FreeRTOS Automatic Car Project

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 recieving WiFi and Bluetooth command concurrently. This project also invloves buiding 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 recieving 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

```c
xReturn = xTaskCreate((TaskFunction_t )AppTaskCreate,  /* 任務入口函數 */
                      (const char*    )"AppTaskCreate",/* 任務名字*/
                      (uint16_t       )512,  /* 任務堆疊大小  */
                      (void*          )NULL,/* 任務入口函數參數 */
                      (UBaseType_t    )1, /* 任務的優先順序*/
                      (TaskHandle_t*  )&AppTaskCreate_Handle);/* 任務控制塊指標 */
/*啟動任務調度 */
if(pdPASS == xReturn)
  vTaskStartScheduler();   /* 啟動任務，開啟調度*/
else
  return -1;
```

3. 建立子任務

```c
//AppTaskCreate_Handle
static TaskHandle_t AppTaskCreate_Handle = NULL;
static TaskHandle_t LED_Task_Handle = NULL;
static TaskHandle_t LED1_Task_Handle = NULL;
static TaskHandle_t LED2_Task_Handle = NULL;
```

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

```c
static void AppTaskCreate(void)
{
  BaseType_t xReturn = pdPASS;/* 定義一個建立資訊返回值，預設為pdPASS */

  taskENTER_CRITICAL();           //進入臨界區

  /* 建立LED_Task任務 */
  xReturn = xTaskCreate((TaskFunction_t )LED_Task, /* 任務入口函數 */
                        (const char*    )"LED_Task",/* 任務名字 */
                        (uint16_t       )512,   /* 任務堆疊大小 */
                        (void*          )NULL,  /* 任務入口函數參數 */
                        (UBaseType_t    )2,     /* 任務的優先順序 */
                        (TaskHandle_t*  )&LED_Task_Handle);/* 任務控制塊指標 */
  if(pdPASS == xReturn)
    printf("LED_Task!\r\n");
  /* 建立LED1_Task_Task任務 */
  xReturn = xTaskCreate((TaskFunction_t )LED1_Task,  /* 任務入口函數 */
                        (const char*    )"LED1_Task",/* 任務名字 */
                        (uint16_t       )512,   /* 任務堆疊大小 */
                        (void*          )NULL,/* 任務入口函數參數 */
                        (UBaseType_t    )3, /* 任務的優先順序 */
                        (TaskHandle_t*  )&LED1_Task_Handle);/* 任務控制塊指標 */
  if(pdPASS == xReturn)
    printf("LED1_Task!\r\n");
  /* 建立LED2_Task_Task任務 */
  xReturn = xTaskCreate((TaskFunction_t )LED2_Task,  /* 任務入口函數 */
                        (const char*    )"LED1_Task",/* 任務名字 */
                        (uint16_t       )512,   /* 任務堆疊大小 */
                        (void*          )NULL,/* 任務入口函數參數 */
                        (UBaseType_t    )3, /* 任務的優先順序 */
                        (TaskHandle_t*  )&LED1_Task_Handle);/* 任務控制塊指標 */
  if(pdPASS == xReturn)
    printf("LED2_Task!\r\n");

  //AppTaskCreate_Handle
  vTaskDelete(AppTaskCreate_Handle); //刪除AppTaskCreate任務

  taskEXIT_CRITICAL();            //退出臨界區
}
```

4. 撰寫各任務內容
   a. sample code

```c
static void LED_Task(void* parameter)
{
  while (1)
  {
    LED0( ON );
    //printf("LED_Task Running,LED1_ON\r\n");
    vTaskDelay(5000);   /* 延時500個tick */

    LED0( OFF );
    //printf("LED_Task Running,LED1_OFF\r\n");
    vTaskDelay(5000);   /* 延時500個tick */
  }
}

static void LED1_Task(void* parameter)
{
  while (1)
  {
    LED1( ON );
    //printf("LED_Task Running,LED1_ON\r\n");
    vTaskDelay(1000);   /* 延時500個tick */

    LED1( OFF );
    //printf("LED_Task Running,LED1_OFF\r\n");
    vTaskDelay(1000);   /* 延時500個tick */
  }
}
```

