



FreeRTOS Automatic Car Project

[Preface](#)

- [I. Project Description](#)
- [II. Project Goals](#)

[Background Knowledge](#)

- [I. Introduction to RTOS](#)
 - [Overview of RTOS](#)
 - [Multitasking and Concurrency](#)
 - [Scheduling](#)
 - [Real-Time Scheduling and Priority](#)
 - [Example Scenario](#)
- [II. Introduction to FreeRTOS](#)
- [III. FreeRTOS Coding Tips](#)
 - [xTaskCreate\(\) & vTaskStartScheduler\(\) 建立任務](#)
 - [taskENTER_CRITICAL\(\) & taskEXIT_CRITICAL\(\)](#)
 - [vTaskDelete\(\)](#)
 - [vTaskPrioritySet\(\) & uxTaskPriorityGet\(\)](#)
 - [vTaskSuspend\(\) & vTaskResume\(\)](#)
 - [vTaskSuspendAll\(\) & vTaskResumeAll\(\)](#)
 - [xTaskGetTickCount \(\) & vTaskDelay\(\) & vTaskDelayUntil\(\) & vTaskSuspend\(NULL \)](#)
 - [uxTaskGetNumberOfTasks\(\)](#)
 - [vTaskList\(\) 取得CPU Usage](#)
 - [vTaskGetRunTimeStats\(\)](#)
 - [xQueueHandle & xQueueCreate\(\) & xQueueReceive\(\) & xQueueSend\(\)](#)
 - [Mutex: xSemaphoreHandle & xSemaphoreCreateMutex\(\) & xSemaphoreTake\(\) & xSemaphoreGive\(\)](#)
 - [Binary Semaphore: xSemaphoreHandle & vSemaphoreCreateBinary\(\) & xSemaphoreTake\(\) & xSemaphoreGive\(\)](#)
 - [Counting Semaphore: xSemaphoreHandle & xSemaphoreCreateCounting\(\) & xSemaphoreTake\(\) & xSemaphoreGive\(\)](#)

[Development Tools](#)

- [I. Development Platform](#)
- [II. Project Package](#)
- [III. Technologies Used](#)
- [IV. Programming Language Used](#)

[Project Scenario](#)

- [I. IN MOTION : AUTO](#)
- [II. IN MOTION : TRACK](#)
- [III. MOTIONLESS](#)
- [IV. OVERRIDE 1](#)
- [V. OVERRIDE 2](#)

[Main Program](#)

- [I. ESP32 Program](#)
 - [Module Setup](#)
 - [FreeRTOS Tasks Creation](#)
 - [Main Loop: Listen to WiFi command](#)
 - [Main Loop: Listen to Bluetooth command](#)
 - [Main Loop: Restart ESP32 if WiFi Client Not Connected](#)
 - [Obstacle Avoidance Task](#)
 - [Line Tracking Task](#)
 - [WiFi Connection Watch Dog](#)
- [II. Control Panel \(run on PC\)](#)
 - [Control panel in OVERRIDE 2 STATE](#)
 - [Control panel in OVERRIDE 1 STATE](#)
 - [Control panel in MOTIONLESS/OVERRIDE 1 STATE, Obstacle Detected](#)

[Control panel in MOTIONLESS/OVERRIDE 1 STATE, No Track Detected](#)

[Problem Discussion](#)

[I. Automatic Car Motor Control Race Condition](#)

[Without Mutex or Semaphore](#)

[Use vTaskSuspend\(\) and vTaskResume\(\) to Stop Line Tracking Task Temporarily](#)

[With Mutex or Semaphore](#)

[II. Late Start of Line Tracking](#)

[Using Binary Semaphore](#)

[Using Two EventGroup Bits](#)

[Using Three EventGroup Bits](#)

Preface

I. Project Description

This project builds an Automatic Car System using FreeRTOS. This automatic car is capable of multitasking such as detecting obstacles, tracking lines and receiving WiFi and Bluetooth command concurrently. This project also involves building a control panel for the main controller to override the functions with command send through WiFi.

II. Project Goals

1. Build an automatic car which is capable of multitasking such as detecting obstacles, tracking lines and receiving WiFi and Bluetooth command concurrently.
2. Build a control panel for the main controller to override the functions with command send through WiFi.

Background Knowledge

I. Introduction to RTOS

Overview of RTOS

- **Real-Time Operating System (RTOS):** A specialized operating system designed for embedded systems requiring deterministic, timely responses to events. It's often lightweight, making it suitable for devices with constraints on memory, compute, and power.

Multitasking and Concurrency

- **Multitasking:** RTOSes, like FreeRTOS, can execute multiple tasks concurrently by rapidly switching between them, although a single-core processor can only run one task at a time.
- **Tasks vs. Threads:** RTOSes often refer to tasks instead of threads, as they usually don't have virtual memory.
- **Benefits of Multitasking:** Partitioning applications into manageable tasks allows for easier testing, team collaboration, and code reuse. The RTOS kernel manages timing and sequencing.

Scheduling

- **Scheduler:** Part of the RTOS kernel that decides which task to execute. Tasks can be paused and resumed, allowing for prioritization based on real-time requirements.
- **Task States:** Tasks can yield, sleep, or block, allowing the kernel to manage and select the appropriate task to run according to the scheduling algorithm.

Real-Time Scheduling and Priority

- **Scheduling Policy:** Real-time systems use priority-based scheduling, ensuring the highest-priority task gets processing time to meet deadlines.
- **Task Prioritization:** Tasks are assigned priorities based on deadlines and the consequences of missing them. For example, a control task with strict timing is prioritized over a key handler task with a more lenient timing requirement.

Example Scenario

- In a system with a **keypad and control function**, a key handler task gives feedback on user input, while a control task performs time-sensitive operations.
- The control task, needing timely execution, is assigned higher priority. The RTOS switches between tasks to ensure deadlines are met, pausing lower-priority tasks like the idle task or the key handler if the control task is ready.

This setup ensures that the system can respond to real-world events within specified time constraints, fulfilling the requirements of a real-time embedded system.

II. Introduction to FreeRTOS

FreeRTOS is a lightweight Real-Time Operating System (RTOS) designed to run on microcontrollers, which are compact and resource-limited processors typically used in embedded applications. These applications often involve specific, dedicated tasks, making a full-featured RTOS unnecessary or impractical. FreeRTOS provides essential real-time scheduling, inter-task communication, timing, and synchronization primitives, making it a real-time kernel rather than a full RTOS. Additional features, like networking or a command console interface, can be added as needed.

III. FreeRTOS Coding Tips

xTaskCreate() & vTaskStartScheduler() 建立任務

1. Each task has:
 - a. Register set
 - b. Stack area
 - c. Priority
2. 建立任務，沒建立task會執行idle task
 - a. 把AppTaskCreate_Handle當作該Task控制點

```

88 白 xReturn = xTaskCreate((TaskFunction_t )AppTaskCreate, /* 任務入口函數 */
89                                (const char* )"AppTaskCreate",/* 任務名字*/
90                                (uint16_t )512, /* 任務堆疊大小 */
91                                (void* )NULL,/* 任務入口函數參數 */
92                                (UBaseType_t )1, /* 任務的優先順序*/
93                                (TaskHandle_t* )&AppTaskCreate_Handle);/* 任務控制塊指標 */
94
95  /*啟動任務調度 */
96  if(pdPASS == xReturn)
97      vTaskStartScheduler(); /* 啟動任務，開啟調度*/
98  else
99      return -1;

```

3. 建立子任務

```

30  //AppTaskCreate_Handle
31  static TaskHandle_t AppTaskCreate_Handle = NULL;
32  static TaskHandle_t LED_Task_Handle = NULL;
33  static TaskHandle_t LED1_Task_Handle = NULL;
34  static TaskHandle_t LED2_Task_Handle = NULL;

```

也可用 `xTaskHandle`，`xTaskHandle` 跟 `TaskHandle_t` 一樣

```

106 static void AppTaskCreate(void)
107 {
108     BaseType_t xReturn = pdPASS; /* 定義一個建立資訊返回值，預設為pdPASS */
109
110     taskENTER_CRITICAL();          //進入臨界區
111
112     /* 建立LED_Task任務 */
113     xReturn = xTaskCreate((TaskFunction_t) LED_Task, /* 任務入口函數 */
114                          (const char*) "LED_Task", /* 任務名字 */
115                          (uint16_t) 512, /* 任務堆疊大小 */
116                          (void*) NULL, /* 任務入口函數參數 */
117                          (UBaseType_t) 2, /* 任務的優先順序 */
118                          (TaskHandle_t*) &LED_Task_Handle); /* 任務控制塊指標 */
119
120     if(pdPASS == xReturn)
121         printf("LED_Task!\r\n");
122     /* 建立LED1_Task任務 */
123     xReturn = xTaskCreate((TaskFunction_t) LED1_Task, /* 任務入口函數 */
124                          (const char*) "LED1_Task", /* 任務名字 */
125                          (uint16_t) 512, /* 任務堆疊大小 */
126                          (void*) NULL, /* 任務入口函數參數 */
127                          (UBaseType_t) 3, /* 任務的優先順序 */
128                          (TaskHandle_t*) &LED1_Task_Handle); /* 任務控制塊指標 */
129
130     if(pdPASS == xReturn)
131         printf("LED1_Task!\r\n");
132     /* 建立LED2_Task任務 */
133     xReturn = xTaskCreate((TaskFunction_t) LED2_Task, /* 任務入口函數 */
134                          (const char*) "LED2_Task", /* 任務名字 */
135                          (uint16_t) 512, /* 任務堆疊大小 */
136                          (void*) NULL, /* 任務入口函數參數 */
137                          (UBaseType_t) 3, /* 任務的優先順序 */
138                          (TaskHandle_t*) &LED2_Task_Handle); /* 任務控制塊指標 */
139
140     if(pdPASS == xReturn)
141         printf("LED2_Task!\r\n");
142
143     //AppTaskCreate_Handle
144     vTaskDelete(AppTaskCreate_Handle); //刪除AppTaskCreate任務
145
146     taskEXIT_CRITICAL();          //退出臨界區
147 }

