

**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](#) [Download LightSwarm.ino](#)(obtained from Prof. Shovic's github)
- **Raspberry Pi** [LightSwarm.py](#) [Download LightSwarm.py](#)(obtained from Prof. Shovic's github)
- **Python 3 version:** [github.com/switchdoclabs/SDL\\_Pi\\_LightSwarm](#)[Links to an external site.](#)

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-tvtTTAICZy/view?usp=sharing](https://drive.google.com/file/d/1wFQlQPM65_qn-iQOzpiUm-tvtTTAICZy/view?usp=sharing)

## #OUTLINE

**-Version History**

**-Components used / Pin Definition / Schematics** [\*<< Code RED\*](#)

**-Overall Flow-chart**

**-RPi Data Log Flow-chart** [\*<< Code RED\*](#)

**-Protocol** [\*<< Code RED\*](#)

**-State Machine**

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

**-Part 1-2 Rpi correctly reacts to received packets**

**-Part 1-3 Rpi Web plot and plot reset** [\*<< Code RED\*](#)

**-Part 1-4 Rpi LED Matrix** [\*<< Code RED\*](#)

**-Part 2-1 ESP WiFi setup and packet delivery**

**-Part 2-2 ESP correctly reacts to received packets**

**-Part 2-3 ESP LED Bar** [\*<< Code RED\*](#)

**-Part 3-1 Generative Ai usage summary** [\*<< Code RED\*](#)

**-Part 3-2 ZotGPT prompts and response** [\*<< Code RED\*](#)

**-Part 3-3 Generated code test and verification** [\*<< Code RED\*](#)

## # Version History

<b><i>Version</i></b>	<b><i>Date</i></b>	<b><i>Comment</i></b>	<b><i>Known Issue/Fix</i></b>
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

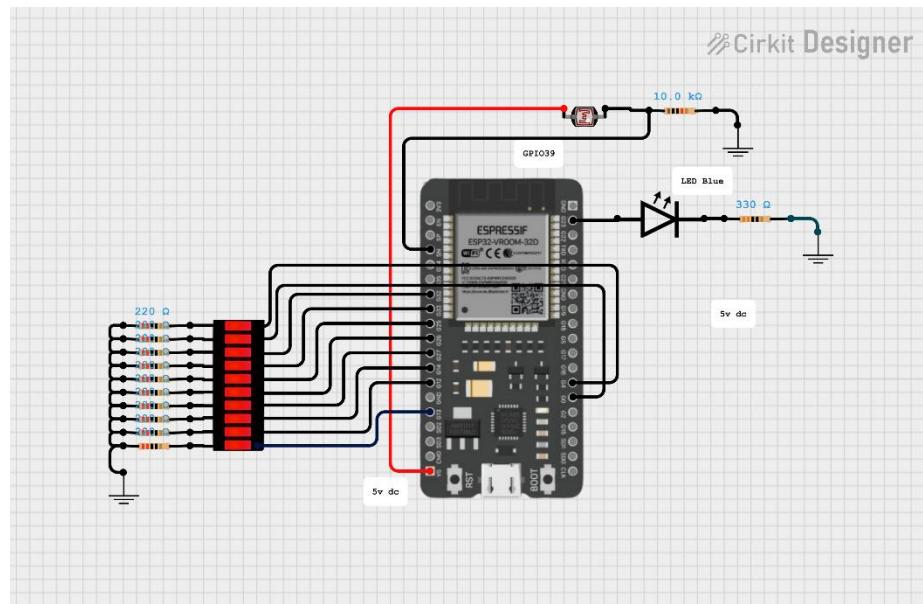


Fig.1 Schematics

## #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_SPI_MOSI	GPIO 10	Pin for SPI_MOSI
LED_Matrix_SPI_CLK	GPIO 11	Pin for SPI_CLK
LED_Matrix_SPI_CE0	GPIO 8	Pin for SPI_CE0