   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()

```c
// set changable delay time
void vTask1( void *pvParameters )
{
    portTickType xDelay, xNextTime;
    /* As per most tasks, this task is implemented in an infinite loop. */
    while(1)
    {
      LED0_TOGGLE;
      //  printf("Task1 is running \r\n");
      xDelay = xNextTime - xTaskGetTickCount ();
      xNextTime += ( portTickType ) 1000;

      if( xDelay <= ( portTickType ) 1000 )
      {
        vTaskDelay( xDelay );
      }
  }
}
```

   i. `portTickType` 是 `xTaskGetTickCount ()` 回傳值的data type

2. sample code 2: xTaskGetTickCount () & vTaskDelayUntil()

```c
void vTask2( void *pvParameters )
{
    portTickType xLastWakeTime;

    const portTickType xFrequency = 100;

     // Initialise the xLastWakeTime variable with the current time.
     xLastWakeTime = xTaskGetTickCount();

    /* As per most tasks, this task is implemented in an infinite loop. */
    while(1)
    {
        //printf("Task2 is running\r\n");
        LED1_TOGGLE;
        vTaskDelayUntil( &xLastWakeTime, xFrequency );

    }
}
```

   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

```c
void vTask3( void *pvParameters )
{
    while(1)
    {
    vTaskSuspend( NULL );
    LED2_TOGGLE;
    }
}
```

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

```c
static void CPU_Task(void* parameter)
{
  uint8_t CPU_RunInfo[400];   //保存任務執行時間資訊

  while (1)
  {
    memset(CPU_RunInfo,0,400);        //信息緩衝區清零

    vTaskList((char *)&CPU_RunInfo);  //獲取任務執行時間資訊

    printf("---------------------------------------------\r\n");
    printf("TaskName    TaskNameState  Priority   Remaining stack TaskSerial\r\n");
    printf("%s", CPU_RunInfo);
    printf("---------------------------------------------\r\n");

    memset(CPU_RunInfo,0,400);        //信息緩衝區清零

    vTaskGetRunTimeStats((char *)&CPU_RunInfo);

    printf("TaskName       Counter      Usage rate\r\n");
    printf("%s", CPU_RunInfo);
    printf("---------------------------------------------\r\n\n");
    vTaskDelay(5000);   /* 延時500個tick */
  }
}
```

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%
GenQ	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

```c
void vTask3( void *pvParameters )
{
    uint8_t Count[5];

    while(1)
    {
    if( xQueue != 0 )
    {
      //put xQueue's data to &Count[0], wait at most portMAX_DELAY
      xQueueReceive( xQueue, &Count[0], portMAX_DELAY);
      xQueueReceive( xQueue, &Count[1], portMAX_DELAY);
      xQueueReceive( xQueue, &Count[2], portMAX_DELAY);
      xQueueReceive( xQueue, &Count[3], portMAX_DELAY);
      xQueueReceive( xQueue, &Count[4], portMAX_DELAY);
    }

        printf("Count0 =  %d\r\n", Count[0]);
        printf("Count1 =  %d\r\n", Count[1]);
        printf("Count2 =  %d\r\n", Count[2]);
        printf("Count3 =  %d\r\n", Count[3]);
        printf("Count4 =  %d\r\n", Count[4]);
    }
}
```

