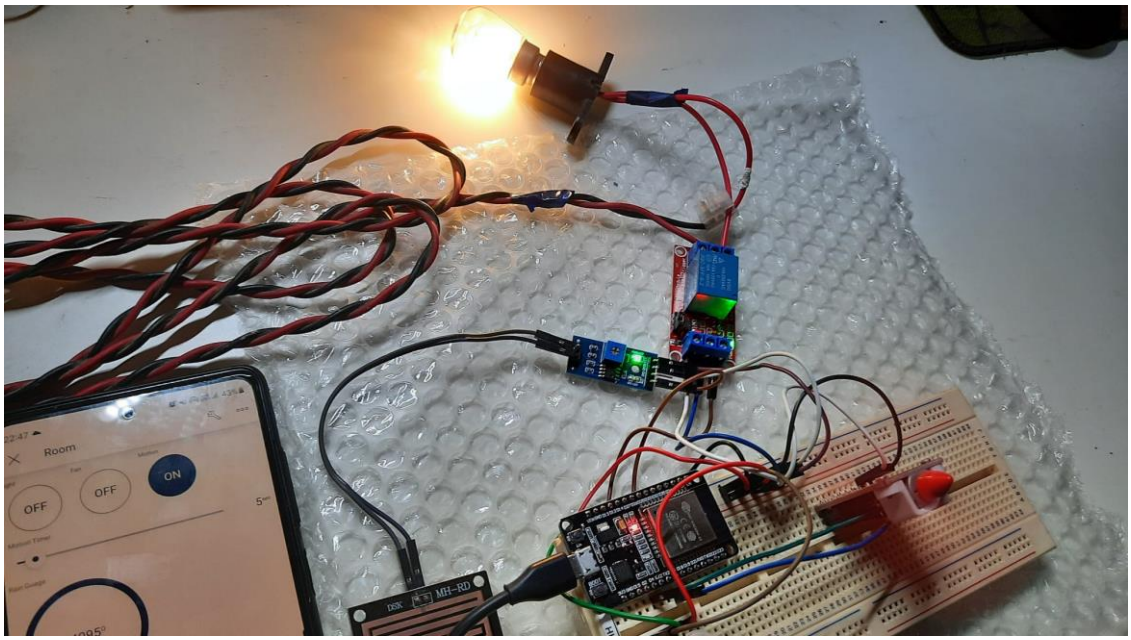


Figure 44: Picture of using Motion Sensor to on Light and Fan

5.1.3 Reading Rain Sensor on Blynk App

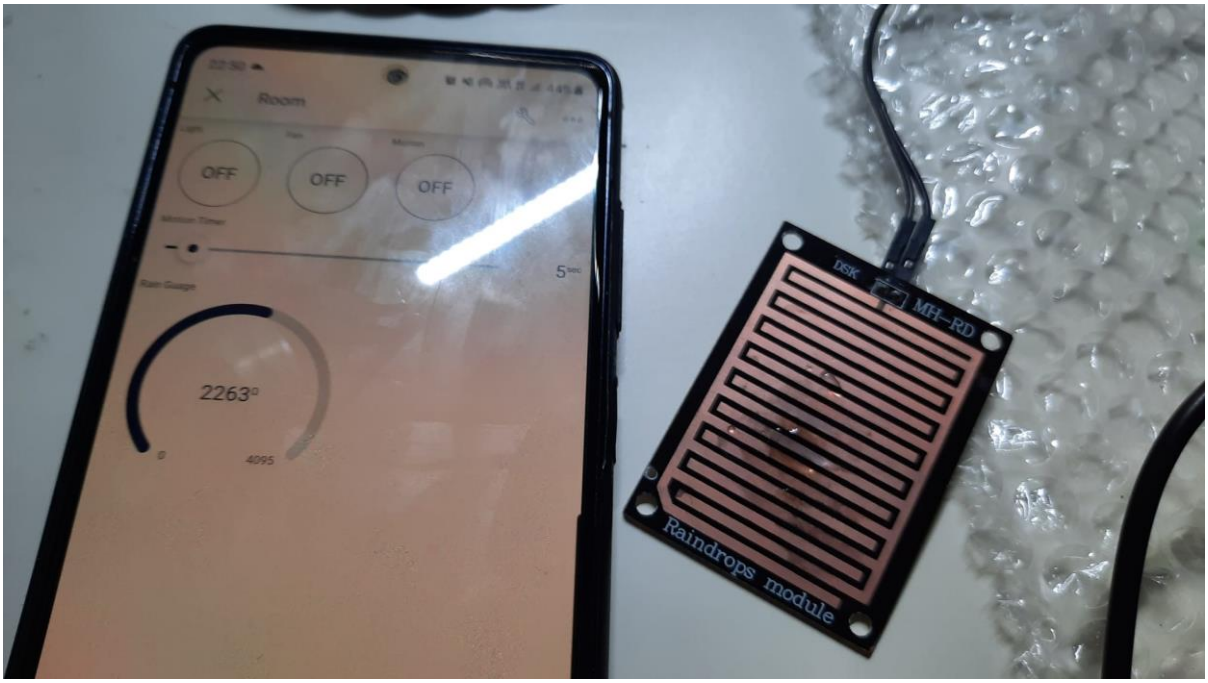


Figure 45: Picture of Blynk App reading Rain Sensor

The light and fan control simulation shows that any non-smart device can indeed be converted into smart devices simply by using a suitable microprocessor with the appropriate logic and the help of relays to control its circuit.

The rain sensor reading also shows that the Blynk App is capable of reading analogue inputs from other types of sensors as well.