Table.2 Pin Definition

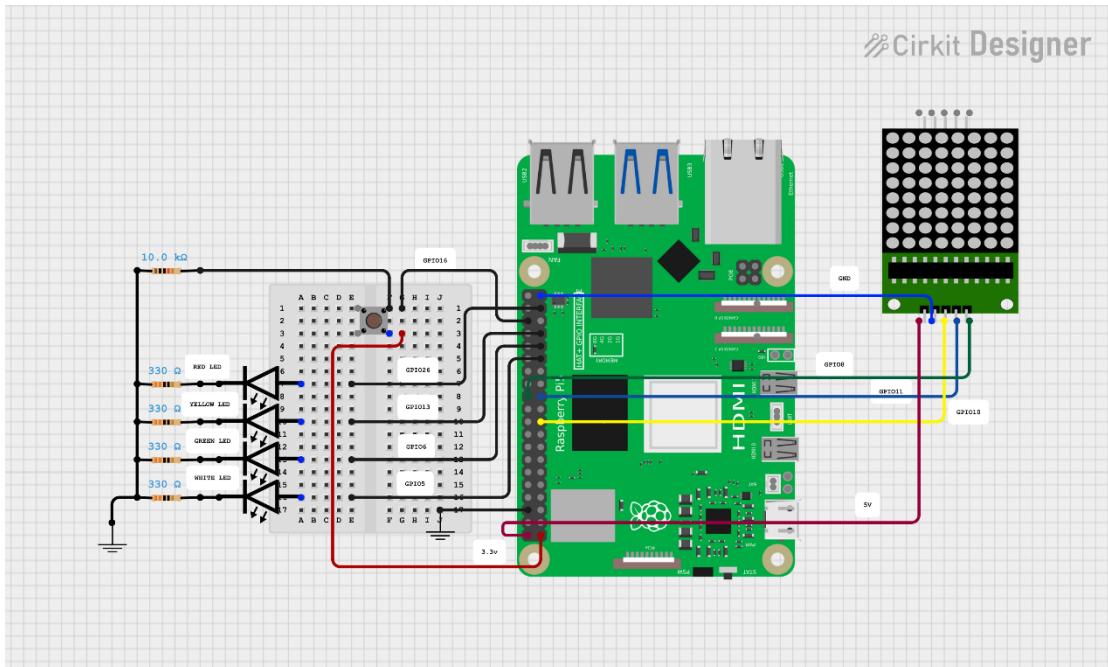
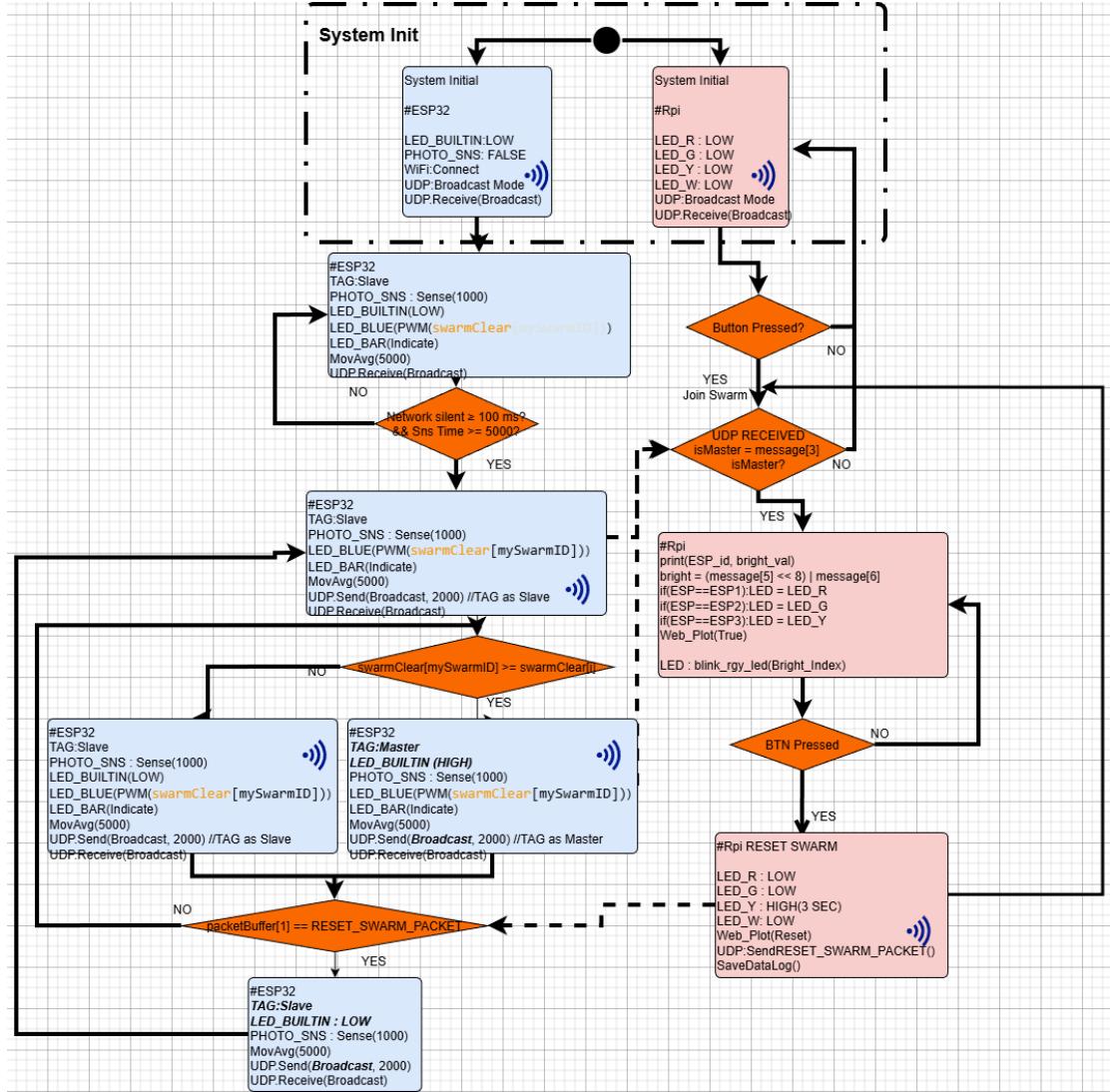
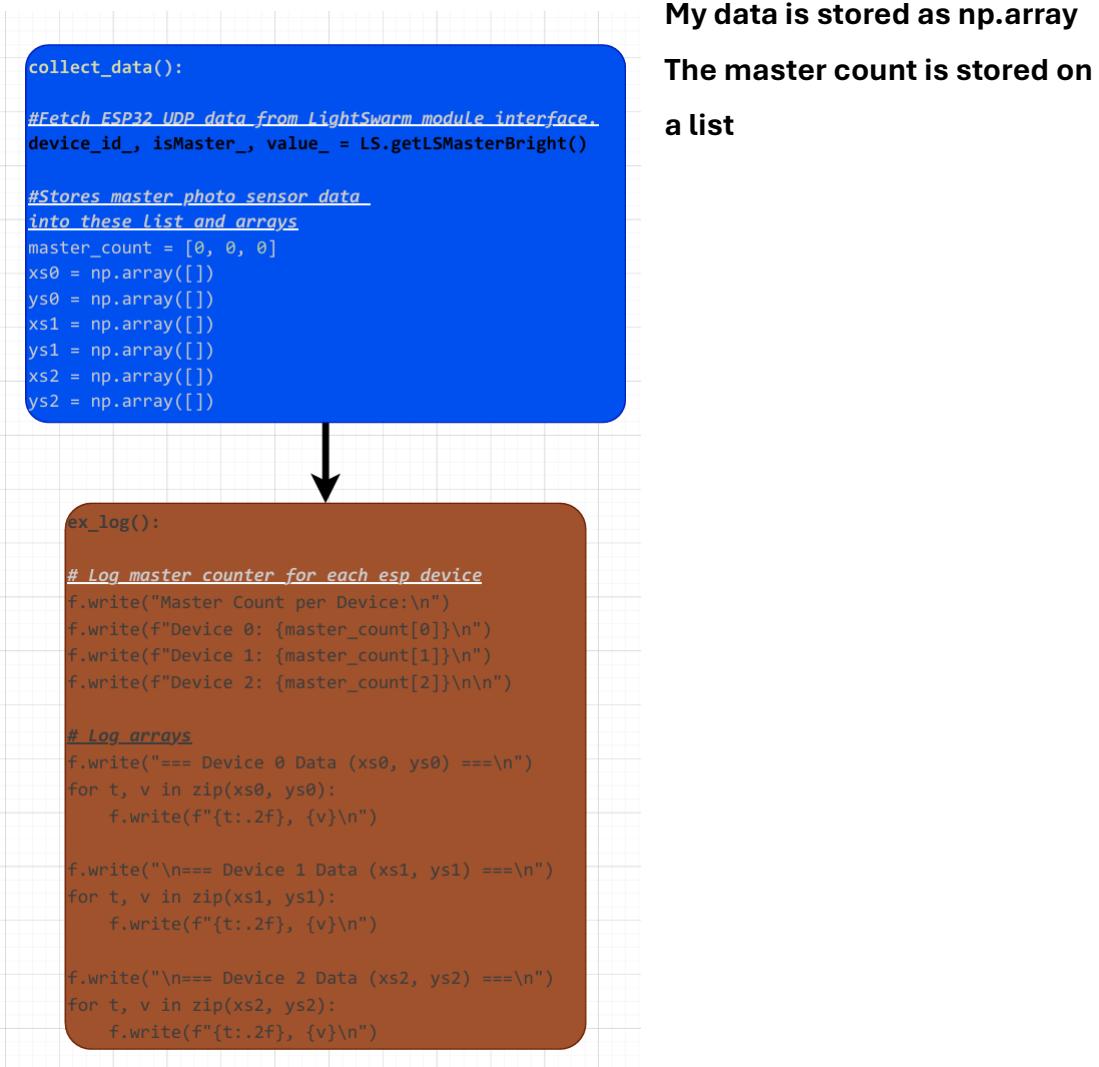


