**CHENG-CHIH LEE**

**29329351**

**Programming Assignment 5 –**

**Code RED**

**CHENG-CHIH LEE**

**Student ID:29329351**

**2025/12/01**

**UCI MECPS**

**#Summary**

**This assignment is built upon the previous lab and creates a swarm to connect at most three ESP32's and a Raspberry Pi (as a data logger). The RPi will have a button and four LEDs (Red, Green, Yellow, White). The ESP will have an analog light sensor (photoresistor in the package).**

* **ESP 8266 [LightSwarm.ino](https://canvas.eee.uci.edu/courses/74175/files/32519698?wrap=1" \t "_blank" \o "LightSwarm.ino)**[**Download LightSwarm.ino**](https://canvas.eee.uci.edu/courses/74175/files/32519698/download?download_frd=1)**(obtained from Prof. Shovic's github)**
* **Raspberry Pi**[**LightSwarm.py**](https://canvas.eee.uci.edu/courses/74175/files/32519724?wrap=1)[**Download LightSwarm.py**](https://canvas.eee.uci.edu/courses/74175/files/32519724/download?download_frd=1)**(obtained from Prof. Shovic's github)**
  + **Python 3 version:**[**github.com/switchdoclabs/SDL\_Pi\_LightSwarmLinks to an external site.**](http://github.com/switchdoclabs/SDL_Pi_LightSwarm)

**Each of the three ESP32's in the swarm is identical. There are no software differences and no hardware differences. They can communicate with each other by broadcasting messages to exchange sensor readings. One of them, the ESP32 with the highest reading, will become the "Master" and will forward the readings to the RPi for data logging.**

**\*I teared down the legacy code and fit it into FreeRTOS architecture.**

**For assignment 4:**

**Extra plotting and data log functions are added to this assignment on the RPi side.**

**External LED with PWM brightness control on the ESPs are added.**

**For assignment 5 Code RED:**

**+ESP32: An Extra LED bar is added to one ESP for indicating the brightness.**

**+RPi: An Extra LED Matrix is added to indicate dynamic brightness received.**

**A local web is developed to substitute the plot in assignment4.**

**(key words: ESP32, Raspberry Pi 5, UDP, Photocell Sensor, GPIO, FreeRTOS, threading)**

**>>>>**[**Demo Video Link**](https://drive.google.com/file/d/1wFQlQPM65_qn-iQOzpiUm-tvtTTAlcZy/view?usp=sharing)**<<<<**

**https://drive.google.com/file/d/1wFQlQPM65\_qn-iQOzpiUm-tvtTTAlcZy/view?usp=sharing**

**#OUTLINE**

[**-Version History**](#version)

[**-Components used / Pin Definition / Schematics**](#component_used)

[**-Overall Flow-chart**](#flowchart)

[**-Protocol**](#protocol)

[**-State Machine**](#state_machine)

[**-Part 1-1** **Raspberry Pi WiFi setup and packet delivery**](#part1)

[**-Part 1-2 Rpi correctly reacts to received packets**](#part2)

**-Part 1-3 Rpi Web plot and plot reset**

**-Part 1-4 Rpi LED Matrix**

[**-Part 2-1 ESP WiFi setup and packet delivery**](#part3)

[**-Part 2-2 ESP correctly reacts to received packets**](#part4)

[**-Part 2-3 ESP LED Bar**](#part2_3)

**-**[**Part 3-1 Generative Ai usage summary**](#part3_1)

**-**[**Part 3-2 ZotGPT prompts and response**](#part3_2)

**-**[**Part 3-3 Generated code test and verification**](#part3_3)

# # Version History

|  |  |  |  |
| --- | --- | --- | --- |
| ***Version*** | ***Date*** | ***Comment*** | ***Known Issue/Fix*** |
| v01\_Beta | 11/04/2025 | Beta version |  |
| V01 | 11/07/2025 | Formal Release |  |
| V02 | 11/18/2025 | Formal Release | With Plotting Function and ESP32 PWM LED |
| V03 | 12/01/2025 | Formal Release | LED Bar, LED Matrix, Local Web Plot |