5.2 Door Simulation

5.2.1 Opening Door on Blynk App

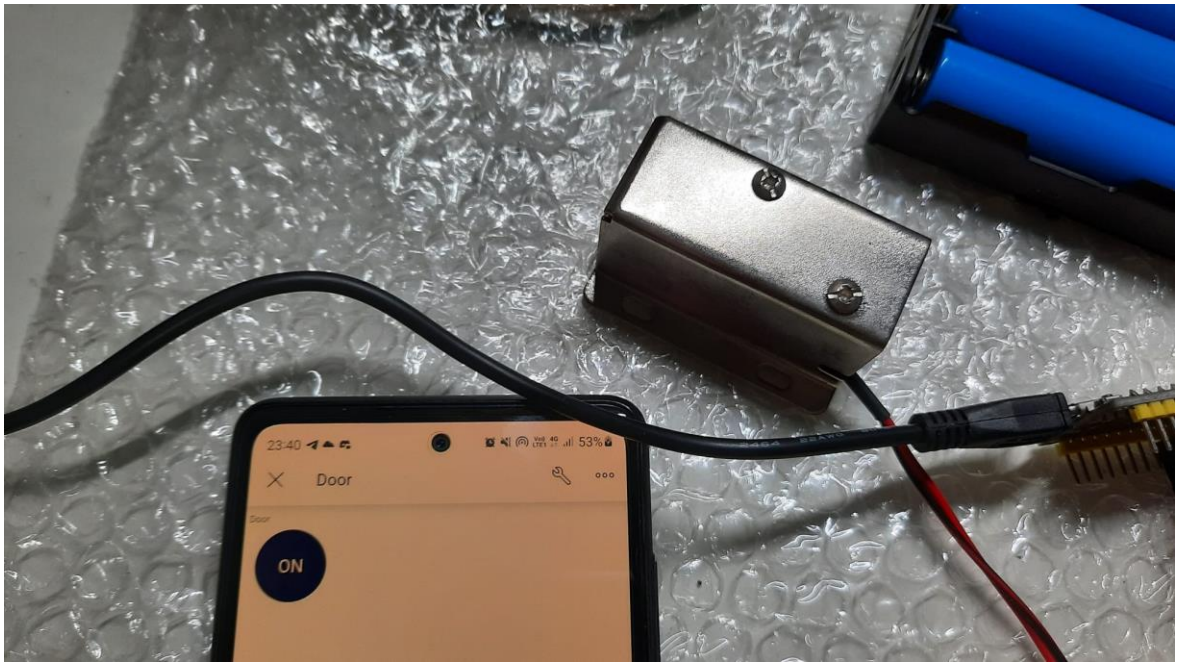


Figure 46: Picture of Door control via Blynk App

5.2.2 Opening Door with RFID Reader

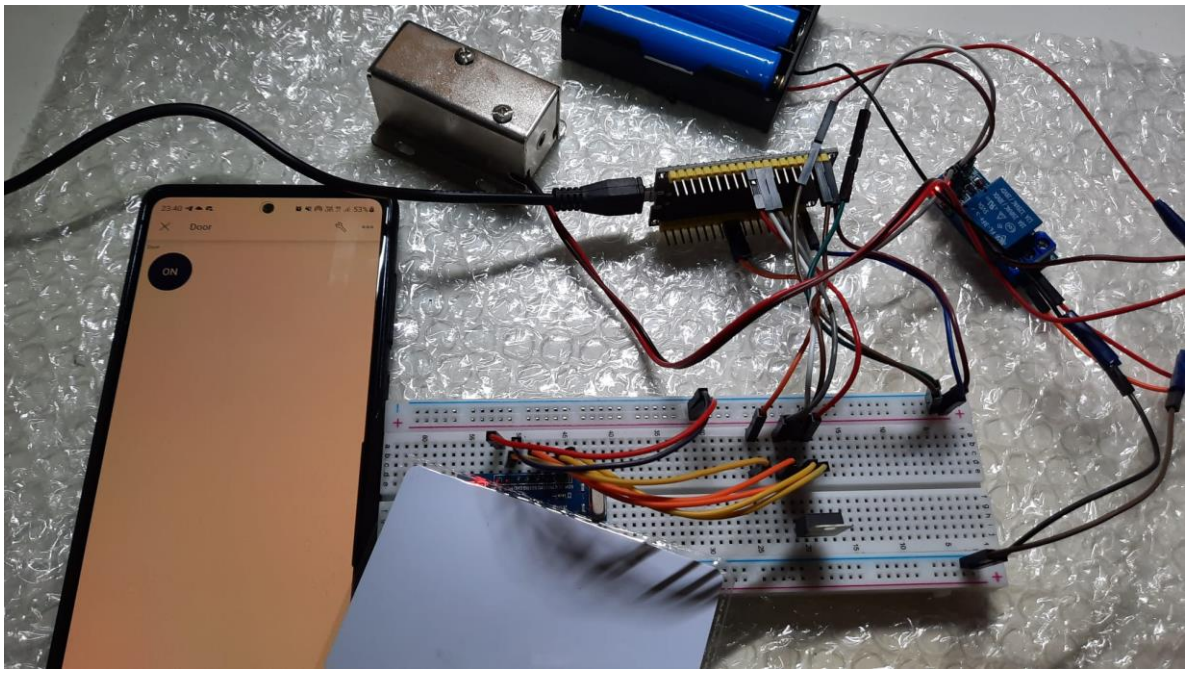


Figure 47: Picture of unlocking Door with RFID Reader

5.2.3 Opening Door with Facial Recognition

Figure 48: Picture of using Facial Recognition to Open Door

The Door Simulation implementation shows that different modes of access can be used to replace traditional keys. These methods of access are much more convenient and faster in terms of authentication and unlocking. It also provides the higher or same level of security as a traditional key.