Fig.2 Schematics

## #Overall Flow-chart



## # RPi Data Log Flow-chart



LIGHTSWARM\_2

**Data log:** 0251201\_095952.

```

==== LightSwarm Data Log ====
Timestamp: 20251201_095952

```

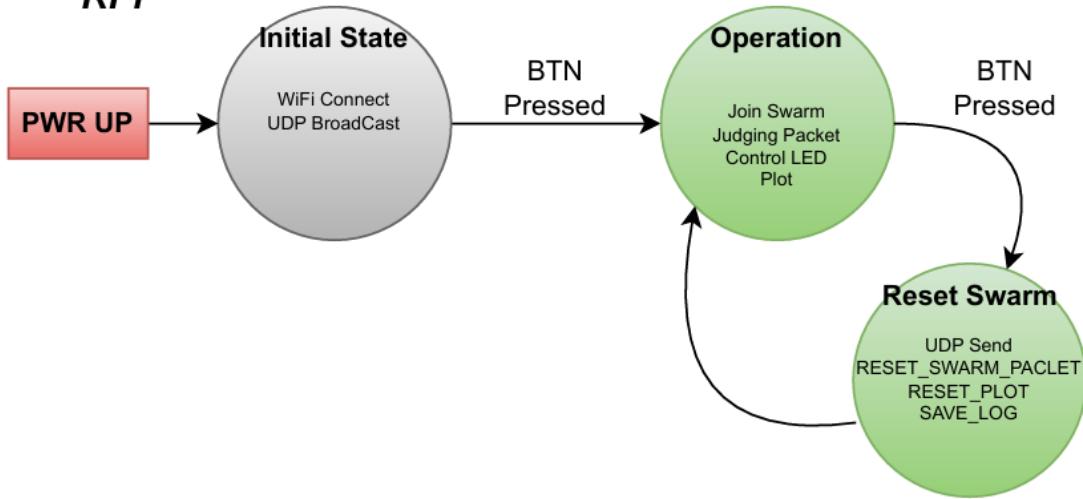
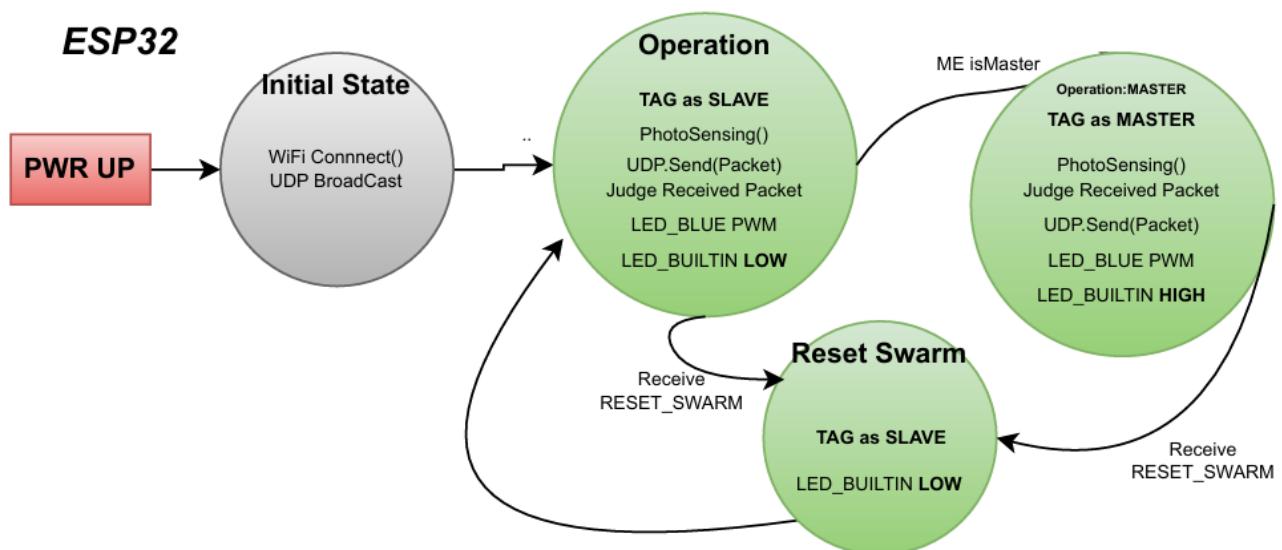
Master Count per Device:

Device 0: 8  
Device 1: 30  
Device 2: 0

==== Device 0 Data (xs0, ys0) ===

33.01, 2366.0  
34.01, 2399.0  
35.01, 2399.0  
36.01, 2399.0  
37.01, 2369.0  
38.01, 2369.0

## #State Machine

*RPi**ESP32*

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



```

1  #Kevin Lee 10/11/2025
2  #v03 Kevin Lee 11/03/2025 Modified for LightSwarm Project
3  import socket
4  print("#####")
5  print("!!!!!!Hi!!!!>0<!!!!!!")
6  print("#####")
7  #UDP_IP = input("Enter Listening IP or Enter to default 0.0.0.0:") or "0.0.0.0"
8  #UDP_PORT = 8386
9  #Laptop_UDP_IP = input("Enter Laptop IP or Enter to default 172.20.10.4:") or "1"
10 #Laptop_UDP_PORT = 823
11 #ESP32_UDP_IP = input("Enter ESP32 IP or Enter to default 172.20.10.2:") or "172"
12 #ESP32_UDP_PORT = 1996
13 UDP_IP = "255.255.255.255"
14 UDP_PORT = 1996
15
16 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
17 sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1) ... # ? allow broadcast
18 sock.bind((UDP_IP, UDP_PORT))