```

4. 撰寫各任務內容

a. sample code

```

153 static void LED_Task(void* parameter)
154 {
155     while (1)
156     {
157         LED0( ON );
158         //printf("LED_Task Running,LED1_ON\r\n");
159         vTaskDelay(5000); /* 延時500個tick */
160
161         LED0( OFF );
162         //printf("LED_Task Running,LED1_OFF\r\n");
163         vTaskDelay(5000); /* 延時500個tick */
164     }
165 }
166
167 static void LED1_Task(void* parameter)
168 {
169     while (1)
170     {
171         LED1( ON );
172         //printf("LED_Task Running,LED1_ON\r\n");
173         vTaskDelay(1000); /* 延時500個tick */
174
175         LED1( OFF );
176         //printf("LED_Task Running,LED1_OFF\r\n");
177         vTaskDelay(1000); /* 延時500個tick */
178     }
179 }

```

b. 任務中通常有:

- i. 一個無窮迴圈
- ii. 呼叫OS Service

taskENTER_CRITICAL() & taskEXIT_CRITICAL()

1. 在臨界區中執行的程式不會被打擾
2. `taskENTER_CRITICAL()` ⇒ `portENTER_CRITICAL()` ⇒ `vPortEnterCritical()` ⇒ `portDISABLE_INTERRUPTS()`

vTaskDelete()

1. `vTaskDelete(AppTaskCreate_Handle); //Delete AppTaskCreate`
2. 不會馬上刪除，需回收時間 (遇return)

vTaskPrioritySet() & uxTaskPriorityGet()

1. `vTaskPrioritySet(LED_Task_Handle,11)`
2. `Task_pri=uxTaskPriorityGet(LED_Task_Handle); //get priority`
3. `printf("New priority is %ld",Task_pri);`
4. 數字越大，優先權越高

vTaskSuspend() & vTaskResume()

1. suspend task and resume task
2. `vTaskSuspend(LED_Task_Handle);`
3. `vTaskResume(LED_Task_Handle);`

vTaskSuspendAll() & vTaskResumeAll()

1. suspend所有除了自己的Task，OS通常也直接不作用(eg. delay won't work)

xTaskGetTickCount () & vTaskDelay() & vTaskDelayUntil() & vTaskSuspend(NULL)

1. sample code 1: `xTaskGetTickCount ()` & `vTaskDelay()`

```
95 // set changable delay time
96 void vTask1( void *pvParameters )
97 {
98     portTickType xDelay, xNextTime;
99     /* As per most tasks, this task is implemented in an infinite loop. */
100     while(1)
101     {
102         LED0_TOGGLE;
103         // printf("Task1 is running \r\n");
104         xDelay = xNextTime - xTaskGetTickCount ();
105         xNextTime += ( portTickType ) 1000;
106
107         if( xDelay <= ( portTickType ) 1000 )
108         {
109             vTaskDelay( xDelay );
110         }
111     }
112 }
```

- i. `portTickType` 是 `xTaskGetTickCount ()` 回傳值的data type
2. sample code 2: `xTaskGetTickCount ()` & `vTaskDelayUntil()`

```
114 void vTask2( void *pvParameters )
115 {
116     portTickType xLastWakeTime;
117
118     const portTickType xFrequency = 100;
119
120     // Initialise the xLastWakeTime variable with the current time.
121     xLastWakeTime = xTaskGetTickCount();
122
123     /* As per most tasks, this task is implemented in an infinite loop. */
124     while(1)
125     {
126         //printf("Task2 is running\r\n");
127         LED1_TOGGLE;
128         vTaskDelayUntil( &xLastWakeTime, xFrequency );
129     }
130 }
131 }
```

- i. `vTaskDelayUntil(&xLastWakeTime, xFrequency);`
 - ii. 從 `xLastWakeTime` 開始向後數 `xFrequency`
3. sample code 3: `vTaskSuspend(NULL)`
- i. `vTaskSuspend(NULL)` 把自己暫停掉

```

133 void vTask3( void *pvParameters )
134 {
135     while(1)
136     {
137         vTaskSuspend( NULL );
138         LED2_TOGGLE;
139     }
140 }

```

uxTaskGetNumberOfTasks()

1. 取得目前存在多少Task

vTaskList() 取得CPU Usage

1. Table produced

Name	State	Priority	Stack	Num
Print	R	4	331	29
Math7	R	0	417	7
Math8	R	0	407	8
QConsB2	R	0	53	14
QProdB5	R	0	52	17
QConsB4	R	0	53	16
SEM1	R	0	50	27
SEM1	R	0	50	28
IDLE	R	0	64	0
Math1	R	0	436	1
Math2	R	0	436	2

2. State Types:
 - i. B: Block (delay會進入block)
 - ii. R: Ready
 - iii. S: Suspend
 - iv. D: has been deleted, but memory has not yet been freed
3. Stack: 剩下的Stack容量
4. Num: 隨機編號
5. sample code

```

158 static void CPU_Task(void* parameter)
159 {
160     uint8_t CPU_RunInfo[400]; //保存任務執行時間資訊
161
162     while (1)
163     {
164         memset(CPU_RunInfo,0,400); //信息緩衝區清零
165
166         vTaskList((char *)&CPU_RunInfo); //獲取任務執行時間資訊
167
168         printf("-----\r\n");
169         printf("TaskName    TaskNameState    Priority    Remaining stack TaskSerial\r\n");
170         printf("%s", CPU_RunInfo);
171         printf("-----\r\n");
172
173         memset(CPU_RunInfo,0,400); //信息緩衝區清零
174
175         vTaskGetRunTimeStats((char *)&CPU_RunInfo);
176
177         printf("TaskName    Counter    Usage rate\r\n");
178         printf("%s", CPU_RunInfo);
179         printf("-----\r\n");
180         vTaskDelay(5000); /* 延時500個tick */
181     }
182 }

```

vTaskGetRunTimeStats()

1. `vTaskGetRunTimeStats((char *)&CPU_RunInfo);`
2. CPU排程執行此Task的次數
3. produced content:

Task	Abs Time	% Time

uIP	12050	<1%
IDLE	587724	24%
QProdB2	2172	<1%
QProdB3	10002	<1%
QProdB5	11504	<1%
QConsB6	11671	<1%
PolSEM1	60033	2%
PolSEM2	59957	2%
IntMath	349246	14%
MuLow	36619	1%
GenO	579715	24%

xQueueHandle & xQueueCreate() & xQueueReceive() & xQueueSend()

1. `xQueueHandle xQueue;`
2. `xQueue = xQueueCreate(5, sizeof(uint8_t));`
3. 這個Queue建立在**同步化**的機制
4. sample code
 - a. 收資料
 - i. 也可用BaseType_t來收回傳值，0代表收不到資料

```

87 void vTask3( void *pvParameters )
88 {
89     uint8_t Count[5];
90
91     while(1)
92     {
93         if( xQueue != 0 )
94         {
95             //put xQueue's data to Count[0], wait at most portMAX_DELAY
96             xQueueReceive( xQueue, &Count[0], portMAX_DELAY);
97             xQueueReceive( xQueue, &Count[1], portMAX_DELAY);
98             xQueueReceive( xQueue, &Count[2], portMAX_DELAY);
99             xQueueReceive( xQueue, &Count[3], portMAX_DELAY);
100            xQueueReceive( xQueue, &Count[4], portMAX_DELAY);
101        }
102
103        printf("Count0 = %d\r\n", Count[0]);
104        printf("Count1 = %d\r\n", Count[1]);
105        printf("Count2 = %d\r\n", Count[2]);
106        printf("Count3 = %d\r\n", Count[3]);
107        printf("Count4 = %d\r\n", Count[4]);
108    }
109 }

```

b. 塞資料

```

111 void vTask4( void *pvParameters )
112 {
113     portTickType xLastWakeTime;
114     const portTickType xFrequency = 1000;
115     uint8_t ucVar[5] = {0};
116
117     // Initialise the xLastWakeTime variable with the current time.
118     xLastWakeTime = xTaskGetTickCount();
119
120     while(1)
121     {
122         ucVar[0]++;
123         ucVar[1]++;
124         ucVar[2]++;
125         ucVar[3]++;
126         ucVar[4]++;
127
128         if( xQueue != 0 )
129         {
130             //put ucVar[0]'s data to xQueue, wait at most 10
131             xQueueSend( xQueue, ( void * ) &ucVar[0], ( portTickType ) 10 );
132             xQueueSend( xQueue, ( void * ) &ucVar[1], ( portTickType ) 10 );
133             xQueueSend( xQueue, ( void * ) &ucVar[2], ( portTickType ) 10 );
134             xQueueSend( xQueue, ( void * ) &ucVar[3], ( portTickType ) 10 );
135             xQueueSend( xQueue, ( void * ) &ucVar[4], ( portTickType ) 10 );
136
137             vTaskDelayUntil( &xLastWakeTime, xFrequency );
138         }
139     }
140 }