5.3 Monitoring System with Object Detection

5.3.1 Google Vision API result

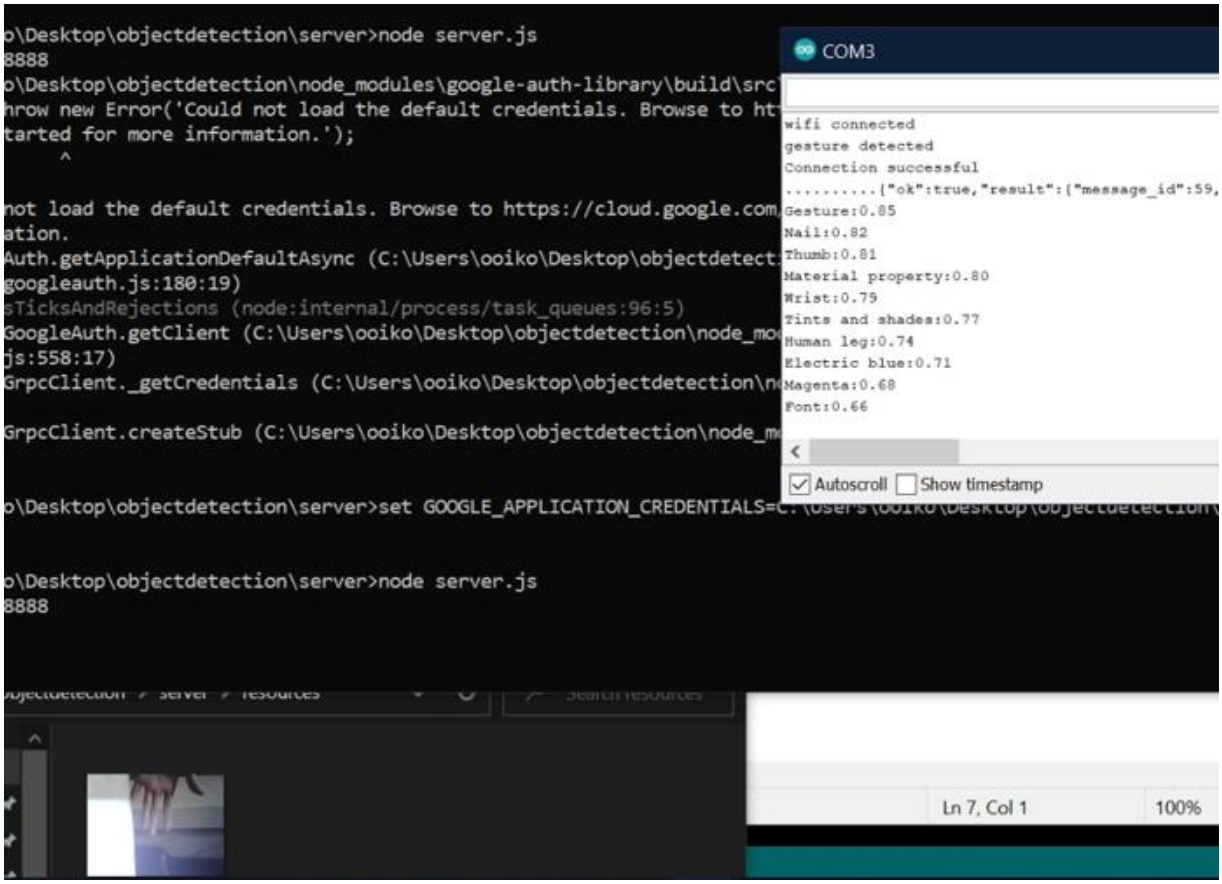


Figure 49: Picture of Google Vision API returning result with confidence score

In this test, the camera was trained to detect a human gesture instead to replace the need for a box prop. The doorbell push button used in this test can also be replaced with other types of sensors to implement a monitoring system for other parts of the house. For instance, a thermistor can be used to trigger the camera to check for an event of a fire outbreak whenever the temperature rises.

5.3.2 Telegram Bot Notification

Figure 50: Picture of desired object detected along with notification on Telegram

The Telegram Bot provides a fast and reliable means of notification on events captured by the monitoring system.

5.4 Touch Screen Remote Controller

Figure 51: Picture of Remote Controller Sending and Receiving data with Blynk

The remote controller does not need to be registered into the Blynk App. It gets and sends data solely using Blynk HTTP API.

6. Conclusion & Future Recommendation

To conclude, this project developed an alternative form of home automation using a mixture of software provided by third-party developers as well as self-developed functions. Though the full features of third-party applications may not be fully accessible to free users, the functionality of smart home devices may not necessarily be affected. This was done by compensating those features with another API to make up for overall performance. One such example was demonstrated on the ILI9341 touch screen remote controller by using HTTP requests to the Blynk server instead of adding the device directly to the Blynk App, which has a limitation of adding up to two devices.

It is also more practical to rely on some of the implementations on third-party software rather than developing from scratch as their software generally tend to be more stable, well-documented with years of development. One such example would be the implementation of the object detection camera, where the image processing was done completely by Google Cloud instead of having to train a model with Tensorflow.

In this project, the ILI9341 touch screen remote controller could be developed further by implementing the design onto an ESP32 watch. Some smartwatches available in the market today do come with Google Assistant for home automation. However, these watches tend to be costly and are not programmable. The ESP32 watch on the other hand is more open to opportunities for IoT-related features as it is fully programmable and cheaper.