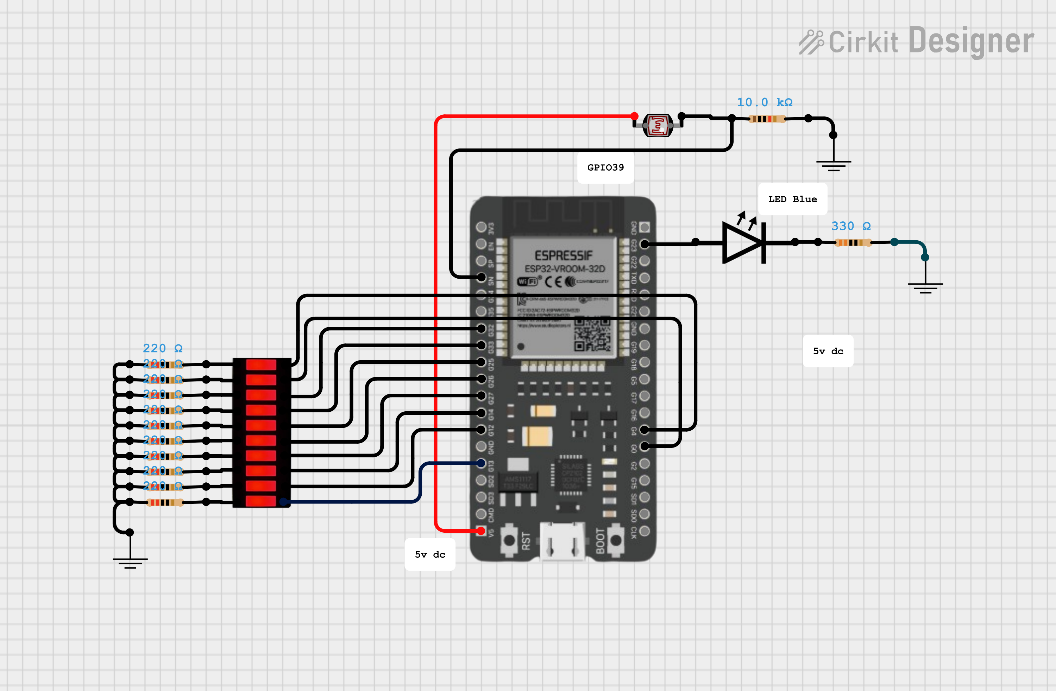
# #Components used

1. **Photocell with a 10k pull-down resistor.**
2. **External LEDs. Need resistors (330 Ohm).**
3. **A button. Need a pull-down resistor (10k Ohm), digital input and with interrupt mode set to that pin.**
4. **LED Bar. Need 10 resistors (220 Ohm)**

**#Pin Definition for *ESP32***

|  |  |  |
| --- | --- | --- |
| **Pin Name** | **Definition** | **Usage and comments** |
| **LED\_BUILTIN** | **GPIO 2** | **Built in onboard LED (fixed)** |
| **LED\_BLUE** | **GPIO 23** | **External LED PWM pin** |
| **PHOTO\_CELL** | **GPIO 34** | **ADC Sampling pin for Photocell sensor** |
| **TEN\_SEG\_LED\_1** | **GPIO 13** | **LED Bar** |
| **TEN\_SEG\_LED\_2** | **GPIO 12** | **LED Bar** |
| **TEN\_SEG\_LED\_3** | **GPIO 14** | **LED Bar** |
| **TEN\_SEG\_LED\_4** | **GPIO 27** | **LED Bar** |
| **TEN\_SEG\_LED\_5** | **GPIO 26** | **LED Bar** |
| **TEN\_SEG\_LED\_6** | **GPIO 25** | **LED Bar** |
| **TEN\_SEG\_LED\_7** | **GPIO 33** | **LED Bar** |
| **TEN\_SEG\_LED\_8** | **GPIO 32** | **LED Bar** |
| **TEN\_SEG\_LED\_9** | **GPIO 0** | **LED Bar** |
| **TEN\_SEG\_LED\_10** | **GPIO 4** | **LED Bar** |