```

Mutex: xSemaphoreHandle & xSemaphoreCreateMutex() & xSemaphoreTake() & xSemaphoreGive()

1. `xSemaphoreHandle xSemaphore;`
2. `xSemaphore = xSemaphoreCreateMutex();`

3. sample code:

- a. `printf` is mutual exclusive
- b. vTask3 跟 vTask4 會輪流取得資源
- c. 可取得 `xSemaphoreTake()` 回傳值，來確認能不能取得資源

```
87 void vTask3( void *pvParameters )
88 {
89     portTickType xLastWakeTime;
90     const portTickType xFrequency = 1000;
91
92     // Initialise the xLastWakeTime variable with the current time.
93     xLastWakeTime = xTaskGetTickCount();
94
95     while(1)
96     {
97         // from xSemaphore take data for at most portMAX_DELAY
98         xSemaphoreTake( xSemaphore, portMAX_DELAY);
99         //===== Critical Section Start =====
100        //this printf can be seen as a shared resource
101        printf("Task3 is running\r\n");
102        //===== Critical Section End =====
103        // return resource
104        xSemaphoreGive( xSemaphore );
105        vTaskDelayUntil( &xLastWakeTime, xFrequency );
106    }
107 }

109 void vTask4( void *pvParameters )
110 {
111     portTickType xLastWakeTime;
112     const portTickType xFrequency = 1000;
113
114     // Initialise the xLastWakeTime variable with the current time.
115     xLastWakeTime = xTaskGetTickCount();
116
117     while(1)
118     {
119         // from xSemaphore take data for at most portMAX_DELAY
120         // finite waiting cus the tasks are arranged in waiting Queue
121         xSemaphoreTake( xSemaphore, portMAX_DELAY);
122         //===== Critical Section Start =====
123        //this printf can be seen as a shared resource
124        printf("Task4 is running\r\n");
125        //===== Critical Section End =====
126        // return resource
127        xSemaphoreGive( xSemaphore );
128        vTaskDelayUntil( &xLastWakeTime, xFrequency );
129    }
130 }
```

Binary Semaphore: xSemaphoreHandle & vSemaphoreCreateBinary() & xSemaphoreTake() & xSemaphoreGive()

1. `xSemaphoreHandle xSemaphore;`
2. `vSemaphoreCreateBinary(xSemaphore);`

3. sample code:

- a. vTask3 要等 vTask4 release `xSemaphore` 才可再存取 ⇒ vTask3 要等 vTask4 執行完才可再執行


```

86 void vTask3( void *pvParameters )
87 {
88     uint8_t Count = 100;
89
90     while(1)
91     {
92         xSemaphoreTake( xSemaphore, portMAX_DELAY); // Take resource
93         Count++;
94         printf("Count = %d\r\n", Count);
95     }
96 }

98 void vTask4( void *pvParameters )
99 {
100     portTickType xLastWakeTime;
101     const portTickType xFrequency = 1000;
102
103     // Initialise the xLastWakeTime variable with the current time.
104     xLastWakeTime = xTaskGetTickCount();
105
106     while(1)
107     {
108         printf("Task4 is running\r\n");
109         xSemaphoreGive( xSemaphore ); // Release resource
110         vTaskDelayUntil( &xLastWakeTime, xFrequency );
111     }
112 }

```

Counting Semaphore: xSemaphoreHandle & xSemaphoreCreateCounting() & xSemaphoreTake() & xSemaphoreGive()

1. `xSemaphoreHandle xSemaphore;`
2. `xSemaphore = xSemaphoreCreateCounting(5,2);`
3. first num: maximum, second num: initial
4. delay少的Task可能可以一次搶得多個資源

Development Tools

I. Development Platform

1. Arduino IDE
2. Qt Creator
3. Visual Studio Code

II. Project Package

1. ESP32 Wrover Module (with WiFi and Bluetooth functions)
2. Line Tracking Module (for line tracking)
3. HC-SR04 Ultrasonic Module (for obstacle avoidance)
4. Dot Matrix Module
5. RGB LED (on board module)
6. Buzzer (on board module)

III. Technologies Used

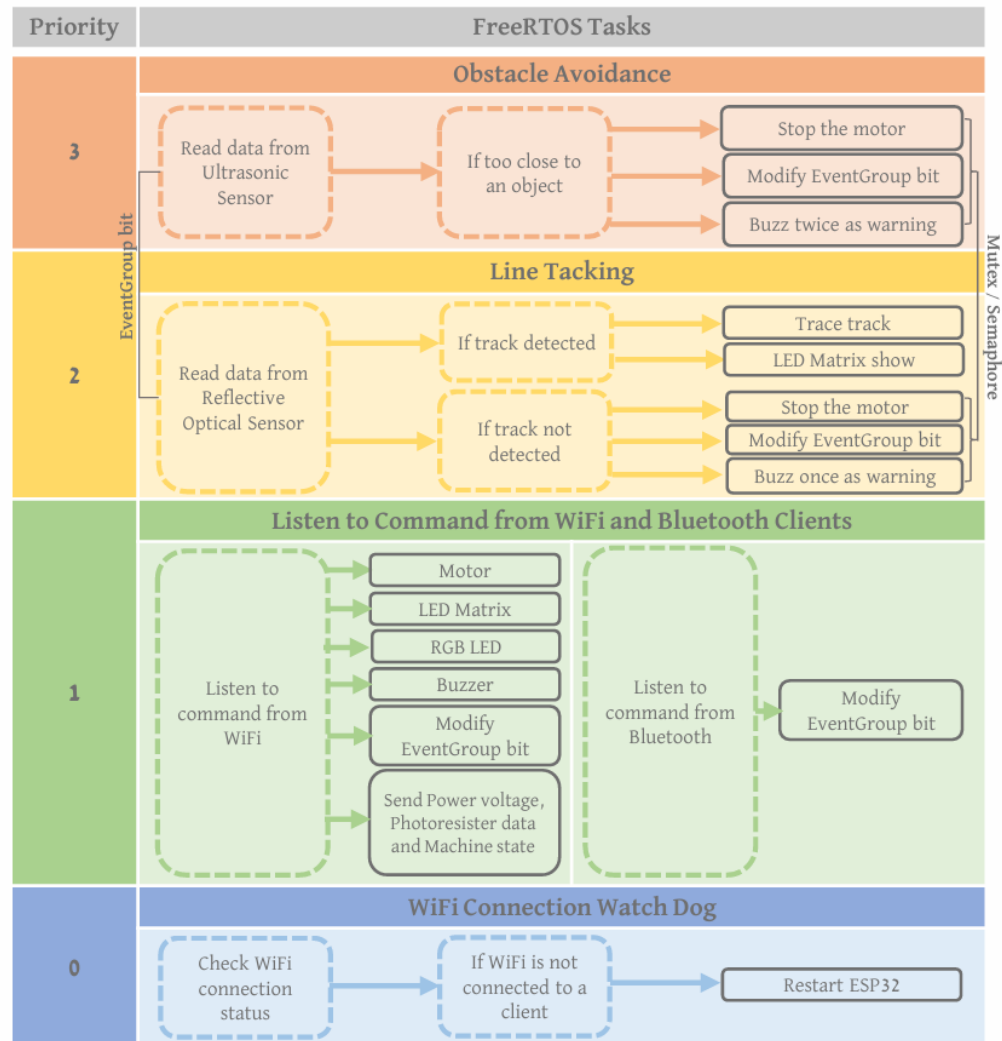
1. MCU (Microcontroller Unit)
2. WiFi
3. Bluetooth
4. FreeRTOS

IV. Programming Language Used

1. C Programming Language
2. Python Programming Language

Project Scenario

The automatic car has three STATE: **IN MOTION : AUTO**, **IN MOTION : TRACK**, **MOTIONLESS**, **OVERRIDE 1** and **OVERRIDE 2**, which are mainly controlled by **Event Group**. In these STATES, the automatic car will perform multiple tasks that have different priorities and are synchronized through **Mutex** and **Event Group**.



Priority Chart

I. IN MOTION : AUTO

In this state, Task 1-4 are all working simultaneously. Obstacle Avoidance and Line Tracking tasks are triggered by setting the first EventGroup bit to 1 after receiving command from Control Panel. Line Tracking task will start after Obstacle Avoidance has been executed one time to avoid the automatic car bumping into an obstacle in sight due to the later execution of Obstacle Avoidance task. The late execution of Line tracking task is controlled by the seventh bit of EventGroup value (Track Begin, TB). When the Obstacle Avoidance task has been executed once, the TB bit will be set to 1. In other word, the Line Tracking task will not be executed until the Obstacle Avoidance task has been executed once.

II. IN MOTION : TRACK

In this state, only Task 2-4 will be working. Obstacle Avoidance will be blocked right after entering the while loop as the first EventGroup bit has been cleared. This state is triggered by setting both the second and the TB bit of EventGroup to 1.

III. MOTIONLESS

The cause of Motionless state can be due to several reasons: 1) Obstacle detected, 2) No track found 3) Bluetooth stop instruction, and 4) WiFi override instruction. The cause of Motionless state will be record with the third to sixth bit of the EventGroup.

IV. OVERRIDE 1

In this state, only RGB LED and LED matrix functionalities will be override, which means only these functionalities can be controlled through control panel. The rest of the functionalities will be controlled through its original state.