7. References

[1] A. S. Gillis, “What is internet of things (IoT)?,” [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT.

[2] K. Lee, “ScienceDirect,” 8 03 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007681315000373#bib0025.

[3] D. J. Lasquety-Reyes, “Number of Smart Homes forecast in the World from 2017 to 2025,” 15 Jun 2021. [Online]. Available: https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world.

[4] M. C. Nathaniel Gyory, “IoTOne: Integrated platform for heterogeneous IoT devices,” 2017 International Conference on Computing, Networking and Communications (ICNC), 29 Jan 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7876230.

[5] IDC, “Worldwide Smart Home Devices Market Grew 10.3%,” 9 December 2021. [Online]. Available: https://tinyurl.com/2p999r8j.

[6] C. Hall, “What is SmartThings and how does it work?,” Pocket-lint, 9 September 2021. [Online]. Available: https://www.pocket-lint.com/smart-home/news/samsung/155496-what-is-smartthings-and-how-does-it-work.

[7] C. Hall, “Samsung's new SmartTag only works with Galaxy devices,” Pocket-lint, 22 January 2021. [Online]. Available: https://www.pocket-lint.com/gadgets/news/samsung/155463-samsung-s-new-smarttag-only-works-with-galaxy-devices-limiting-its-wider-appeal.

[8] A. Martonik, “Google Assistant now works with over 5,000 smart home products,” androidcentral, 23 October 2018. [Online]. Available: https://www.androidcentral.com/google-assistant-now-works-5000-smart-home-products-new-big-brands-now-board.

[9] M. Katerey, “How Does Your Google Assistant Really Work?,” The Startup, 30 November 2020. [Online]. Available: https://medium.com/swlh/how-does-your-google-assistant-really-work-88c64d5dd38d.

[10] august, [Online]. Available: https://august.com/.

[11] Y. Butt, “Blynk working principal,” September 2020. [Online]. Available: https://www.researchgate.net/figure/Blynk-working-principle_fig2_346462878.

[12] M. Cohen, “Using the Vision API,” 26 June 2021. [Online]. Available: https://codelabs.developers.google.com/codelabs/cloud-vision-api-python#0.

[13] T. Cheng, “Introduction to Google Cloud Vision,” [Online]. Available: https://nanonets.com/blog/google-cloud-vision/.

[14] P. Singh, “Telegram Bot,” 13 June 2020. [Online]. Available: https://chatbotslife.com/telegram-bot-with-esp8266-6d3d3278e483.

[15] Random Nerd Tutorials, “Telegram: Control ESP32/ESP8266 Outputs (Arduino IDE),” [Online]. Available: https://randomnerdtutorials.com/telegram-control-esp32-esp8266-nodemcu-outputs/ .

[16] Bodmer, “GitHub - Bodmer/TFT_eSPI,” [Online]. Available: https://github.com/Bodmer/TFT_eSPI.

[17] Utilities for Online Operating System , “Convert Image to XBM,” [Online]. Available: https://www.online-utility.org/image/convert/to/XBM.

[18] Bodmer, “Arduino Analogue 'ring' Meter on Colour TFT Display,” [Online]. Available: https://www.instructables.com/Arduino-analogue-ring-meter-on-colour-TFT-display/.

[19] Blynk, “Get Multiple Datastream Values,” Blynk, [Online]. Available: https://docs.blynk.io/en/blynk.cloud/get-multiple-datastream-values .

[20] Blynk, “Batch Update of the Datastreams,” Blynk, [Online]. Available: https://docs.blynk.io/en/blynk.cloud/batch-update-values.

[21] Random Nerd Tutorials, “ESP32 Pinout Reference,” [Online]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/.