**Table.1 Pin Definition**

****

**Fig.1 Schematics**

**#Pin Definition for *Raspberry Pi***

|  |  |  |
| --- | --- | --- |
| **Pin Name** | **Definition** | **Usage and comments** |
| **LED\_R** | **GPIO 26** | **Pin for Red External LED** |
| **LED\_Y** | **GPIO 13** | **Pin for Red External YELLOW** |
| **LED\_G** | **GPIO 6** | **Pin for Red External GREEN** |
| **LED\_W** | **GPIO 5** | **Pin for Red External WHITE** |
| **BTN** | **GPIO 16** | **Pin for External Push Button** |
| **LED\_Matrix SPIMOSI** | **GPIO 10** | **Pin for SPIMOSI** |
| **LED\_Matrix SPICLK** | **GPIO 11** | **Pin for SPICLK** |
| **LED\_Matrix SPICE0** | **GPIO 8** | **Pin for SPICE0** |

**Table.2 Pin Definition**

**一張含有 文字, 圖表, 電路, 方案 的圖片

AI 產生的內容可能不正確。**

**Fig.2 Schematics**

# #Overall Flow-chart

**一張含有 文字, 圖表, 螢幕擷取畫面, 方案 的圖片

AI 產生的內容可能不正確。**

# 一張含有 文字, 圖表, 螢幕擷取畫面, 圓形 的圖片 AI 產生的內容可能不正確。#State Machine

# #Protocols

During my development, UDP broadcasting is configured. That means all devices in the network should be able to receive all packets sent. No IP is hardcoded in any device. Port is set as 1996.

**Laptop：IP ----, Port 1996**

**RPi ：IP ----, Port 1996**

**ESP32：IP ----, Port 1996**

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Field** | **Value** | **Description** |
| **0** | 0xF0 | Constant | **Start-of-frame marker**. Helps receivers validate the packet type. |
| **1** | LIGHT\_UPDATE\_PACKET (=0) | 0 | **Packet type ID** — distinguishes from reset, blink, etc. |
| **2** | localIP[3] | e.g., 123 | **Node ID** — the last octet of its IP (unique in local subnet). Used as swarm identifier. |
| **3** | masterState | 0 or 1 | **Role flag:** 1 = master, 0 = slave. |
| **4** | VERSIONNUMBER | e.g., 28 | **Firmware version** for compatibility tracking. |
| **5–6** | clearColor (high, low) | sensor value | **Clear (luminance)** 16-bit reading from the photocell (moving average). |
| **7–8** | redColor (high, low) | Not used | Placeholder for **red channel intensity**. (Unused here since no color sensor.) |
| **9–10** | greenColor (high, low) | Not used | Placeholder for green. |
| **11–12** | blueColor (high, low) | Not used | Placeholder for blue. |
| **13** | 0x0F | 0x0F | **End-of-frame marker** (symmetrical with start marker). |

# Part 1-1 Raspberry Pi WiFi setup and packet delivery

**Files: talk\_v04.py, UDP\_v03.py, state\_machine\_v05py, LightSwarm.py, plot.py, led\_matrix.py, web.py**

**In this project, the LightSwarm functionality is modified/integrated into the previous “we\_need\_to\_talk” project.** talk\_v04 works as the main file, initializing some threads. UDP\_v03 works as the UDP rx/tx library module. The state machine module is modified to keep only two states, including operation and reset Swarm. And the RPi is connected to my personal hotspot named IPhoneKL. WiFi related functionalities are developed in the ***“UDP\_ v03.py”*** module.

**#WiFi Setup and UDP**

For the UDP setup, it’s set to broadcast mode, thus sending to 255.255.255.255. port is 1996.

一張含有 文字, 螢幕擷取畫面, 軟體, 數字 的圖片

AI 產生的內容可能不正確。

In the main file (***talk\_ v04.py***), I put UDP\_Receive() and processPacket() and mat.show\_swarm() to ***thread*** to make sure it keeps listening and driving LED matrix.

The local web is also called to run here to initialize the web page.