```

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.

```

13
14 if __name__ == '__main__':
15     ...
16     STATE.gpio_setup()
17     #init a led matrix
18     mat = MAT.LED_MAT("RPi.LED.Mat")
19     mat.spi_init(0, 0, 1000000, 0)
20     time.sleep(0.2)
21     mat.mat_init()
22     time.sleep(0.2)
23
24     receiver_thread = threading.Thread(target=UDP.UDP_Receive, daemon=True)
25     receiver_thread.start() # Put UDP listening to thread to ensuring listening
26     processPacket_thread = threading.Thread(target=LS.processPacket, daemon=True)
27     processPacket_thread.start()
28     ledMatrix_thread = threading.Thread(target=mat.show_swarm, daemon=True)
29     ledMatrix_thread.start()
30
31     print("#####")
32     print("System Up! Listening to LightSwarm Packets!")
33     print("#####")
34
35     print("=*60)
36     print("Starting LightSwarm Web Dashboard (Real-Time Plotting)")
37     print("=*60)
38
39     print("\nDashboard URLs:")
40     print("... Local: ... http://localhost:5000")
41     print("... Network: http://<raspberry-pi-ip>:5000")
42     print("\n💡 Press Ctrl+C to stop")
43     print("=*60)
44
45     try:
46         WEB.web.run(host="0.0.0.0", port=5000, debug=False, threaded=True)
47
48     except KeyboardInterrupt:
49         PLOT.plot_stop.set()
50         time.sleep(0.1)
51         mat.close()
52         print("Exiting...")
53

```

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.

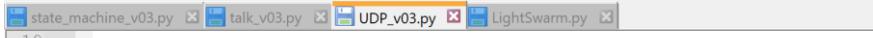
```

40     def get_message_r():
41         ... #for getting the received UDP message
42         return message_r
43
44     def get_new_msg_cnt():
45         ... #for getting the UDP message incoming status
46         return new_msg_cnt
47
48     def getLSCommand():
49         global LScmd
50         return LScmd
51
52     def setLSCommand(cmd):
53         global LScmd
54         LScmd = cmd

```

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 UPDs with this method put to thread. Upon receiving UPDs, it also updated the ***new\_msg\_cnt*** for checking message incoming status.



```

19
20     message_r = "", ""
21     new_msg_cnt = 0
22     LScmd = ""
23
24     def UDP_Send(message):
25         ... #BroadCast UDP
26         ... sock.sendto(message, (UDP_IP, UDP_PORT)).encode()
27
28         print(f"Broadcasting UDP Message:{message} to:{UDP_IP}:{UDP_PORT}")
29
30     def UDP_Receive():
31         ... global message_r
32         ... global new_msg_cnt
33
34         while True:
35             ... data, addr = sock.recvfrom(1024)
36             ... message_r = (data, addr) #store tuple (bytes, address)
37             ... new_msg_cnt += 1
38             ... #print(f"Received {len(data)} bytes from {addr}: {list(data)}")
39
40     def get_message_r():
41         ... #for getting the received UDP message
42         return message_r
43
44     def get_new_msg_cnt():
45         ... #for getting the UDP message incoming status
46         return new_msg_cnt
47
48     def getLSCommand():
49         global LScmd
50         return LScmd
51
52     def setLSCommand(cmd):
53         global LScmd
54         LScmd = cmd

```

## 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.

```

40     def state_machine():
41         global sys_state
42         ...
43         if sys_state == 0: #from init to operation
44             m_operation()
45
46         elif sys_state == 1: #from norm to reset
47             photosns_stop.set()
48             blink_rgy_stop.set()
49             m_reset() #to reset
50             m_operation()
51
52         elif sys_state == 2: #Reset swarm and plot
53             photosns_stop.set()
54             blink_rgy_stop.set()
55             m_reset() #reset swarm
56             m_operation()

```

## **#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 photosensing 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.

```

112  def m_operation():
113      global sys_state
114      global message_r
115
116      sys_state = 1
117      GPIO.output(led_w, GPIO.HIGH)
118      print("#####")
119      print("In operation mode")
120      print("#####")
121
122      photosns_stop.clear() #clear the stop event
123      photosns_thread = threading.Thread(target=photo_sns, daemon=True)
124      photosns_thread.start()
125
126      blink_rgy_stop.clear() #clear the stop event
127      blink_RGY_thread = threading.Thread(target=blink_rgy_led, daemon=True)
128      blink_RGY_thread.start()
129
130      m_plot() #to plot state
131
132      time.sleep(0.2)

```

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.

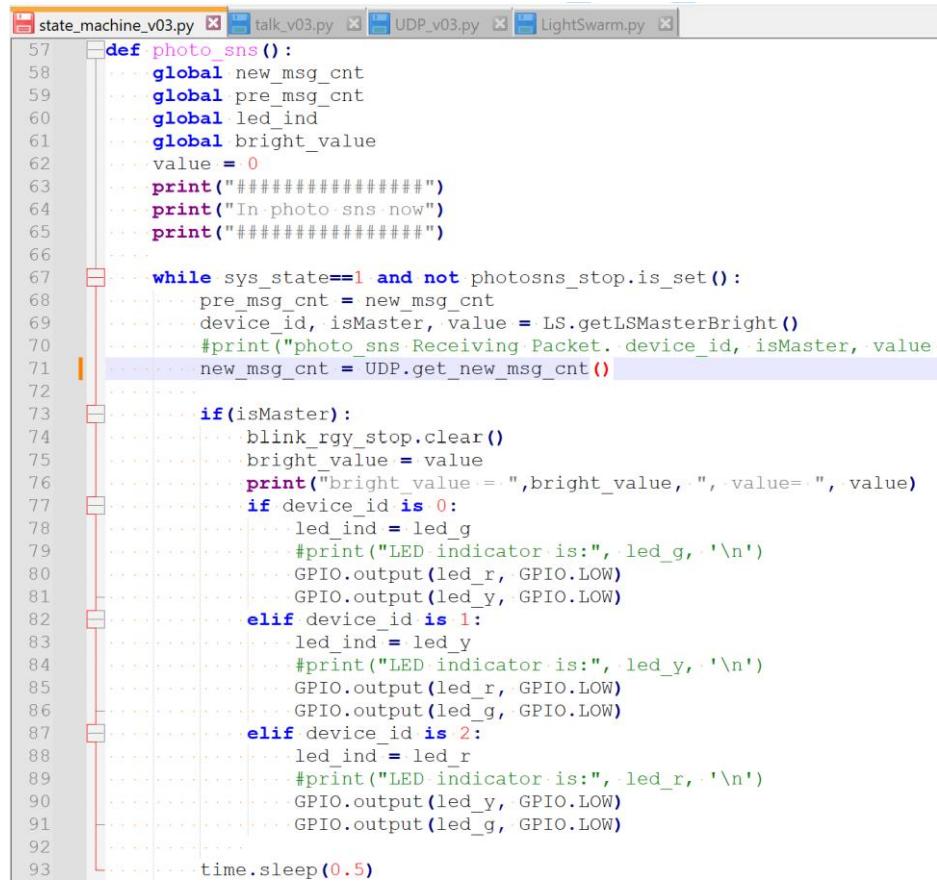
```

137  def m_plot():
138      global sys_state
139      global message_r
140      global plot_enb
141
142      sys_state = 2
143      plot_enb = True
144
145      PLOT.plot_stop.clear()
146      plot_thread = threading.Thread(target=PLOT.collect_data, daemon=True)
147      plot_thread.start()
148
149      print("#####")
150      print("Plot")
151      print("#####")