V. OVERRIDE 2

In this state, all the functionalities will be override by the control panel. Before entering this state, the automatic car will be forced to a stop, which correspond to the fourth reason of Motionless state: WiFi override instruction.

Main Program

I. ESP32 Program

Module Setup

```
81 void setup() {
82     Buzzer_Setup();           //Buzzer initialization
83     Serial.begin(115200);
84     Serial.setDebugOutput(true);
85     WiFi_Init();             //WiFi parameters initialization
86     WiFi_Setup(0);           //Start AP Mode. If you want to connect to a router, change 1 to 0.
87     server_Cmd.begin(4000);   //Start the command server
88     // server_Camera.begin(7000); //Turn on the camera server
89     Bluetooth_Setup();
90
91     // cameraSetup();         //camera initialization
92     Emotion_Setup();          //Emotion initialization
93     WS2812_Setup();           //WS2812 initialization
94     PCA9685_Setup();          //PCA9685 initialization
95     Light_Setup();            //Light initialization
96     Track_Setup();            //Track initialization
97     Ultrasonic_Setup();       //Initialize the ultrasonic module
98
99     //create event group and assign it a earlier created referene handler
100     xEventGroup = xEventGroupCreate();
101     mutex = xSemaphoreCreateMutex();
102     binary_sem = xSemaphoreCreateBinary();
```

FreeRTOS Tasks Creation

```

105 //xTaskCreateUniversal(loopTask_Camera, "loopTask_Camera", 8192, NULL, 0, NULL, 0);
106 xTaskCreateUniversal(loopTask_WTD, "loopTask_WTD", 8192, NULL, 0, NULL, 0);
107 xTaskCreate( // Use xTaskCreate() in FreeRTOS
108             start_Obst_Avoid, // Function to be called
109             "start_Obst_Avoid", // Name of task
110             2000, // Stack size (bytes in ESP32, words in FreeRTOS)
111             NULL, // Parameter to pass to function
112             3, // Task priority (0 to configMAX_PRIORITIES - 1)
113             &obst_Avoid_Handle // Task handle
114 );
115 xTaskCreate( // Use xTaskCreate() in FreeRTOS
116             start_Line_Track, // Function to be called
117             "start_Line_Track", // Name of task
118             2000, // Stack size (bytes in ESP32, words in FreeRTOS)
119             NULL, // Parameter to pass to function
120             2, // Task priority (0 to configMAX_PRIORITIES - 1)
121             &line_Track_Handle // Task handle
122 );

```

Main Loop: Listen to WiFi command

```

127 void loop() {
128     WiFiClient client = server_Cmd.accept(); //listen for incoming clients
129     if (client) { //if you get a client
130         Serial.println("Cmd_Server connected to a client.");
131         while (client.connected()) { //loop while the client's connected
132             // WiFi Cmd
133             if (client.available()) { //if there's bytes to read from the client
134                 String inputStringTemp = client.readStringUntil('\n'); //Read the command by WiFi
135                 Serial.println(inputStringTemp); //Print out the command received by WiFi
136                 Get_Command(inputStringTemp);
137
138                 if (CmdArray[0] == CMD_LED_MOD) //Set the display mode of car colored lights
139                     WS2812_SetMode(parameters[1]);
140                 if (CmdArray[0] == CMD_LED) //Set the color and brightness of the car lights
141                     WS2812_Set_Color_1(parameters[1], parameters[2], parameters[3], parameters[4]);
142                 if (CmdArray[0] == CMD_MATRIX_MOD) //Set the display mode of the LED matrix
143                     Emotion_SetMode(parameters[1]);
144                 if (CmdArray[0] == CMD_VIDEO) //Video transmission command
145                     videoFlag = parameters[1];
146                 if (CmdArray[0] == CMD_BUZZER) //Buzzer control command
147                     Buzzer_Variable(parameters[1], parameters[2]);
148                 if (CmdArray[0] == CMD_POWER) { //Power and Light query command
149                     float battery_voltage = Get_Battery_Voltage();
150                     float light_value = Get_Photosensitive();
151                     EventBits_t xEventGroupValue = xEventGroupGetBits(xEventGroup);
152                     client.print(CMD_POWER);
153                     client.print(INTERVAL_CHAR);
154                     client.print(battery_voltage);
155                     client.print(INTERVAL_CHAR);
156                     client.print(light_value);

```

```

157             // State update
158             if ((xEventGroupValue & (E1_BIT)) != 0) {
159                 client.print(INTERVAL_CHAR);
160                 client.print(STATE_0);
161             } else if ((xEventGroupValue & (E2_BIT)) != 0) {
162                 client.print(INTERVAL_CHAR);
163                 client.print(STATE_1);
164             } else if ((xEventGroupValue & (E3_BIT)) != 0) {
165                 client.print(INTERVAL_CHAR);
166                 client.print(STATE_OS);
167             } else if ((xEventGroupValue & (E4_BIT)) != 0) {
168                 client.print(INTERVAL_CHAR);
169                 client.print(STATE_TS);
170             } else if ((xEventGroupValue & (E5_BIT)) != 0) {
171                 client.print(INTERVAL_CHAR);
172                 client.print(STATE_BS);
173             } else if ((xEventGroupValue & (E6_BIT)) != 0) {
174                 client.print(INTERVAL_CHAR);
175                 client.print(STATE_RS);
176             } else {
177                 client.print(INTERVAL_CHAR);
178                 client.print(STATE_NONE);
179             }
180             client.print(ENTER);
181         }

```

```

182     if (CmdArray[0] == CMD_MOTOR) { //Network control car movement command
183         Car_SetMode(0);
184         if (parameters[1] == 0 && parameters[3] == 0)
185             Motor_Move(0, 0, 0, 0); //Stop the car
186         else //If the parameters are not equal to 0
187             Motor_Move(parameters[1], parameters[1], parameters[3], parameters[3]);
188     }
189     if (CmdArray[0] == CMD_SERVO) { //Network control servo motor movement command
190         if (parameters[1] == 0)
191             Servo_1_Angle(parameters[2]);
192         else if (parameters[1] == 1)
193             Servo_2_Angle(parameters[2]);
194     }
195     if (CmdArray[0] == CMD_CAMERA) { //Network control servo motor movement command
196         Servo_1_Angle(parameters[1]);
197         Servo_2_Angle(parameters[2]);
198     }

```

```

199     if (CmdArray[0] == CMD_LIGHT) { //Automatic car command
200         if (parameters[1] == 1){
201             Car_SetMode(0); //1
202             xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
203             xEventGroupSetBits(xEventGroup, E1_BIT); // Set E1 bit
204         }else if (parameters[1] == 0){
205             Car_SetMode(0);
206             xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
207             xEventGroupSetBits(xEventGroup, E6_BIT); // Set E6 bit
208         }
209     }
210     else if (CmdArray[0] == CMD_TRACK) { //Tracking car command
211         if (parameters[1] == 1){
212             Car_SetMode(0); //2
213             xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
214             xEventGroupSetBits(xEventGroup, (E2_BIT|TB_BIT)); // Set E2/TB bit
215         }else if (parameters[1] == 0){
216             Car_SetMode(0);
217             xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
218             xEventGroupSetBits(xEventGroup, E6_BIT); // Set E6 bit
219         }
220     }

```

```

224     if (CmdArray[0] == CMD_OVERRIDE) { //Override Mode
225         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
226         xEventGroupSetBits(xEventGroup, E6_BIT); // Set E6 bit
227     }
228     //Clears the command array and parameter array
229     memset(CmdArray, 0, sizeof(CmdArray));
230     memset(parameters, 0, sizeof(parameters));
231 }
232 Emotion_Show(emotion_task_mode); //Led matrix display function
233 WS2812_Show(ws2812_task_mode); //Car color lights display function
234 Car_Select(carFlag); //ESP32 Car mode selection function

```

Main Loop: Listen to Bluetooth command

```

236 // Bluetooth Cmd
237 if (SerialBT.available()){
238     message=SerialBT.read();
239     Serial.write(message);
240
241     if(message==turnON){
242         WS2812_Show(4); //Car color lights display function
243         Serial.println(F(" :LED ON"));
244         SerialBT.println(F("LED ON"));
245     }
246     else if(message==turnOFF){
247         WS2812_Show(0); //Turn LED Off
248         Serial.println(F(" :LED OFF"));
249         SerialBT.println(F("LED OFF"));
250     }
251     else if(message==restart){
252         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
253         xEventGroupSetBits(xEventGroup, E1_BIT); // Set E1 bit
254         Serial.println(F(" :RESTART"));
255         SerialBT.println(F("RESTART"));
256     }
257     else if(message==stop){
258         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
259         xEventGroupSetBits(xEventGroup, E5_BIT); // Set E5 bit
260         Serial.println(F(" :STOP"));
261         SerialBT.println(F("STOP"));
262     }

```

```

263     else{
264         WS2812_Show(5);
265         Serial.println(F(" :Invalid Input"));
266         SerialBT.println(F("Invalid Input"));
267     }
268 }
269 }

```

Main Loop: Restart EPS32 if WiFi Client Not Connected

```

270     client.stop();//close the connection:
271     Serial.println("Command client Disconnected.");
272     ESP.restart();
273 }
274 }