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

This UDP module also works as a layer for state\_machine and LightSwarm to exchange the LSCommand. The set functionset ***LSCommand(cmd)*** is for writing to this layer; ***get*** functions are for accessing values.

一張含有 文字, 螢幕擷取畫面, 軟體, 電腦圖示 的圖片

AI 產生的內容可能不正確。

For UDP **sending**, I developed ***UDP\_Send()*** method to send messages to both my laptop and ESP32 for the beta version. This method will later be called by the ***state\_maching*** module.

For UDP **receiving**, I developed ***UDP\_Receive()*** method to receive messages. Now RPi is listening to all UDPs with this method put to thread. Upon receiving UPDs, it also updated the ***new\_msg\_cnt*** for checking message incoming status.

一張含有 文字, 螢幕擷取畫面, 軟體, 電腦圖示 的圖片

AI 產生的內容可能不正確。

# Part 1-2 Rpi reacts to received packets

**#State Machine**

In the State machine, 2 state transitions are developed. The ***button\_callback()*** is attached to external push button input interrupt for triggering state change.

一張含有 文字, 螢幕擷取畫面, 數字, 軟體 的圖片

AI 產生的內容可能不正確。

**#Operation Mode *m\_operation()***

This state is triggered when status flag ***sys\_state == 0*** and the **button is pressed**.

The sys\_state is then set to 1 indicating it’s in operation mode. The phototsensing and RGY led indicating tasks will be put to threading. And the system will then remove plot stop event flag. This will un-block the plot thread in the main thread.

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

AI 產生的內容可能不正確。

A function collect\_data() is developed in plot module to execute photosensor reading related for the web plotting in this assignment. It will be put to thread if plot mode is set.

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

**#Photocell value process *photo\_sns()***

When photo\_sns() is put to thread, with ***sys\_state==1 and the thread is not stopped***. This method checks if the packet is from a master device or not, clear led indicator thread stop flag, and it also shuts down leds accordingly.

一張含有 文字, 電子產品, 螢幕擷取畫面, 軟體 的圖片

AI 產生的內容可能不正確。

**#Reset mode *m\_reset ()***

***Works as reset Swarm task***

During normal operation, when the external button is pressed, the state machine will generate the photo\_sns() and blink\_rgy\_stop() thread stopping event and sent **UDP.setLSCommand("RESETSWARM")**.

This mode will then go through stopping plot thread by calling reset\_plot(), exporting log file and resetting the plot.

# 一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片 AI 產生的內容可能不正確。Part 1-3 Rpi plot and plot reset

This assignment includes the plotting functionality. A plot module is developed and integrated into the application. There are 3 major functions in the plot module. **collect\_data, reset\_plot and data log exporting. Another function is get\_plot\_data() added for web module to fetch required data from plot module.**

**#plot**

In this assignment, no plot is developed in this module. This modules now works as data collector for the local web application.

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

**#reset\_plot()**

When the reset\_plot is called, the data for bar chart data will be cleared, the time axis will be set to zero and setting a reset flag for the plot() thread to clear the data chart.

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 螢幕擷取畫面, 數字, 軟體 的圖片

AI 產生的內容可能不正確。

**#ex\_log ()**

When the ex\_log is executed this function export the data in the previous 30 secs to a .txt file.

一張含有 文字, 螢幕擷取畫面, 數字, 軟體 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 螢幕擷取畫面, 電腦, 軟體 的圖片

AI 產生的內容可能不正確。

**#web**

***I used ZOTGPT to develop this part. Please see Part 3-1.***

In this module, flask library is used to realize the local web application. Object “web” is instantiated at the top of this module.

**一張含有 文字, 字型, 行, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。**

ZotGPT developed a HTML that worked pretty well for the web so I just kept it.

**一張含有 文字, 螢幕擷取畫面, 軟體, 數字 的圖片

AI 產生的內容可能不正確。**

5 endpoints were developed. They are home, get\_data, reset\_plot, export\_log and status.