    b. 塞資料

```c
void vTask4( void *pvParameters )
{
  portTickType xLastWakeTime;
  const portTickType xFrequency = 1000;
  uint8_t ucVar[5] = {0};

   // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount();

    while(1)
    {

    ucVar[0]++;
    ucVar[1]++;
    ucVar[2]++;
    ucVar[3]++;
    ucVar[4]++;

    if( xQueue != 0 )
    {
      //put &ucVar[0]'s data to xQueue, wait at most 10
      xQueueSend( xQueue, ( void * ) &ucVar[0], ( portTickType ) 10 );
      xQueueSend( xQueue, ( void * ) &ucVar[1], ( portTickType ) 10 );
      xQueueSend( xQueue, ( void * ) &ucVar[2], ( portTickType ) 10 );
      xQueueSend( xQueue, ( void * ) &ucVar[3], ( portTickType ) 10 );
      xQueueSend( xQueue, ( void * ) &ucVar[4], ( portTickType ) 10 );
    }
        vTaskDelayUntil( &xLastWakeTime, xFrequency );
    }
}
```

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

```c
void vTask3( void *pvParameters )
{
    portTickType xLastWakeTime;
    const portTickType xFrequency = 1000;

  // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount();

    while(1)
    {
      // from xSemaphore take data for at most portMAX_DELAY
      xSemaphoreTake( xSemaphore, portMAX_DELAY);
      //====== Critical Section Start ======
      //this printf can be seen as a shared resource
      printf("Task3 is running\r\n");
      //====== Critical Section End ========
      // return resource
      xSemaphoreGive( xSemaphore );
      vTaskDelayUntil( &xLastWakeTime, xFrequency );
    }
}
```

```c
void vTask4( void *pvParameters )
{
    portTickType xLastWakeTime;
    const portTickType xFrequency = 1000;

  // Initialise the xLastWakeTime variable with the current time.
    xLastWakeTime = xTaskGetTickCount();

    while(1)
    {
      // from xSemaphore take data for at most portMAX_DELAY
      // finite waiting cus the tasks are arranged in waiting Queue
      xSemaphoreTake( xSemaphore, portMAX_DELAY);
      //====== Critical Section Start ======
      //this printf can be seen as a shared resource
      printf("Task4 is running\r\n");
      //====== Critical Section End ========
      // return resource
      xSemaphoreGive( xSemaphore );
      vTaskDelayUntil( &xLastWakeTime, xFrequency );
    }
}
```

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

```c
void vTask3( void *pvParameters )
{
    uint8_t Count = 100;

    while(1)
    {
        xSemaphoreTake( xSemaphore, portMAX_DELAY); // Take resource
        Count++;
        printf("Count = %d\r\n", Count);
    }
}
```

```c
void vTask4( void *pvParameters )
{
    portTickType xLastWakeTime;
    const portTickType xFrequency = 1000;

   // Initialise the xLastWakeTime variable with the current time.
     xLastWakeTime = xTaskGetTickCount();

    while(1)
    {
      printf("Task4 is running\r\n");
      xSemaphoreGive( xSemaphore ); // Release resource
      vTaskDelayUntil( &xLastWakeTime, xFrequency );
    }
}
```