```

Obstacle Avoidance Task

```

336 void start_Obst_Avoid(void *pvParameters){
337     Serial.println("start_Obst_Avoid Created");
338     // define a variable which holds the state of events
339     const EventBits_t xBitsToWaitFor = (E1_BIT);
340     EventBits_t xEventGroupValue;
341     while(1){
342         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
343                                             xBitsToWaitFor,
344                                             pdFALSE,
345                                             pdTRUE,
346                                             portMAX_DELAY
347                                             );
348         if((xEventGroupValue & E1_BIT) != 0){
349             Serial.println("Enter start_Obst_Avoid");
350             xEventGroupSetBits(xEventGroup, TB_BIT); // Set TB bit
351             float distance = get_distance();
352             if (distance <= OBSTACLE_DISTANCE){
353                 xSemaphoreTake(mutex, portMAX_DELAY);
354                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
355                 Buzzer_Alert(2, 1);
356                 Emotion_SetMode(0);
357                 Emotion_Show(emotion_task_mode);
358                 Serial.println("STOP");
359                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
360                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
361                 xSemaphoreGive(mutex);
362             }
363             vTaskDelay(450);
364         }
365     }
366 }

```

Line Tracking Task

```

276 void start_Line_Track(void *pvParameters){
277     Serial.println("start_Line_Track Created");
278     // define a variable which holds the state of events
279     const EventBits_t xBitsToWaitFor = (E1_BIT|E2_BIT);
280     EventBits_t xEventGroupValue;
281     while(1){
282         if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
283             xEventGroupValue = xEventGroupWaitBits(xEventGroup,
284                                             xBitsToWaitFor,
285                                             pdFALSE,
286                                             pdFALSE,
287                                             portMAX_DELAY
288                                             );
289             if((xEventGroupValue & (TB_BIT)) != 0){
290                 Serial.println("Enter start_Line_Track");
291                 Track_Read();
292                 switch (sensorValue[3])
293                 {
294                     case 2: //010
295                     case 5: //101
296                         Emotion_SetMode(3);
297                         Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1); //Move Forward
298                         break;
299                     case 0: //000

```

```

300     case 7: //111
301         Emotion_SetMode(6);
302         vTaskDelay(100);
303         Motor_Move(0, 0, 0, 0); //Stop
304         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
305         xEventGroupSetBits(xEventGroup, E4_BIT); // Set E4 bit
306         Buzzer_Alert(1, 1);
307         Emotion_SetMode(0);
308         Emotion_Show(emotion_task_mode);
309         break;
310     case 1: //001
311     case 3: //011
312         Emotion_SetMode(4);
313         Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4); //Turn Left
314         break;
315     case 4: //100
316     case 6: //110
317         Emotion_SetMode(5);
318         Motor_Move(SPEED_LV4, SPEED_LV4, -SPEED_LV3, -SPEED_LV3); //Turn Right
319         break;
320
321     default:
322         break;
323 }
324 Emotion_Show(emotion_task_mode); //Led matrix display function
325 }
326 xSemaphoreGive(mutex);
327 }else{
328     continue;
329 }
330 }
331 }

```

WiFi Connection Watch Dog

```

76 int wtdFlag = 0;
77 void loopTask_WTD(void *pvParameters) {
78     while (1)
79     {
80         if (WiFi_MODE == 0)
81         {
82             if ((WiFi.isConnected() == 0) && wtdFlag == 0)
83             {
84                 delay(100);
85             }
86             else if ((WiFi.isConnected() != 0) && wtdFlag == 0)
87             {
88                 wtdFlag = 1;
89                 delay(100);
90             }
91             else if ((WiFi.isConnected() == 0) && wtdFlag == 1)
92             {
93                 wtdFlag = 0;
94                 ESP.restart();
95             }
96         }
97     }
98 }

```

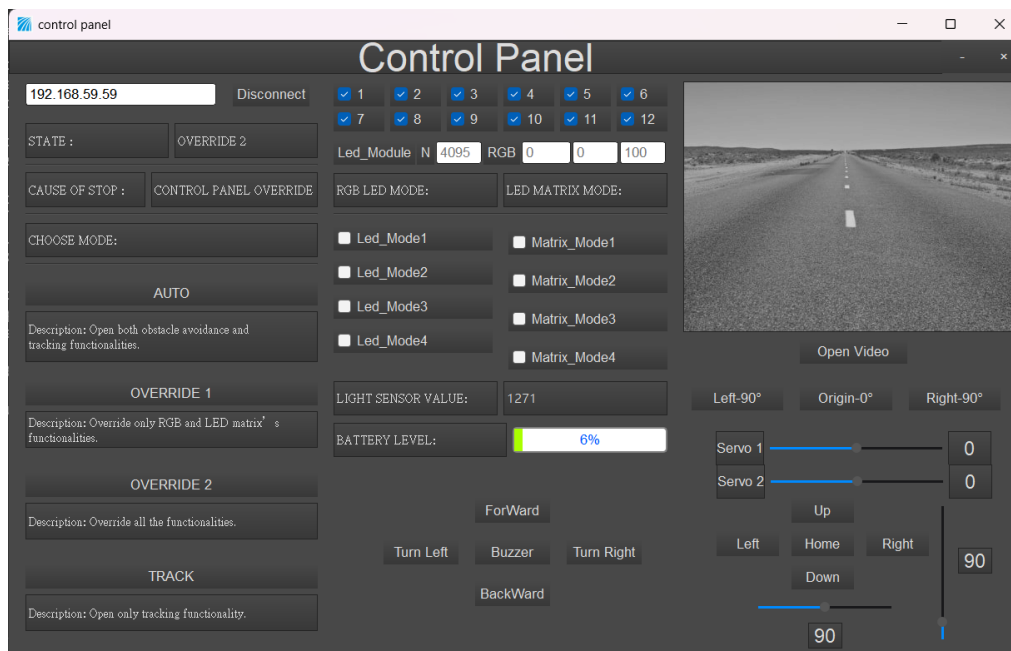
```

97     else
98     {
99         if ((WiFi.softAPgetStationNum() == 0) && wtdFlag == 0)
100         {
101             delay(100);
102         }
103         else if ((WiFi.softAPgetStationNum() != 0) && wtdFlag == 0)
104         {
105             wtdFlag = 1;
106             delay(100);
107         }
108         else if ((WiFi.softAPgetStationNum() == 0) && wtdFlag == 1)
109         {
110             wtdFlag = 0;
111             ESP.restart();
112         }
113     }
114 }
115 }

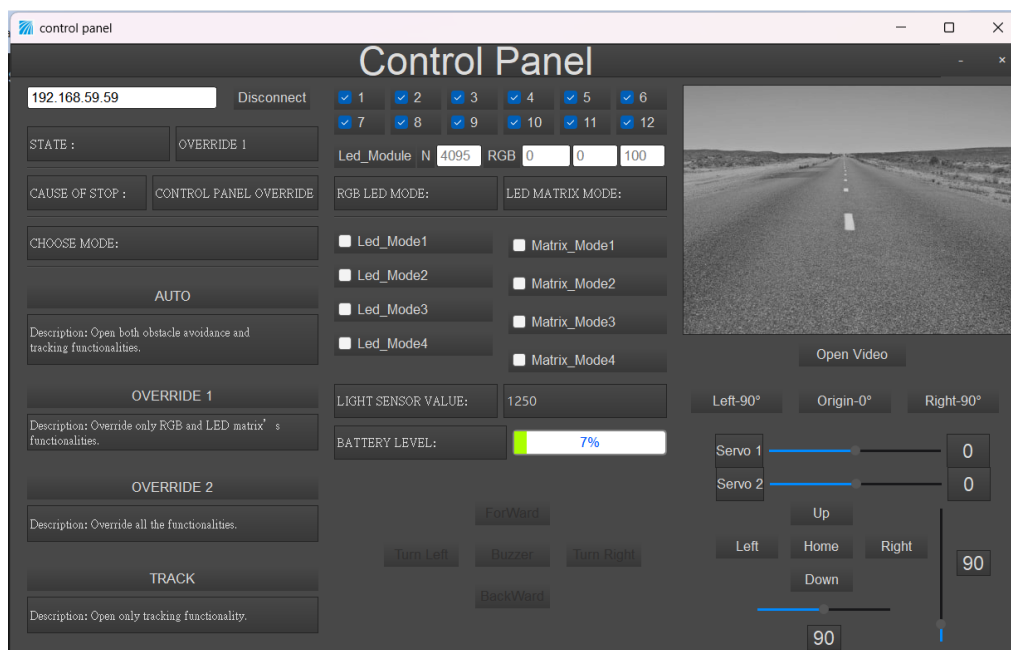
```