**一張含有 文字, 螢幕擷取畫面, 文件, 字型 的圖片

AI 產生的內容可能不正確。**

一張含有 螢幕擷取畫面, 文字, 軟體, 多媒體軟體 的圖片

AI 產生的內容可能不正確。This the what the web looks like:

**-Part 1-4 Rpi LED Matrix**

In this assignment, a LED Matrix is added to indicate the dynamic sensor reading from ESP32 master device. I used the LED 8\*8 matrix from the kit. It comes with MAX7219CNG controller. SPI protocol is used to communicate with it.

Basic usage is to send it (register, data). For just controlling the matrix, the register 1-9 stands for matrix rows 1-9; data is as easy as 8 bits 1/0 and converted to hex. For example, to control on row 2, led 2, we need to send (2, 0100 0000(hex)).

I developed a class for controlling the matrix. Initialization requires setting spi0 module, go through the several settings and I set all leds to LOW.

The MAT\_VAL array at the top is ready for controlling LED on status from none to all 8 leds.

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

AI 產生的內容可能不正確。

To make the LED matrix show dynamic and continuous data, I decided to put the received value to a data queue.

So the show\_swarm() function will push new data into the data\_queue and call show() function to send all data in the updated data\_queue through spi.

show\_swarm() is also put to thread in the system init status.

一張含有 文字, 螢幕擷取畫面, 數字, 軟體 的圖片

AI 產生的內容可能不正確。

# Part 2-1 ESP WiFi setup and packet delivery

ESP is connected to my hotspot. It is also developed to UDP broadcasting.

一張含有 文字, 多媒體軟體, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

**Swarm data array is initialized here.**

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

AI 產生的內容可能不正確。

**#Tasks**

After WiFi connection, six tasks are created. **UDP receiving Task** for UDP receiving, **Task\_PhotoSns** for Photocell sensing, **Task\_UDP\_Send** for sending UDP packet. **Task\_Swarm** for Swarm logic implementation. **Task\_LED\_Blink** for controlling Built-In led blinking behavior. **Task\_Monitor** for checking stack watermark.

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

**#UDP Receiving *UDP\_Receive(void)***

In the UDP\_Receive() function, UDP incoming messages are processed by calling the Packet\_Helper function. There’s a mutex set to protect the incoming packet.

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。

**#UDP Sending *UDP\_Send(const char\* msg)***

In the UDP\_Send() function, broadcastARandomUpdatePacket() is called and sendLightUpdatePacket() is called accordingly.

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。

# Part 2-2 ESP correctly reacts to received packets

**#Task\_Swarm**

The Task\_Swarm is for checking the main logic for Light Swarm by calling checkAndSetIfMaster().

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

AI 產生的內容可能不正確。

**#Photocell sensing function**

The Photo\_Sns(void) function is developed to execute sensing with analog reading. A moving average function MovAvg(uint16\_t input, uint16\_t \*buffer) is used to derive 5 sec sliding window average.

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

AI 產生的內容可能不正確。

**#Task\_LED\_Blink function**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。This function is now modified to set PWM duty to the external Blue LED pin to indicate the photo sensor reading. The reading is 0~4095, so I set the duty resolution to 2^12.



# Part 2-3 ESP LED Bar

**#Task\_LED\_Bar**

This function was developed to control the 10 segment LED bar. The LED bar is as easy as ten leds. Controlling HIGH/LOW of GPIO is enough. To make it easier, I store the leds to an array, so the execution function only needs to run a loop to scan through leds needed to be turned on and off.

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

AI 產生的內容可能不正確。一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

AI 產生的內容可能不正確。This function is also created as a FreeRTOS task with cycle time 200ms.