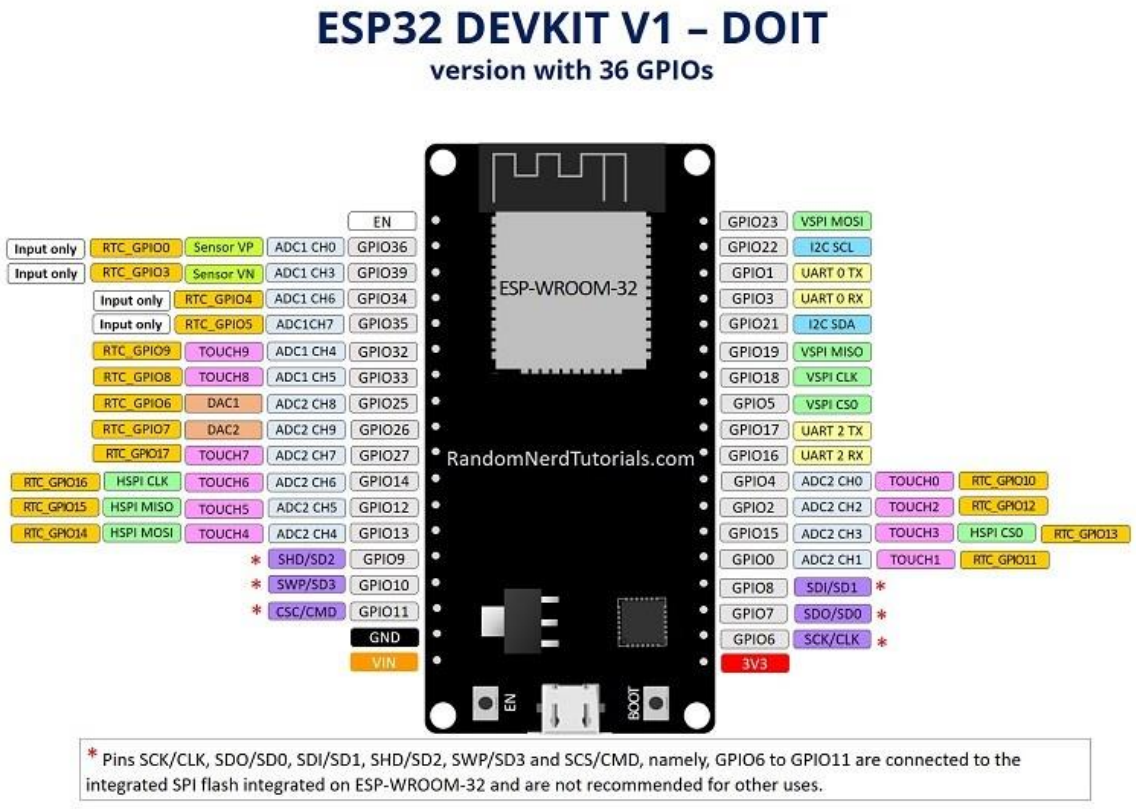
[22] Random Nerd Tutorials, “ESP32-CAM AI-Thinker Pinout Guide,” [Online]. Available: https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/.

[23] Blynk, “Blynk Documentation,” [Online]. Available: https://docs.blynk.cc/#blynk-firmware-blynktimer-blynk_writevpin .

[24] Telegram, “Bots: An introduction for developers,” [Online]. Available: https://core.telegram.org/bots.

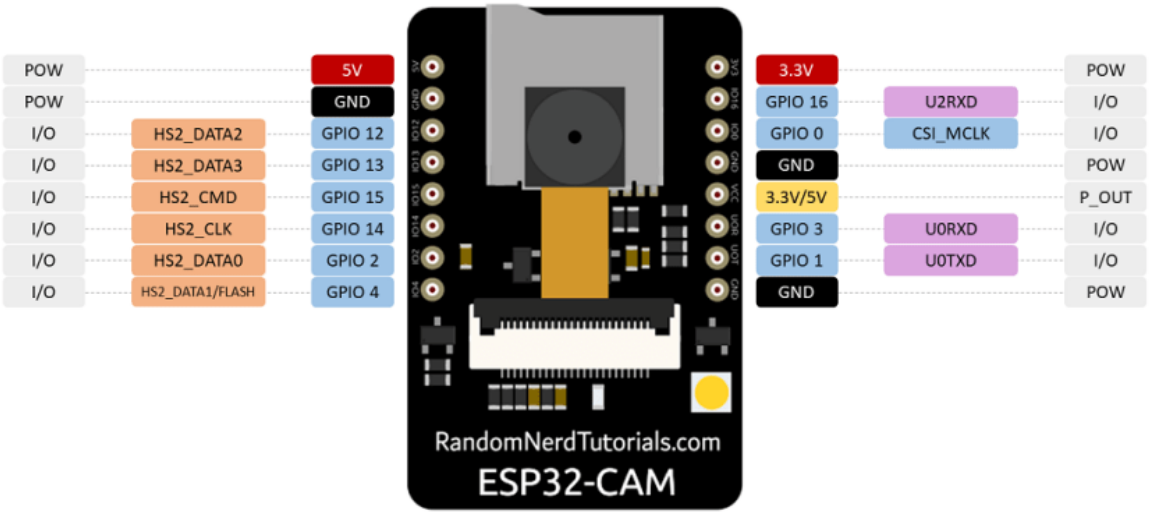
[25] witnessmenow, “Universal Telegram Bot Library,” [Online]. Available: https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot.