II. Contol Panel (run on PC)

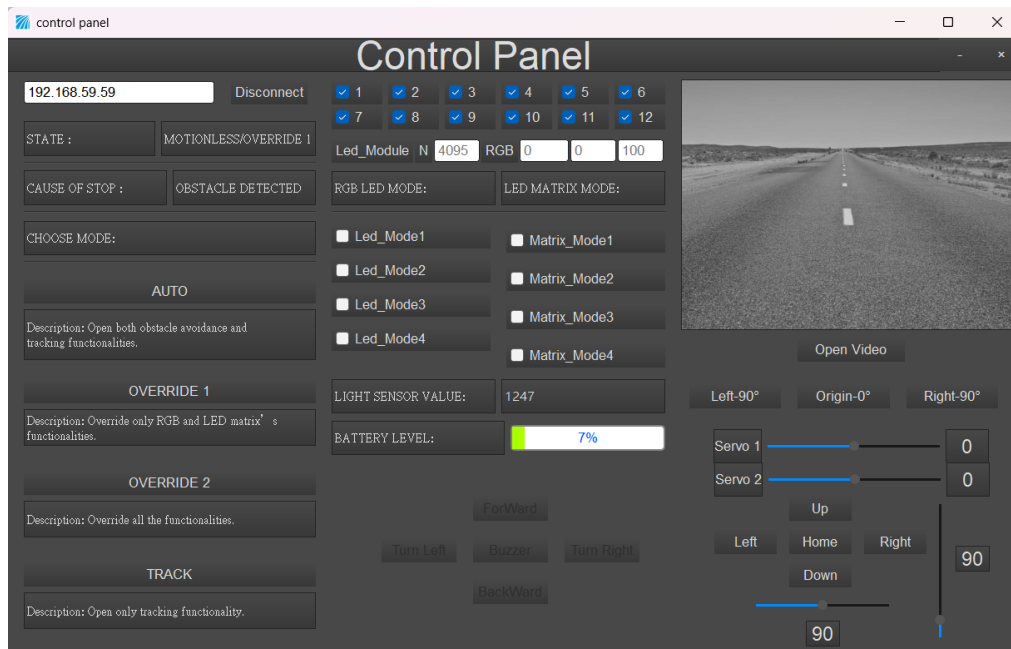
Control panel in OVERRIDE 2 STATE



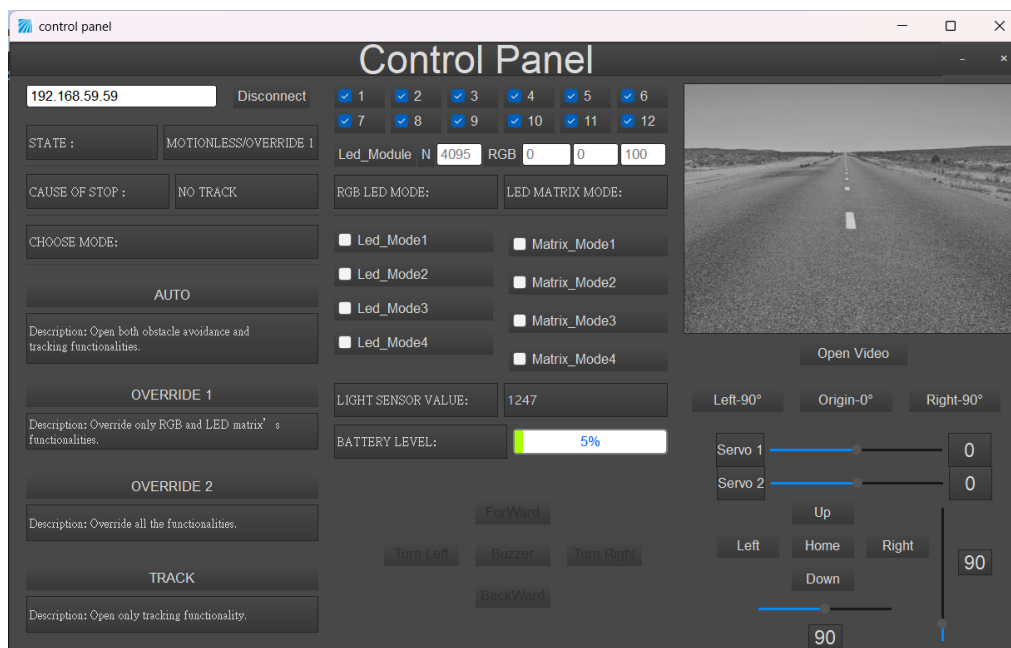
Control panel in OVERRIDE 1 STATE



Control panel in MOTIONLESS/OVERRIDE 1 STATE, Obstacle Detected



Control panel in MOTIONLESS/OVERRIDE 1 STATE, No Track Detected



Problem Discussion

I. Automatic Car Motor Control Race Condition

If we did not protect the automatic car motor with Mutex or Semaphore, **race condition** can occur. A race condition is a problematic situation that arises when a device or system tries to perform multiple operations simultaneously, but the correct outcome depends on a specific order of execution. In programming, race conditions occur when two processes or threads try to access the same resource at the same time, potentially causing system errors or unexpected behavior. In our case, race condition occurs when both Line Tracking task and Obstacle Avoidance task try to control the wheel motor.

Without Mutex or Semaphore

In the following code, because the marked code block is not protected with Mutex or Semaphore, the automatic car might not be able to stop in time when detecting obstacles.

Obstacle Avoidance Task

```
387 void start_Obst_Avoid(void *pvParameters){
388     Serial.println("start_Obst_Avoid Created");
389     // define a variable which holds the state of events
390     const EventBits_t xBitsToWaitFor = (E1_BIT);
391     EventBits_t xEventGroupValue;
392     while(1){
393         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
394         xBitsToWaitFor,
395         pdFALSE,
396         pdTRUE,
397         portMAX_DELAY
398         );
399         if((xEventGroupValue & E1_BIT) != 0){
400             Serial.println("Enter start_Obst_Avoid");
401             xEventGroupSetBits(xEventGroup, TB_BIT); // Set TB bit
402             float distance = get_distance();
403             if (distance <= OBSTACLE_DISTANCE){
404                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
405                 Buzzer_Alert(2, 1);
406                 Emotion_SetMode(0);
407                 Emotion_Show(emotion_task_mode);
408                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
409                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
410                 Serial.println("STOP");
411             }
412             vTaskDelay(450);
413         }
414     }
415 }
```

Line Tracking Task

```
335 void start_Line_Track(void *pvParameters){
336     Serial.println("start_Line_Track Created");
337     // define a variable which holds the state of events
338     const EventBits_t xBitsToWaitFor = (E1_BIT|E2_BIT);
339     EventBits_t xEventGroupValue;
340     while(1){
```

```

341 xEventGroupValue = xEventGroupWaitBits(xEventGroup,
342                                     xBitsToWaitFor,
343                                     pdFALSE,
344                                     pdFALSE,
345                                     portMAX_DELAY
346                                     );
347 if((xEventGroupValue & (TB_BIT)) != 0){
348     Serial.println("Enter start_Line_Track");
349     Track_Read();
350     switch (sensorValue[3])
351     {
352     case 2: //010
353     case 5: //101
354         Emotion_SetMode(3);
355         Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1); //Move Forward
356         break;
357     case 0: //000
358     case 7: //111
359         Emotion_SetMode(6);
360         vTaskDelay(100);
361         Motor_Move(0, 0, 0, 0); //Stop
362         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
363         xEventGroupSetBits(xEventGroup, E4_BIT); // Set E4 bit
364         Buzzer_Alert(1, 1);
365         Emotion_SetMode(0);
366         Emotion_Show(emotion_task_mode);
367         break;
368     case 1: //001
369     case 3: //011
370         Emotion_SetMode(4);
371         Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4); //Turn Left
372         break;
373     case 4: //100
374     case 6: //110
375         Emotion_SetMode(5);
376         Motor_Move(SPEED_LV4, SPEED_LV4, -SPEED_LV3, -SPEED_LV3); //Turn Right
377         break;
378     default:
379         break;
380     }
381 }
382 Emotion_Show(emotion_task_mode); //Led matrix display function
383 }
384 }

```

From the following output, we can discover that when the automatic car detect obstacles, it does not stop directly due to two reasons:

1. The Line Tracking task can still be running concurrently on another core (ESP32 has two cores)
2. Context switch can occur after obstacle is detected, or before clearing up the EventGroup bits.

In other words, by stopping the car and clearing up the EventGroup bits sequentially is not enough to stop the car upon detecting the obstacles.

```

17:42:49.912 -> Enter start_Line_Track
17:42:49.956 -> Enter start_Obst_Avoid
17:42:50.034 -> Enter start_Line_Track
17:42:50.122 -> Enter start_Line_Track
17:42:50.228 -> Enter start_Line_Track
17:42:50.362 -> Enter start_Line_Track
17:42:50.472 -> Enter start_Line_Track
17:42:50.583 -> Enter start_Line_Track
17:42:50.676 -> Enter start_Line_Track
17:42:50.768 -> Enter start_Line_Track
17:42:50.861 -> STOP
17:42:51.172 -> CMD_POWER#
17:42:54.091 -> CMD_POWER#

```

Use vTaskSuspend() and vTaskResume() to Stop Line Tracking Task Temporarily

In the following code, using vTaskSuspend() and vTaskResume() to suspend the line tracking task until the EventGroup bits are cleared can potentially stop the automatic car in time. Yet, problem can still occur when the Line Tracking task is resumed.

```

387 void start_Obst_Avoid(void *pvParameters){
388     Serial.println("start_Obst_Avoid Created");
389     // define a variable which holds the state of events
390     const EventBits_t xBitsToWaitFor = (E1_BIT);
391     EventBits_t xEventGroupValue;
392     while(1){
393         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
394                                                 xBitsToWaitFor,
395                                                 pdFALSE,
396                                                 pdTRUE,
397                                                 portMAX_DELAY
398                                                 );
399         if((xEventGroupValue & E1_BIT) != 0){
400             Serial.println("Enter start_Obst_Avoid");
401             xEventGroupSetBits(xEventGroup, TB_BIT); // Set TB bit
402             float distance = get_distance();
403             if (distance <= OBSTACLE_DISTANCE){
404                 if (line_Track_Handle != NULL){
405                     vTaskSuspend(line_Track_Handle);
406                 }
407                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
408                 Buzzer_Alert(2, 1);
409                 Emotion_SetMode(0);
410                 Emotion_Show(emotion_task_mode);
411                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
412                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
413                 Serial.println("STOP");
414                 vTaskResume(line_Track_Handle);
415             }
416             vTaskDelay(450);
417         }
418     }
419 }

```

From the following output, we can discover that the automatic car stops upon detecting an obstacle and the EventGroup bits are successfully cleared. However, when the Line Tracking task is resumed, The Line Tracking task will not be executed from the beginning of the task where the EventGroup bits will be checked. Instead, the Line Tracking task will start from where it had left before being suspended. Accordingly, the automatic car might move unexpectedly after the Line Tracking task has resumed.

```

17:59:01.037 -> Enter start_Obst_Avoid
17:59:01.068 -> Enter start_Line_Track
17:59:01.206 -> Enter start_Line_Track
17:59:01.299 -> Enter start_Line_Track
17:59:01.378 -> Enter start_Line_Track
17:59:01.501 -> Enter start_Obst_Avoid
17:59:02.390 -> STOP
17:59:04.020 -> CMD_POWER#
17:59:07.013 -> CMD_POWER#

```

With Mutex or Semaphore

In the following code, mutex is used to protect the code block in which wheel motors have been controlled in both the Line Tracking and Obstacle Avoidance tasks.