# -Part 3-1 Generative Ai usage summary

For this assignment, the Raspberry Pi python files are:

**talk\_v04.py** >> Modified by me for this assignment.

**UDP\_v03.py** >> Not changed.

**state\_machine\_v05py** >> Modified by me for this assignment.

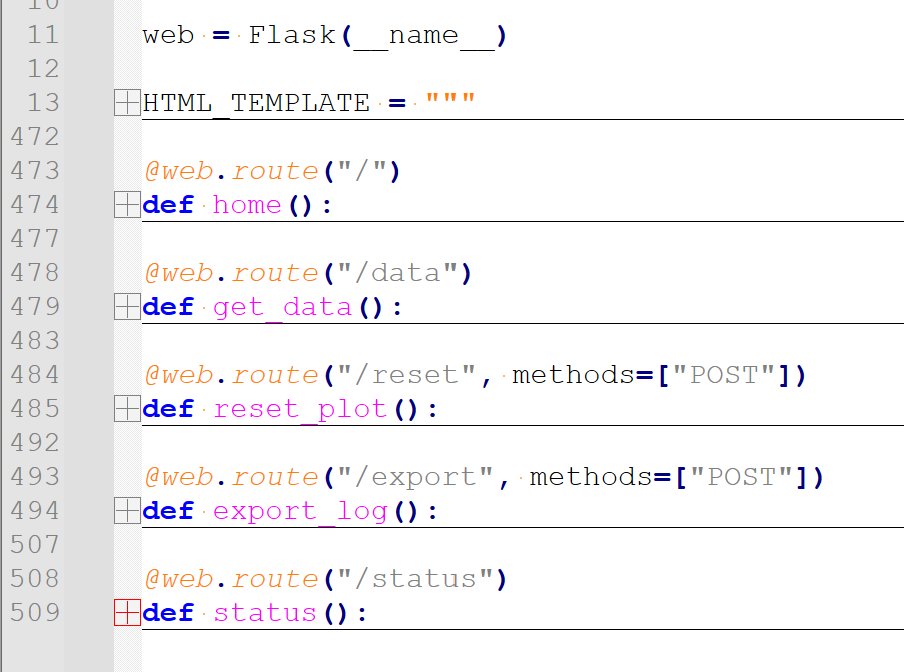
**LightSwarm.py** >> Not changed.

**plot.py** >> Modified by me for this assignment.

**led\_matrix.py** >> Newly developed by me for this assignment.

**web.py** >> ZotGPT developed draft, I modified and integrated.

The web.py consists of 2 major parts. First part is **HTML and CSS** for the web structure and style. Second part is the **web route**. This is responsible for the endpoint behaviors.



# -Part 3-2 ZotGPT prompts and response

There were two parts of my prompt. 

First, I paste the plot.py from **The Plot Thickens** assignment. And I ask Zotgpt:

***My prompt:***

My plot.py from **The Plot Thickens [text omitted]**

I want to move this plot to a local web on my RPi with a single py script with flask and html. I will keep this plot.py to collect data. So I'll change the plot() to collect\_data(). Please provide a web.py script. And necessary additive functions to my plot.py script.

***Responses:***

Scripts from ZotGPT **[text omitted]**

## 3. \*\*How to Run\*\*

### Install Dependencies:

```bash

pip3 install flask

```

### Start the Web Dashboard:

```bash

python3 web.py

```

### Access the Dashboard:

- \*\*On Raspberry Pi\*\*: `http://localhost:5000`

- \*\*From another device\*\*: `http://<raspberry-pi-ip>:5000`

Example: `http://192.168.1.100:5000`

---

## 4. \*\*Key Features\*\*

✅ \*\*Thread-safe data access\*\* with locks

✅ \*\*Automatic data collection\*\* in background thread

✅ \*\*Real-time web charts\*\* using Chart.js

✅ \*\*Reset functionality\*\* with counter tracking

✅ \*\*Export logs\*\* to timestamped files

✅ \*\*Rolling 30-second window\*\* for efficient memory usage

✅ \*\*Responsive design\*\* works on mobile devices

✅ \*\*Accessible from any device\*\* on your network

---

## 5. \*\*File Structure\*\*

```

your\_project/

├── plot.py # Data collection (modified)

├── web.py # Flask web server (NEW)

├── LightSwarm.py # Your existing module

└── log/ # Exported logs (auto-created)

```

The web dashboard will automatically start the data collection thread when you run `web.py`, so you don't need to run `plot.py` separately anymore! 🎉

# -Part 3-3 Generated code test and verification

Now with the code generated from ZotGPT, I started the integration. The generated code is fine, but it does not perfectly work with my system architecture.