Appendix A – ESP32-WROOM-32 Pinout Diagram



[21]

Appendix B - ESP32-CAM AI-Thinker Pinout Diagram

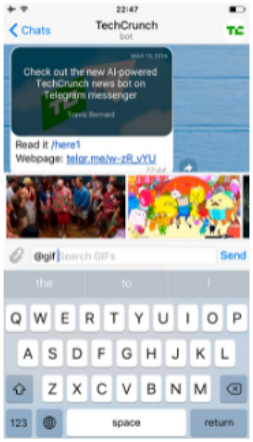


[22]

Appendix C – Features of Telegram Bot API

1. What can I do with bots?

To name just a few things, you could use bots to:



- **Get customized notifications and news**. A bot can act as a smart newspaper, sending you relevant content as soon as it's published.
- **Integrate with other services**. A bot can enrich Telegram chats with content from external services. **Gmail Bot**, **GIF bot**, **IMDB bot**, **Wiki bot**, **Music bot**, **Youtube bot**, **GitHubBot**
- **Accept payments from Telegram users**. A bot can offer paid services or work as a virtual storefront. Read more » **Demo Shop Bot**, **Demo Store**
- **Create custom tools**. A bot may provide you with alerts, weather forecasts, translations, formatting or other services. **Markdown bot**, **Sticker bot**, **Vote bot**, **Like bot**
- **Build single- and multiplayer games**. A bot can offer rich HTML5 experiences, from simple arcades and puzzles to 3D-shooters and real-time strategy games. **GameBot**, **Gamee**
- **Build social services**. A bot could connect people looking for conversation partners based on common interests or proximity.
- **Do virtually anything else**. Except for dishes — bots are terrible at doing the dishes.

[24]

Appendix D – Blynk functions to Read and Write data to Server

Blynk.virtualWrite(vPin, value)

NOTE: Use BlynkTimer when you use this command to send data. Otherwise your hardware will be disconnected from the server

Send data in various formats to Virtual Pins.

```
// Send string
Blynk.virtualWrite(pin, "abc");

// Send integer
Blynk.virtualWrite(pin, 123);

// Send float
Blynk.virtualWrite(pin, 12.34);

// Send multiple values as an array
Blynk.virtualWrite(pin, "hello", 123, 12.34);

// Send RAW data
Blynk.virtualWriteBinary(pin, buffer, length);
```

Calling `virtualWrite` attempts to send the value to the network immediately.

Note: For virtual pins with numbers > 127, the `V128` syntax is not available.
Please use plain virtual pin number, for example:

```
Blynk.virtualWrite(128, "abc");
```

BLYNK_WRITE(vPIN)

`BLYNK_WRITE` is a function called every time device gets an update of Virtual Pin value from the server (or app):

To read the received data use:

```
BLYNK_WRITE(V0)
{
  int value = param.asInt(); // Get value as integer

  // The param can contain multiple values, in such case:
  int x = param[0].asInt();
  int y = param[1].asInt();
}
```

`BLYNK_WRITE` can't be used inside of any loop or function. It's a standalone function.

Note: For virtual pins with numbers > 127, please use `BLYNK_WRITE_DEFAULT()` API

[23]

Appendix E – Telegram Bot Send Message Function

Sending messages	Your bot can send messages to any Telegram or group. This can be useful to get the arduino to notify you of an event e.g. Button pressed etc (Note: bots can only message you if you messaged them first)	`bool sendMessage(String chat_id, String text, String parse_mode = "")` Sends the message to the chat_id. Returns if the message sent or not.

[25]