```

## #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.



```

57     def photo_sns():
58         global new_msg_cnt
59         global pre_msg_cnt
60         global led_ind
61         global bright_value
62         value = 0
63         print("#####")
64         print("In photo sns.now")
65         print("#####")
66
67         while sys_state==1 and not photosns_stop.is_set():
68             pre_msg_cnt = new_msg_cnt
69             device_id, isMaster, value = LS.getLSMasterBright()
70             #print("photo_sns Receiving Packet. device_id, isMaster, value")
71             new_msg_cnt = UDP.get_new_msg_cnt()
72
73             if(isMaster):
74                 blink_ryg_stop.clear()
75                 bright_value = value
76                 print("bright_value = ",bright_value," value= ",value)
77                 if device_id is 0:
78                     led_ind = led_g
79                     #print("LED indicator is:", led_g, '\n')
80                     GPIO.output(led_r, GPIO.LOW)
81                     GPIO.output(led_y, GPIO.LOW)
82                 elif device_id is 1:
83                     led_ind = led_y
84                     #print("LED indicator is:", led_y, '\n')
85                     GPIO.output(led_r, GPIO.LOW)
86                     GPIO.output(led_g, GPIO.LOW)
87                 elif device_id is 2:
88                     led_ind = led_r
89                     #print("LED indicator is:", led_r, '\n')
90                     GPIO.output(led_y, GPIO.LOW)
91                     GPIO.output(led_g, GPIO.LOW)
92
93             time.sleep(0.5)

```

**#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.

```

153     def m_reset():
154         global sys_state, plot_enb
155         ...
156         sys_state = 3
157
158         UDP.setLSCommand("RESETSWARM") #setting LS cmd to UDP layer
159         print("#####")
160         print("Reset Swarm")
161         print("#####")
162         plot_enb = False
163         PLOT.ex_log() #<--log data
164         PLOT.reset_plot() #<--resetting plot time and data
165         ...
166         GPIO.output(led_y, GPIO.HIGH)
167         GPIO.output(led_w, GPIO.HIGH)
168         GPIO.output(led_r, GPIO.LOW)
169         GPIO.output(led_w, GPIO.LOW)
170         GPIO.output(led_g, GPIO.LOW)
171         time.sleep(3)
172         GPIO.output(led_y, GPIO.LOW)
```

## 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.

```

42     'def collect_data():
43         """Background thread collects sensor data"""
44         ...
45         ...global t0, xs0, ys0, xs1, ys1, xs2, ys2, current_time
46         ...global master_count
47         ...
48         ...device_id = 99
49         ...value = 0
50
51         print("[COLLECT] Data collection thread started")
52
53         while not plot_stop.is_set():
54             print("Collecting Data!!!!!!!!!!!!!!-//////////")
55
56             try:
57                 ...device_id_, isMaster_, value_ = LS.getLSSMasterBright()
58                 if isMaster_:
59                     ...device_id = device_id_
60                     ...value = value_
61                 except Exception as e:
62                     ...print(f"[ERROR] getLSSMasterBright failed: {e}")
63                     ...time.sleep(0.1)
64                 ...continue
65
66                 ...current_timestamp = time.time() - t0
67
68                 if device_id == 0:
69                     ...if master_count[device_id] >= 30:
70                         ...master_count[0] = 0
71                         ...master_count[1] = 0
72                         ...master_count[2] = 0
73
74                     ...master_count[device_id] += 1
75                     ...xs0 = np.append(xs0, current_timestamp)
76                     ...ys0 = np.append(ys0, value)
77                     ...current_time = xs0[-1]
78
79                     ...if current_time > WINDOW:
80                         ...mask = xs0 >= (current_time - WINDOW)
81                         ...xs0 = xs0[mask]
82                         ...ys0 = ys0[mask]
```

```

84     ..... elif device_id == 1:
85         ..... if master_count[device_id] >= 30:
86             ..... master_count[0] = 0
87             ..... master_count[1] = 0
88             ..... master_count[2] = 0
89
90         ..... master_count[device_id] += 1
91         ..... xs1 = np.append(xs1, current_timestamp)
92         ..... ys1 = np.append(ys1, value)
93         ..... current_time = xs1[-1]
94
95         ..... if current_time > WINDOW:
96             ..... mask = xs1 >= (current_time - WINDOW)
97             ..... xs1 = xs1[mask]
98             ..... ys1 = ys1[mask]
99
100    ..... elif device_id == 2:
101        ..... if master_count[device_id] >= 30:
102            ..... master_count[0] = 0
103            ..... master_count[1] = 0
104            ..... master_count[2] = 0
105
106        ..... master_count[device_id] += 1
107        ..... xs2 = np.append(xs2, current_timestamp)
108        ..... ys2 = np.append(ys2, value)
109        ..... current_time = xs2[-1]
110
111        ..... if current_time > WINDOW:
112            ..... mask = xs2 >= (current_time - WINDOW)
113            ..... xs2 = xs2[mask]
114            ..... ys2 = ys2[mask]
115
116    ..... print("master_count = ", master_count[0], master_count[1], master_count[2])
117
118    ..... time.sleep(1)

```

## #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.

```

134 def reset_plot():
135     """Reset all plot data"""
136     global t0, xs0, ys0, xs1, ys1, xs2, ys2, current_time
137     global master_count, reset_counter
138
139     reset_counter+=1
140
141     plot_stop.set() #<---stopping plot thread
142
143     t0 = time.time()
144     print("Resetting Data!!!!!!!!!!!!!!")
145     time.sleep(5)
146
147     xs0 = np.array([])
148     ys0 = np.array([])
149     xs1 = np.array([])
150     ys1 = np.array([])
151     xs2 = np.array([])
152     ys2 = np.array([])
153
154     master_count = [0, 0, 0]
155
156     current_time = 0
157
158     print(f"[RESET] Counter: {reset_counter}")