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

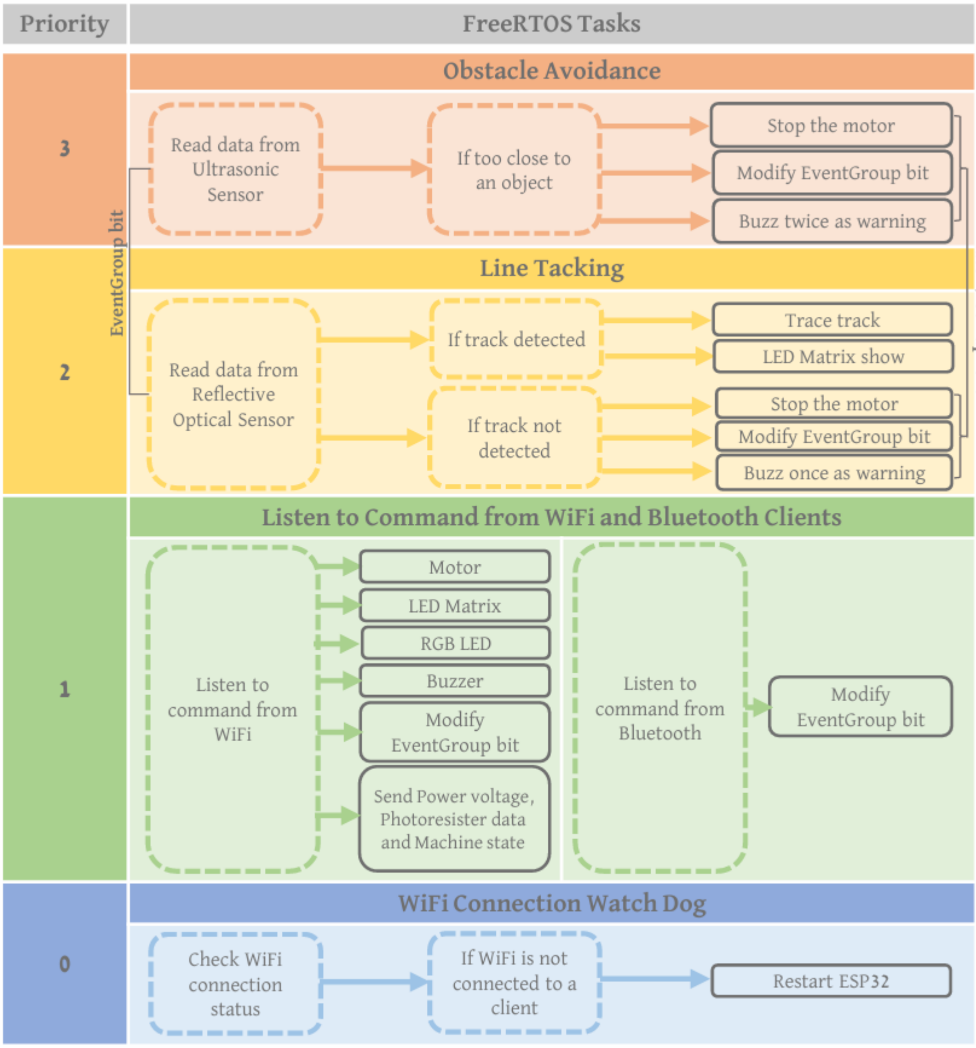
1. MCU (Microcontroller Unit)
2. WiFi
3. Bluetooth
4. FreeRTOS

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 sychronized 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 recieving 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 contolled 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 Progam

I. ESP32 Program

Module Setup

```
void setup() {
  Buzzer_Setup();                 //Buzzer initialization
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  WiFi_Init();                   //WiFi paramters initialization
  WiFi_Setup(0);                 //Start AP Mode. If you want to connect to a router, change 1 to 0.
  server_Cmd.begin(4000);        //Start the command server
  // server_Camera.begin(7000);//Turn on the camera server
  Bluetooth_Setup();

  // cameraSetup();           //Camera initialization
  Emotion_Setup();            //Emotion initialization
  WS2812_Setup();             //WS2812 initialization
  PCA9685_Setup();            //PCA9685 initialization
  Light_Setup();              //Light initialization
  Track_Setup();              //Track initialization
  Ultrasonic_Setup();         //Initialize the ultrasonic module

  //create event group and assign it a earlier created referene handler
  xEventGroup  =  xEventGroupCreate();
  mutex = xSemaphoreCreateMutex();
  binary_sem = xSemaphoreCreateBinary();
```

FreeRTOS Tasks Creation

```
  //xTaskCreateUniversal(loopTask_Camera, "loopTask_Camera", 8192, NULL, 0, NULL, 0);
  xTaskCreateUniversal(loopTask_WTD, "loopTask_WTD", 8192, NULL, 0, NULL, 0);
  xTaskCreate(                // Use xTaskCreate() in  FreeRTOS
              start_Obst_Avoid,   // Function to be called
              "start_Obst_Avoid", // Name of task
              2000,               // Stack size (bytes in ESP32, words in FreeRTOS)
              NULL,               // Parameter to pass to function
              3,                  // Task priority (0 to configMAX_PRIORITIES - 1)
              &obst_Avoid_Handle   // Task handle
  );
  xTaskCreate(                // Use xTaskCreate() in  FreeRTOS
              start_Line_Track,   // Function to be called
              "start_Line_Track", // Name of task
              2000,               // Stack size (bytes in ESP32, words in FreeRTOS)
              NULL,               // Parameter to pass to function
              2,                  // Task priority (0 to configMAX_PRIORITIES - 1)
              &line_Track_Handle               // Task handle
  );
```