Line Tracking Task

```

335 void start_Line_Track(void *pvParameters){
336     Serial.println("start_Line_Track Created");
337     // define a variable which holds the state of events
338     const EventBits_t xBitsToWaitFor = (E1_BIT|E2_BIT);
339     EventBits_t xEventGroupValue;
340     while(1){
341         Serial.println("Enter start Line Track");
342         if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
343             xEventGroupValue = xEventGroupWaitBits(xEventGroup,
344                                                     xBitsToWaitFor,
345                                                     pdFALSE,
346                                                     pdFALSE,
347                                                     portMAX_DELAY
348                                                     );
349             if((xEventGroupValue & (TB_BIT)) != 0){
350                 Track_Read();
351                 switch (sensorValue[3])
352                 {
353                     case 2: //010
354                     case 5: //101
355                         Emotion_SetMode(3);
356                         Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1); //Move Forward
357                         break;
358                     case 0: //000
359                     case 7: //111
360                         Emotion_SetMode(6);
361                         vTaskDelay(100);
362                         Motor_Move(0, 0, 0, 0); //Stop
363                         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
364                         xEventGroupSetBits(xEventGroup, E4_BIT); // Set E4 bit
365                         Buzzer_Alert(1, 1);
366                         Emotion_SetMode(0);
367                         Emotion_Show(emotion_task_mode);
368                         break;
369                     case 1: //001
370                     case 3: //011
371                         Emotion_SetMode(4);
372                         Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4); //Turn Left
373                         break;
374                     case 4: //100
375                     case 6: //110
376                         Emotion_SetMode(5);
377                         Motor_Move(SPEED_LV4, SPEED_LV4, -SPEED_LV3, -SPEED_LV3); //Turn Right
378                         break;
379                     default:
380                         break;
381                 }
382                 Emotion_Show(emotion_task_mode); //Led matrix display function
383             }
384             xSemaphoreGive(mutex);
385         }else{
386             continue;
387         }
388     }
389 }
390 }

```

Obstacle Avoidance Task

```

430 void start_Obst_Avoid(void *pvParameters){
431     Serial.println("start_Obst_Avoid Created");
432     // define a variable which holds the state of events
433     const EventBits_t xBitsToWaitFor = (E1_BIT);
434     EventBits_t xEventGroupValue;
435     while(1){
436         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
437                                             xBitsToWaitFor,
438                                             pdFALSE,
439                                             pdTRUE,
440                                             portMAX_DELAY
441                                             );
442         if((xEventGroupValue & E1_BIT) != 0){
443             Serial.println("Enter start_Obst_Avoid");
444             xEventGroupSetBits(xEventGroup, TB_BIT); // Set TB bit
445             float distance = get_distance();
446             if (distance <= OBSTACLE_DISTANCE){
447                 xSemaphoreTake(mutex, portMAX_DELAY);
448                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
449                 Buzzer_Alert(2, 1);
450                 Emotion_SetMode(0);
451                 Emotion_Show(emotion_task_mode);
452                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
453                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
454                 Serial.println("STOP");
455                 xSemaphoreGive(mutex);
456             }
457             vTaskDelay(450);
458         }
459     }
460 }

```

From the following output, we can discover that before stopping, the Line Tracking task is still running. Yet, when it tries to gain access to the wheel motor, it is blocked outside of the code block as the key has already been taken by the Obstacle Avoidance task. Afterward, when the key is retruned, the Line Tracking Task is allowed to execute the protected code block. However, the EventGroup bits have already been cleared. Thus, the Line Tracking task is still unable to control the wheel motor. As a result, the automatic car can be stopped in time when an obstacle is detected and avoid unexpected movement caused by the Line Tracking task.

```

18:34:25.734 -> Enter start_Obst_Avoid
18:34:25.734 -> Enter start_Line_Track
18:34:25.870 -> Enter start_Line_Track
18:34:25.964 -> Enter start_Line_Track
18:34:26.051 -> Enter start_Line_Track
18:34:26.181 -> Enter start_Line_Track
18:34:26.181 -> Enter start_Obst_Avoid
18:34:26.261 -> Enter start_Line_Track
18:34:27.201 -> STOP

```

II. Late Start of Line Tracking

Another problem we face in this project is the late start problem. When the automatic car enter AUTO STATE, we want to guarantee that the Obstacle Avoidance task starts first and the Line Tracking task starts after the Obstacle Avoidance task has finished its first round. By doing so, we can prevent the automatic car from bumping into an obstacle in sight because the Line Tracking task has already been working while the Obstacle Avoidance task has not. The following approaches are some potential solutions.

Using Binary Semaphore

In FreeRTOS, a binary semaphore is a synchronization mechanism used to manage resource access between tasks or signal task completion. It operates with two states: "taken" (0) or "available" (1). A task can "take" the semaphore if it's available, indicating exclusive access to a shared resource, and other tasks must wait until it's released. Binary semaphores are often used for simple signaling, such as notifying a task that an event has occurred.

By letting the Obstacle Avoidance task to release the resource after finish executing the first round, and letting the Line Tracking task to take the resource in the beginning of the task, the Line Tracking task is forced to wait until the Obstacle Avoidance task releases the resource. The late start of Line Tracking task is then fulfilled.

The following is the code.

Obstacle Avoidance Task

```
426 void start_Obst_Avoid(void *pvParameters){
427     Serial.println("start_Obst_Avoid Created");
428     // define a variable which holds the state of events
429     const EventBits_t xBitsToWaitFor = (E1_BIT);
430     EventBits_t xEventGroupValue;
431     while(1){
432         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
433                                             xBitsToWaitFor,
434                                             pdFALSE,
435                                             pdTRUE,
436                                             portMAX_DELAY
437                                             );
438         if((xEventGroupValue & E1_BIT) != 0){
439             Serial.println("Enter start_Obst_Avoid");
440             float distance = get_distance();
441             if (distance <= OBSTACLE_DISTANCE){
442                 xSemaphoreTake(mutex, portMAX_DELAY);
443                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
444                 Buzzer_Alert(2, 1);
445                 Emotion_SetMode(0);
446                 Emotion_Show(emotion_task_mode);
447                 Serial.println("STOP");
448                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
449                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
450                 xSemaphoreGive(mutex);
451             }
452             xSemaphoreGive(binary_sem, portMAX_DELAY);
453             vTaskDelay(450);
454         }
455     }
456 }
```

Line Tracking Task

```
334 void start_Line_Track(void *pvParameters){
335     Serial.println("start Line Track Created");
336     xSemaphoreTake(binary_sem, portMAX_DELAY);
337     // define a variable which holds the state of events
338     const EventBits_t xBitsToWaitFor = (E1_BIT|E2_BIT);
339     EventBits_t xEventGroupValue;
340     while(1){
341         if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
342             xEventGroupValue = xEventGroupWaitBits(xEventGroup,
343                                                 xBitsToWaitFor,
344                                                 pdFALSE,
345                                                 pdFALSE,
346                                                 portMAX_DELAY
347                                                 );
348             Serial.println("Enter start_Line_Track");
349             Track_Read();
350             switch (sensorValue[3])
351             {
352                 case 2: //010
353                 case 5: //101
354                     Emotion_SetMode(3);
355                     Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1); //Move Forward
356                     break;
357                 case 0: //000
358                 case 7: //111
359                     Emotion_SetMode(6);
360                     vTaskDelay(100);
361                     Motor_Move(0, 0, 0, 0); //Stop
362                     xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
363                     xEventGroupSetBits(xEventGroup, E4_BIT); // Set E4 bit
364                     Buzzer_Alert(1, 1);
365                     Emotion_SetMode(0);
366                     Emotion_Show(emotion_task_mode);
367                     break;
368                 case 1: //001
369                 case 3: //011
370                     Emotion_SetMode(4);
371                     Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4); //Turn Left
372                     break;
373             }
374         }
375     }
376 }
```

```

373     case 4: //100
374     case 6: //110
375         Emotion_SetMode(5);
376         Motor_Move(SPEED_LV4, SPEED_LV4, -SPEED_LV3, -SPEED_LV3); //Turn Right
377         break;
378
379     default:
380         break;
381 }
382 Emotion_Show(emotion_task_mode); //Led matrix display function
383
384 xSemaphoreGive(mutex);
385 }else{
386     continue;
387 }
388 }
389 }

```

However, when using binary semaphore, there are some undesired effects for our scenario. After the first round of execution, the Line Tracking task is still affected by this binary semaphore, which causes the Line Tracking task to wait for the Obstacle Avoidance task releasing its resource from time to time. Yet, after the first round of execution, we want the two tasks to run non-blockingly. Thus, we change the code to the following.