```

```

def plot():
    global t0, xs0, ys0, xs1, ys1, xs2, ys2, current_time, line0, line1, line2, master_count, fig
    device_id = 99
    value = 0

    ax1.set_xlabel("Time (s)")
    ax1.set_ylabel("PhotoCell Reading")
    ax1.set_title("LightSwarm Brightness")
    ax1.set_ylim(0, 5000) # adjust based on brightness range
    ax2.set_xlabel("Device ID")
    ax2.set_ylabel("Accumulative Master Count")
    ax2.set_title("Device Master Chart")
    ani = animation.FuncAnimation(fig, update_plot, interval=1000)
    plt.show(block=False)

    while not plot_stop.is_set():
        if plot_reset_flag.is_set(): #reset plot
            print("Resetting Plot!!\n")
            xs0.clear(); ys0.clear()
            xs1.clear(); ys1.clear()
            xs2.clear(); ys2.clear()

            line0.set_xdata([]); line0.set_ydata([])
            line1.set_xdata([]); line1.set_ydata([])
            line2.set_xdata([]); line2.set_ydata([])

            ax1.set_xlim(0, WINDOW)
            plot_reset_flag.clear()

```

## #ex\_log()

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

```

150 def ex_log():
151     ... global xs0, ys0, xs1, ys1, xs2, ys2, current_time, master_count
152     .... timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
153
154     ... os.makedirs("log", exist_ok=True)
155     ... with open(f"log/{timestamp}.txt", "w") as f:
156         ... f.write("== LightSwarm Data Log ==\n")
157         ... f.write(f"Timestamp: {timestamp}\n\n")
158
159         ... #Log master counter
160         ... f.write("Master Count per Device:\n")
161         ... f.write(f"Device 0: {master_count[0]}\n")
162         ... f.write(f"Device 1: {master_count[1]}\n")
163         ... f.write(f"Device 2: {master_count[2]}\n\n")
164
165         ... #Log arrays
166         ... f.write("\n== Device 0 Data (xs0, ys0) ==\n")
167         ... for t, v in zip(xs0, ys0):
168             ... f.write(f"{t:.2f}, {v}\n")
169
170         ... f.write("\n== Device 1 Data (xs1, ys1) ==\n")
171         ... for t, v in zip(xs1, ys1):
172             ... f.write(f"{t:.2f}, {v}\n")
173
174         ... f.write("\n== Device 2 Data (xs2, ys2) ==\n")
175         ... for t, v in zip(xs2, ys2):
176             ... f.write(f"{t:.2f}, {v}\n")
177
178     ... print(f"[LOG] Exported data to {timestamp}.txt")

```

```

*~/Desktop/ECPS216/swarm2/log/20251118_154955.txt - Mousepad
File Edit Search View Document Help
== LightSwarm Data Log ==
Timestamp: 20251118_154955

Master Count per Device:
Device 0: 6
Device 1: 5
Device 2: 0

== Device 0 Data (xs0, ys0) ==
31.56, 2368
32.58, 2368
33.60, 2368
34.61, 2368
35.62, 2368
36.64, 2354
37.65, 2354
38.66, 2354
39.68, 2308
40.69, 2308
41.70, 2308
42.72, 2308
43.73, 2308
44.75, 2252
72.17, 2559
73.19, 2559
74.20, 2555
75.22, 2551

```

## #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.

```

7   from flask import Flask, render_template_string, jsonify
8   import plot as PLOT
9   import state_machine_v05 as STATE
10
11 web = Flask(__name__)

```

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

```

13 HTML_TEMPLATE = """
14 <!DOCTYPE html>
15 <html>
16 <head>
17 ...<title>LightSwarm·Real-Time·Dashboard</title>
18 ...<meta name="viewport" content="width=device-width, initial-scale=1.0">
19 ...<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
20 ...<style>
21 ...    * {
22 ...        margin: 0;
23 ...        padding: 0;
24 ...        box-sizing: border-box;
25 ...    }
26 ...
27 ...    body {
28 ...        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
29 ...        background: linear-gradient(135deg, #0d0d0d 0%, #1a1a1a 100%);
30 ...        color: white;
31 ...        min-height: 100vh;
32 ...        padding: 20px;
33 ...    }
34 ...
35 ...    .container {
36 ...        max-width: 1400px;
37 ...        margin: 0 auto;
38 ...    }
39 ...
40 ...    .header {
41 ...        text-align: center;
42 ...        margin-bottom: 30px;
43 ...    }
44 ...
45 ...    h1 {

```

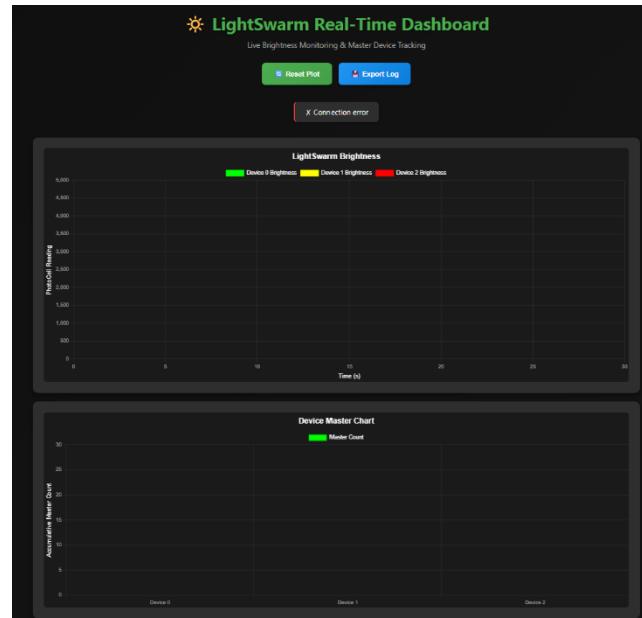
5 endpoints were developed. They are `home`, `get_data`, `reset_plot`, `export_log` and `status`.

```

473     @web.route("/")
474     def home():
475         """Main dashboard page"""
476         return render_template_string(HTML_TEMPLATE)
477
478     @web.route("/data")
479     def get_data():
480         """API endpoint: Get current plot data"""
481         data = PLOT.get_plot_data()
482         return jsonify(data)
483
484     @web.route("/reset", methods=["POST"])
485     def reset_plot():
486         """API endpoint: Reset plot"""
487         STATE.state_machine()
488         return jsonify({
489             "status": "success",
490             "message": "Plot reset successfully!"
491         })
492
493     @web.route("/export", methods=["POST"])
494     def export_log():
495         """API endpoint: Export log"""
496         try:
497             PLOT.ex_log()
498             return jsonify({
499                 "status": "success",
500                 "message": "Log exported successfully!"
501             })
502         except Exception as e:
503             return jsonify({
504                 "status": "error",
505                 "message": f"Export failed: {str(e)}"
506             }), 500
507
508     @web.route("/status")
509     def status():
510         """API endpoint: System status"""
511         data = PLOT.get_plot_data()
512         return jsonify({
513             "status": "running",
514             "master_count": data['master_count'],
515             "window_size": PLOT.WINDOW,
516             "data_points": {
517                 "device0": len(data['time0']),
518                 "device1": len(data['time1']),
519                 "device2": len(data['time2'])
520             }
521         })