Now the **plot.py** works as a data collection module for the web page. The main difference is this:

**def** get\_plot\_data**():**

"""Returns current plot data (thread-safe)"""

**return** **{**

'time0'**:** xs0**.**tolist**(),**

'brightness0'**:** ys0**.**tolist**(),**

'time1'**:** xs1**.**tolist**(),**

'brightness1'**:** ys1**.**tolist**(),**

'time2'**:** xs2**.**tolist**(),**

'brightness2'**:** ys2**.**tolist**(),**

'master\_count'**:** master\_count**.**copy**(),**

'current\_time'**:** current\_time**,**

'reset\_counter'**:** reset\_counter

**}**

This is an interface for the web.py to fetch data from plot.py. Other than this, this plot.py does not need further change.

Also, in my main file **talk\_v04.py**, I integrated the web.run to start the web page.

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

STATE**.**gpio\_setup**()**

#init a led matrix

mat **=** MAT**.**LED\_MAT**(**"RPi LED Mat"**)**

mat**.**spi\_init**(**0**,** 0**,** 1000000**,** 0**)**

time**.**sleep**(**0.2**)**

mat**.**mat\_init**()**

time**.**sleep**(**0.2**)**

receiver\_thread **=** threading**.**Thread**(**target**=**UDP**.**UDP\_Receive**,** daemon**=True)**

receiver\_thread**.**start**()**# Put UDP listening to thread to ensuring listening

processPacket\_thread **=** threading**.**Thread**(**target**=**LS**.**processPacket**,** daemon**=True)**

processPacket\_thread**.**start**()**

ledMatrix\_thread **=** threading**.**Thread**(**target**=**mat**.**show\_swarm**,** daemon**=True)**

ledMatrix\_thread**.**start**()**

**print(**"####################################################"**)**

**print(**"System Up! Listening to LightSwarm Packets!!"**)**

**print(**"####################################################"**)**

**print(**"=" **\*** 60**)**

**print(**"ߚ Starting LightSwarm Web Dashboard (Real-Time Plotting)"**)**

**print(**"=" **\*** 60**)**

**print(**"\nߓʠDashboard URLs:"**)**

**print(**" Local: http://localhost:5000"**)**

**print(**" Network: http://<raspberry-pi-ip>:5000"**)**

**print(**"\nߒ`Press Ctrl+C to stop"**)**

**print(**"=" **\*** 60**)**

**try:**

WEB**.**web**.**run**(**host**=**"0.0.0.0"**,** port**=**5000**,** debug**=False,** threaded**=True)**

**except** **KeyboardInterrupt:**

PLOT**.**plot\_stop**.set()**

time**.**sleep**(**0.1**)**

mat**.**close**()**

**print(**"Exiting..."**)**

After Integration, I kept these functions for utilizing the web.

**from** flask **import** Flask**,** render\_template\_string**,** jsonify

**import** plot **as** PLOT

**import** state\_machine\_v05 **as** STATE

web **=** Flask**(**\_\_name\_\_**)**

HTML\_TEMPLATE **=** """

"""

@web**.**route**(**"/"**)**

**def** home**():**

@web**.**route**(**"/data"**)**

**def** get\_data**():**

@web**.**route**(**"/reset"**,** methods**=[**"POST"**])**

**def** reset\_plot**():**

@web**.**route**(**"/export"**,** methods**=[**"POST"**])**

**def** export\_log**():**

@web**.**route**(**"/status"**)**

**def** status**():**

Tests for the generated code are shown as below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Test Type** | **Input** | **Expected Result** | **Test Result** |
| Web\_Init | Functional Test | Run script and access web page | Web page without data | Pass |
| Web plotting | Functional Test | Press RPi button to start plotting | Data from different ESPs got plotted as expected. | Pass |
| Plot Reset | Functional Test | Press RPi button to reset plotting | Web Plot reset with time axel reset. | Pass |
| Data 30 seconds | Boundary Test | Let plot data run over 30 seconds | Plot clears for the next 30 seconds | Pass |
| Web page refresh | Boundary Test | Refresh the web page | Web plot continues without crashing | Pass |