Using Two EventGroup Bits

Override Stop (OS)	Bluetooth Stop (BS)	No Track Stop (TS)	Obstacle Stop (OS)	Line Tracking (T)	Obstacle Avoidance (O)
1	1	1	1	1	1

Here, we use the first two bits of the EventGroup to trigger the execution of the Obstacle avoidance task and the Line Tracking task. When the automatic car enter AUTO STATE, (O) will first be set to 1 while (T) will be set to 1 after the Obstacle Avoidance task had finished the first round of execution. By doing so, the Line Tracking task will be triggered later than the Obstacle Avoidance task.

The following is the code.

Obstacle Avoidance Task

```

427 void start_Obst_Avoid(void *pvParameters){
428     Serial.println("start_Obst_Avoid Created");
429     // define a variable which holds the state of events
430     const EventBits_t xBitsToWaitFor = (E1_BIT);
431     EventBits_t xEventGroupValue;
432     while(1){
433         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
434                                             xBitsToWaitFor,
435                                             pdFALSE,
436                                             pdTRUE,
437                                             portMAX_DELAY
438                                             );
439         if((xEventGroupValue & E1_BIT) != 0){
440             Serial.println("Enter start Obst AVOID");
441             xEventGroupSetBits(xEventGroup, E2_BIT); // Set E2 bit
442             float distance = get_distance();
443             if (distance <= OBSTACLE_DISTANCE){
444                 xSemaphoreTake(mutex, portMAX_DELAY);
445                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
446                 Buzzer_Alert(2, 1);
447                 Emotion_SetMode(0);
448                 Emotion_Show(emotion_task_mode);
449                 Serial.println("STOP");
450                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
451                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
452                 xSemaphoreGive(mutex);
453             }
454             xSemaphoreGive(binary_sem, portMAX_DELAY);
455             vTaskDelay(450);
456         }
457     }
458 }

```

Line Tracking Task


```

333 void start_Line_Track(void *pvParameters){
334     Serial.println("start_Line_Track Created");
335     xSemaphoreTake(binary_sem, portMAX_DELAY);
336     // define a variable which holds the state of events
337     const EventBits_t xBitsToWaitFor = (E2_BIT);
338     EventBits_t xEventGroupValue;
339     while(1){
340         if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
341             xEventGroupValue = xEventGroupWaitBits(xEventGroup,
342                                                     xBitsToWaitFor,
343                                                     pdFALSE,
344                                                     pdFALSE,
345                                                     portMAX_DELAY
346                                                     );
347             if((xEventGroupValue & (E2_BIT)) != 0){
348                 Serial.println("Enter start_Line_Track");
349                 Track_Read();
350                 switch (sensorValue[3])
351                 {
352                     case 2: //010
353                     case 5: //101
354                         Emotion_SetMode(3);
355                         Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1); //Move Forward
356                         break;
357                     case 0: //000
358                     case 7: //111
359                         Emotion_SetMode(6);
360                         vTaskDelay(100);
361                         Motor_Move(0, 0, 0, 0); //Stop
362                         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
363                         xEventGroupSetBits(xEventGroup, E4_BIT); // Set E4 bit
364                         Buzzer_Alert(1, 1);
365                         Emotion_SetMode(0);
366                         Emotion_Show(emotion_task_mode);
367                         break;
368                     case 1: //001
369                     case 3: //011
370                         Emotion_SetMode(4);
371                         Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4); //Turn Left
372                         break;
373                     case 4: //100
374                     case 6: //110
375                         Emotion_SetMode(5);
376                         Motor_Move(SPEED_LV4, SPEED_LV4, -SPEED_LV3, -SPEED_LV3); //Turn Right
377                         break;
378                     default:
379                         break;
380                 }
381                 Emotion_Show(emotion_task_mode); //Led matrix display function
382             }
383             xSemaphoreGive(mutex);
384         }else{
385             continue;
386         }
387     }
388 }
389 }

```

However, when using only two bits of the EventGroup to control the sequence of execution, some undesired side effects will occur. For instance, when the automatic car stops due to no track detected in the Line Tracking task (marked with blue in the above code), both (O) and (T) will be cleared to stop the two tasks from entering the next round. Nonetheless, the Obstacle Avoidance task can still be executing its current round, which include setting (T) to 1. This will cause the Line Tracking task starts working again, which isn't what we want. Therefore, we change the code to the following.

Using Three EventGroup Bits

Track Begin (TB)	Override Stop (OS)	Bluetooth Stop (BS)	No Track Stop (TS)	Obstacle Stop (OS)	Line Tracking (T)	Obstacle Aoidance (O)
1	1	1	1	1	1	1

By adding another Track Begin bit (TB), we can avoid the aforementioned problems to occur. The Line Tracking task will be triggered only when either (T) and (TB) or (O) and (TB) are set to 1. When the automatic car enter

AUTO STATE, (O) will first be set to 1 while (TB) will be set to 1 after the Obstacle Avoidance task had finished the first round of execution.

The following is the code.

Obstacle Avoidance Task

```
335 void start_Obst_Avoid(void *pvParameters){
336     Serial.println("start_Obst_Avoid Created");
337     // define a variable which holds the state of events
338     const EventBits_t xBitsToWaitFor = (E1_BIT);
339     EventBits_t xEventGroupValue;
340     while(1){
341         xEventGroupValue = xEventGroupWaitBits(xEventGroup,
342                                             xBitsToWaitFor,
343                                             pdFALSE,
344                                             pdTRUE,
345                                             portMAX_DELAY
346                                             );
347         if((xEventGroupValue & E1_BIT) != 0){
348             Serial.println("Enter start_Obst_Avoid");
349             xEventGroupSetBits(xEventGroup, TB_BIT); // Set TB bit
350             float distance = get_distance();
351             if (distance <= OBSTACLE_DISTANCE){
352                 xSemaphoreTake(mutex, portMAX_DELAY);
353                 Motor_Move(0, 0, 0, 0); //Stop the car to judge the situation
354                 Buzzer_Alert(2, 1);
355                 Emotion_SetMode(0);
356                 Emotion_Show(emotion_task_mode);
357                 Serial.println("STOP");
358                 xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
359                 xEventGroupSetBits(xEventGroup, E3_BIT); // Set E3 bit
360                 xSemaphoreGive(mutex);
361             }
362             vTaskDelay(450);
363         }
364     }
365 }
```

Line Tracking Task

```
276 void start_line_Track(void *pvParameters){
277     Serial.println("start_Line_Track Created");
278     // define a variable which holds the state of events
279     const EventBits_t xBitsToWaitFor = (E1_BIT|E2_BIT);
280     EventBits_t xEventGroupValue;
281     while(1){
282         if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
283             xEventGroupValue = xEventGroupWaitBits(xEventGroup,
284                                                 xBitsToWaitFor,
285                                                 pdFALSE,
286                                                 pdFALSE,
287                                                 portMAX_DELAY
288                                                 );
289             if((xEventGroupValue & (TB_BIT)) != 0){
290                 Serial.println("Enter start_Line_Track");
291                 Track_Read();
292                 switch (sensorValue[3])
293                 {
294                     case 2: //010
295                     case 5: //101
296                         Emotion_SetMode(3);
297                         Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1); //Move Forward
298                         break;
299                     case 0: //000
300                     case 7: //111
301                         Emotion_SetMode(6);
302                         vTaskDelay(100);
303                         Motor_Move(0, 0, 0, 0); //Stop
304                         xEventGroupClearBits(xEventGroup, ALL_BIT); // Clear all bit
305                         xEventGroupSetBits(xEventGroup, E4_BIT); // Set E4 bit
306                         Buzzer_Alert(1, 1);
307                         Emotion_SetMode(0);
308                         Emotion_Show(emotion_task_mode);
309                         break;
310                 }
311             }
312         }
313     }
314 }
```

```

310         case 1: //001
311         case 3: //011
312             Emotion_SetMode(4);
313             Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4); //Turn Left
314             break;
315         case 4: //100
316         case 6: //110
317             Emotion_SetMode(5);
318             Motor_Move(SPEED_LV4, SPEED_LV4, -SPEED_LV3, -SPEED_LV3); //Turn Right
319             break;
320
321         default:
322             break;
323     }
324     Emotion_Show(emotion_task_mode); //Led matrix display function
325 }
326 xSemaphoreGive(mutex);
327 }else{
328     continue;
329 }
330 }
331 }

```

When the automatic car stops due to no track detected in the Line Tracking task, both (O) and (T) will be cleared to stop the two tasks from entering the next round. Although the Obstacle Avoidance task can still be executing its current round, which include setting (TB) to 1, it is not enough to cause the Line Tracking task to enter the next round as both (O) and (TB) need to be set. As for the TRACK STATE in which only Line Tracking task is working, (T) and (TB) should be set together at once so that the Line Tracking task can be triggered directly. At this point, all the requirements are fulfilled.