```

This is 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.

```

6   MAT_VAL = [0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xFE, 0xFF]
7   DATA = [0, 0, 0, 0, 0, 0, 0, 0, 0]
8   raw_value = 0
9
10  class LED_MAT:
11      def __init__(self, name):
12          # Constructor (runs when object is created)
13          self.name = name
14          self.status = False
15
16      def spi_init(self, bus, device, max_spd, mode):
17
18          self.spi = spidev.SpiDev()
19          self.spi.open(bus, device) # bus 0, device 0 (spidev0.0)
20          self.spi.max_speed_hz = max_spd # 1 MHz
21          self.spi.mode = mode
22          self.status = True
23
24      def mat_init(self):
25          # MAX7219 init
26          self.show(0x09, 0x00) # no decode
27          self.show(0x0A, 0x08) # brightness (0x00-0x0F)
28          self.show(0x0B, 0x07) # scan all 8 rows
29          self.show(0x0C, 0x01) # exit shutdown mode
30          self.show(0x0F, 0x00) # display test off
31
32          for row in range(1, 9):
33              self.show(row, 0x00)
34              time.sleep(0.5)

```

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.

```

36     .....def.data_queue(self, data):
37         .....DATA[8] = DATA[7]
38         .....DATA[7] = DATA[6]
39         .....DATA[6] = DATA[5]
40         .....DATA[5] = DATA[4]
41         .....DATA[4] = DATA[3]
42         .....DATA[3] = DATA[2]
43         .....DATA[2] = DATA[1]
44         .....DATA[1] = DATA[0]
45         .....DATA[0] = data
46
47     .....def.show(self, reg, data):
48         .....if self.status is True:
49             .....self.spi.xfer2([reg, data])
50         .....else:
51             .....print("LED Matrix Not Yet Init YO~~")
52
53     .....def.close(self):
54         .....self.spi.xfer2([0x0C, 0x00])
55
56     .....def.show_swarm(self):
57         .....global raw_value
58
59     .....while True:
60         .....device_id_, isMaster_, value_ = LS.getLSSMasterBright()
61         .....if(isMaster_):
62             .....device_id = device_id_
63             .....raw_value = value_
64             .....raw_value /= 511 #to 0 ~ 8
65
66             .....if raw_value > 8:
67                 .....raw_value = 8
68             .....if raw_value < 0:
69                 .....raw_value = 0
70             .....raw_value = int(raw_value)
71             .....value = MAT_VAL[raw_value] #get hex
72             .....self.data_queue(value) #push data to DATA[]
73             .....for i in range(0, 8):
74                 .....self.show(i+1, DATA[i])
75
76             .....time.sleep(1)

```

## Part 2-1 ESP WiFi setup and packet delivery

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

```
LightSwarm.ino
12  char ssid[] = "iPhoneKL"; // your network SSID (name)
13  char pass[] = [REDACTED]; // your network password
14
15  unsigned int localPort = 1996; // local port to listen for UDP packets
16  IPAddress serverAddress = IPAddress(255, 255, 255, 255); // default no IP Address

114 Serial.print("Connecting to ");
115 Serial.println(ssid);
116 WiFi.begin(ssid, pass);
117
118 // initialize Swarm Address - we start out as swarmID of 0
119
120 ~ while (WiFi.status() != WL_CONNECTED) {
121   delay(500);
122   Serial.print(".");
123 }
124 Serial.println("");
125
126 Serial.println("WiFi connected");
127 Serial.println("IP address: ");
128 Serial.println(WiFi.localIP());
129
130 Serial.println("Starting UDP");
131
132 udp.begin(localPort);      // bind & listen on port
133 //udp.setBroadcast(true);    // allow broadcast TX
```

**Swarm data array is initialized here.**

```
137 // initialize light sensor and arrays
138 int i;
139 for (i = 0; i < SWARMSIZE; i++)
140 {
141   swarmAddresses[i] = 0;
142   swarmClear[i] = 0;
143   swarmTimeStamp[i] = -1;
144 }
145 swarmClear[mySwarmID] = 0;
146 swarmTimeStamp[mySwarmID] = 1; // I am always in time to myself
147 clearColor = swarmClear[mySwarmID];
148 swarmVersion[mySwarmID] = VERSIONNUMBER;
149 swarmState[mySwarmID] = masterState;
150 Serial.print("clearColor=");
151 Serial.println(clearColor);
152 // set SwarmID based on IP address
153 localIP = WiFi.localIP();
154 swarmAddresses[0] = localIP[3];
155 mySwarmID = 0;
156 Serial.print("MySwarmID=");
157 Serial.println(mySwarmID);
```

## #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.

```

159  /*Mutesx to protect shared swarm data field*/
160  swarmMutex = xSemaphoreCreateMutex();
161  /*Task Creation*/
162  xTaskCreatePinnedToCore(Task_PhotoSns, "Task_PhotoSns", 4096, NULL, 1, &xHandle_Task_PhotoSns, 1);
163  xTaskCreatePinnedToCore(Task_UDP_Receive, "Task_UDP_Receive", 4096, NULL, 2, &xHandle_Task_UDP_Receive, 1);
164  xTaskCreatePinnedToCore(Task_UDP_Send, "Task_UDP_Send", 4096, NULL, 1, &xHandle_Task_UDP_Send, 1);
165  xTaskCreatePinnedToCore(Task_Swarm, "Task_Swarm", 4096, NULL, 1, &xHandle_Task_Swarm, 1);
166  xTaskCreatePinnedToCore(Task_LED_Blink, "Task_LED_Blink", 4096, NULL, 1, &xHandle_Task_LED_Blink, 1);
167  xTaskCreatePinnedToCore(Task_Monitor, "Task_Monitor", 4096, NULL, 1, NULL, 1);

```

## #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.

```

356 void Task_UDP_Receive(void *pvParameters){
357
358     for(;;){
359         int cb = udp.parsePacket();
360         if (cb) {
361             udp.read(packetBuffer, PACKET_SIZE);
362             xSemaphoreTake(swarmMutex, portMAX_DELAY);
363             // update swarm arrays and clearColor
364             Incoming_Packet_Helper(packetBuffer);
365             xSemaphoreGive(swarmMutex);
366         }
367         vTaskDelay(100 / portTICK_PERIOD_MS);
368     }
369 }
```

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

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

```

371 void Task_UDP_Send(void *pvParameters){
372     for(;;){
373         vTaskDelay(1); // let RX run first
374         broadcastARandomUpdatePacket();
375         vTaskDelay(1000 / portTICK_PERIOD_MS);
376     }
377 }
```

```

206 void broadcastARandomUpdatePacket()
207 {
208     int sendToLightSwarm = 255;
209     int randomDelay;
210     randomDelay = random(0, MAXDELAY);
211     Serial.print("Delay = ");
212     Serial.print(randomDelay);
213     Serial.print("ms : ");
214
215     vTaskDelay(randomDelay / portTICK_PERIOD_MS);
216
217     IPAddress sendSwarmAddress(255, 255, 255, sendToLightSwarm); // my Swarm Address
218     sendLightUpdatePacket(sendSwarmAddress);
219 }
```

## 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().

```

379  void Task_Swarm(void *pvParameters){
380  for(;;){
381      // Check to see if I am master!
382      checkAndSetIfMaster();
383      vTaskDelay(500 / portTICK_PERIOD_MS);
384  }
385 }
```

```

221 void checkAndSetIfMaster()
222 {
223     int i;
224     int howLongAgo;
225     for (i = 0; i < SWARMSIZE; i++)
226     {
227
228 #ifdef DEBUG
229     Serial.print("swarmClear[");
230     Serial.print(i);
231     Serial.print("] = ");
232     Serial.print(swarmClear[i]);
233     Serial.print("  swarmTimeStamp[");
234     Serial.print(i);
235     Serial.print("] = ");
236     Serial.println(swarmTimeStamp[i]);
237 #endif
238     Serial.print("#");
239     Serial.print(i);
240     Serial.print("/");
241     Serial.print(swarmState[i]);
242     Serial.print("/");
243     Serial.print(swarmVersion[i]);
244     Serial.print(":");
245     // age data
246     howLongAgo = millis() - swarmTimeStamp[i]
```

## #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.

```
LightSwarm.ino
530
531 int Photo_Sns(void){
532     int val = 0u;
533
534     val = analogRead(PHOTO_SNS_PIN);
535     val = MovAvg(val, &PHOTO_READINGS[0]);/*Moving average*/
536     data_cnt+=1;
537     sprintf(photo_sns_buffer, "%d", val); /*convert int → text*/
538     Serial.print("Photo Sns Read=");
539     Serial.print(photo_sns_buffer);
540     Serial.println(" ");
541     return val;
542 }
543
544 uint16_t MovAvg(uint16_t input, uint16_t *buffer){
545     /*Buffer with 5 elements*/
546     float avg = 0.0f;
547
548     buffer[4] = buffer[3];
549     buffer[3] = buffer[2];
550     buffer[2] = buffer[1];
551     buffer[1] = buffer[0]; /*pop out old element and push new*/
552     buffer[0] = input;
553
554     avg = (buffer[0]+buffer[1]+buffer[2]+buffer[3]+buffer[4])/5;
555
556     return avg;
557 }
```

## #Task LED Blink function

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.

```
99     pinMode(LED_BUILTIN, OUTPUT);
100    analogWriteResolution(LED_BLE, 12); //0-4095
101    analogWriteFrequency(LED_BLE, 5000); // 0 kHz global PWM frequency
102    analogWrite(LED_BLE, 0); //duty
103    /*Start*/
104
105    /*Task LED Blink*/
106    void Task_LED_Blink(void *pvParameters){
107        for(;;){
108            Serial.print("Bright Value = ");
109            Serial.print(swarmClear[mySwarmID]);
110            Serial.print('\n');
111            analogWrite(LED_BLE, swarmClear[mySwarmID]);
112            //vTaskDelay(BLNK_MS / portTICK_PERIOD_MS);
113            vTaskDelay(50 / portTICK_PERIOD_MS);
114        }
115    }
```

## 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.

This function is also created as a FreeRTOS task with cycle time 200ms.

<pre> 33     uint8_t led_seg[] = { 34         TEN_SEG_LED_1, 35         TEN_SEG_LED_2, 36         TEN_SEG_LED_3, 37         TEN_SEG_LED_4, 38         TEN_SEG_LED_5, 39         TEN_SEG_LED_6, 40         TEN_SEG_LED_7, 41         TEN_SEG_LED_8, 42         TEN_SEG_LED_9, 43         TEN_SEG_LED_10, 44     }; </pre>	<pre> 468    void Task_LED_Bar(void *pvParameters){ 469        uint8_t i = 0; 470        uint8_t k = 0; 471        uint8_t j = 0; 472        for(;;){ 473            i = clearColor/409; 474            for(j = i; j&lt;11; j++){ 475                digitalWrite(led_seg[j], LOW); 476            } 477            for(k = 0; k&lt;i; k++){ 478                digitalWrite(led_seg[k], HIGH); 479            } 480            vTaskDelay(200 / portTICK_PERIOD_MS); 481        } 482    } 483 } 484 }</pre>
--	---

## -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.

```

10
11     web = Flask(__name__)
12
13     HTML_TEMPLATE = """
472
473     @web.route("/")
474     def home():
475
476         @web.route("/data")
477         def get_data():
478
479             @web.route("/reset", methods=["POST"])
480             def reset_plot():
481
482                 @web.route("/export", methods=["POST"])
483                 def export_log():
484
485                     @web.route("/status")
486                     def status():

```

## -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("Starting LightSwarm Web Dashboard (Real-Time Plotting)")
    print("=" * 60)

    print("\nFqDashboard URLs:")
    print("  Local: http://localhost:5000")
    print("  Network: http://<raspberry-pi-ip>:5000")
    print("\nPress 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():

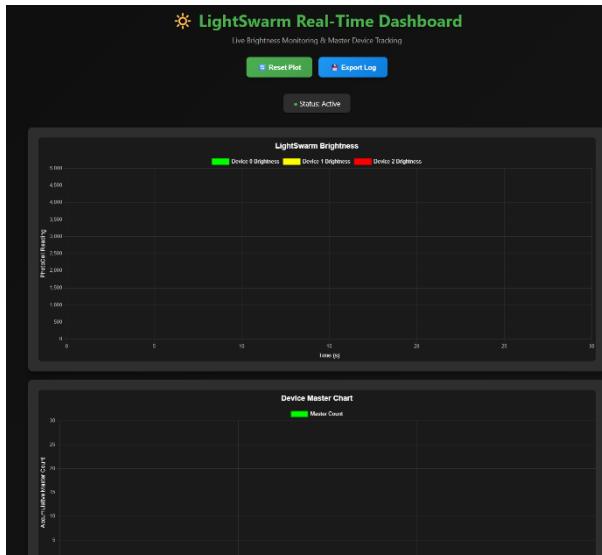
                @web.route("/status")
                def status():


```

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        |

#### \*Initialization



#### \*Plotting



**\*Data > 30secs**



**\*Reset and refresh**