Main Loop: Listen to WiFi command

```
void loop() {
  WiFiClient client = server_Cmd.accept();                    //listen for incoming clients
  if (client) {                                               //if you get a client
    Serial.println("Cmd_Server connected to a client.");
    while (client.connected()) {                              //loop while the client's connected
      // WiFi Cmd
      if (client.available()) {                               //if there's bytes to read from the client
        String inputStringTemp = client.readStringUntil('\n');//Read the command by WiFi
        Serial.println(inputStringTemp);                      //Print out the command received by WiFi
        Get_Command(inputStringTemp);

        if (CmdArray[0] == CMD_LED_MOD)//Set the display mode of car colored lights
          WS2812_SetMode(paramters[1]);
        if (CmdArray[0] == CMD_LED) //Set the color and brightness of the car lights
          WS2812_Set_Color_1(paramters[1], paramters[2], paramters[3], paramters[4]);
        if (CmdArray[0] == CMD_MATRIX_MOD)//Set the display mode of the LED matrix
          Emotion_SetMode(paramters[1]);
        if (CmdArray[0] == CMD_VIDEO)//Video transmission command
          videoFlag = paramters[1];
        if (CmdArray[0] == CMD_BUZZER) //Buzzer control command
          Buzzer_Variable(paramters[1], paramters[2]);
        if (CmdArray[0] == CMD_POWER) {//Power and Light query command
          float battery_voltage = Get_Battery_Voltage();
          float light_value = Get_Photosensitive();
          EventBits_t xEventGroupValue = xEventGroupGetBits(xEventGroup);
          client.print(CMD_POWER);
          client.print(INTERVAL_CHAR);
          client.print(battery_voltage);
          client.print(INTERVAL_CHAR);
          client.print(light_value);
```

```
          // State update
          if((xEventGroupValue & (E1_BIT)) != 0){
            client.print(INTERVAL_CHAR);
            client.print(STATE_O);
          }else if((xEventGroupValue & (E2_BIT)) != 0){
            client.print(INTERVAL_CHAR);
            client.print(STATE_T);
          }else if((xEventGroupValue & (E3_BIT)) != 0){
            client.print(INTERVAL_CHAR);
            client.print(STATE_OS);
          }else if((xEventGroupValue & (E4_BIT)) != 0){
            client.print(INTERVAL_CHAR);
            client.print(STATE_TS);
          }else if((xEventGroupValue & (E5_BIT)) != 0){
            client.print(INTERVAL_CHAR);
            client.print(STATE_BS);
          }else if((xEventGroupValue & (E6_BIT)) != 0){
            client.print(INTERVAL_CHAR);
            client.print(STATE_RS);
          }else{
            client.print(INTERVAL_CHAR);
            client.print(STATE_NONE);
          }
          client.print(ENTER);
        }
```

```
    if (CmdArray[0] == CMD_MOTOR) {//Network control car movement command
      Car_SetMode(0);
      if (paramters[1] == 0 && paramters[3] == 0)
        Motor_Move(0, 0, 0, 0);//Stop the car
      else //If the parameters are not equal to 0
        Motor_Move(paramters[1], paramters[1], paramters[3], paramters[3]);
    }
    if (CmdArray[0] == CMD_SERVO) {//Network control servo motor movement command
      if (paramters[1] == 0)
        Servo_1_Angle(paramters[2]);
      else if (paramters[1] == 1)
        Servo_2_Angle(paramters[2]);
    }
    if (CmdArray[0] == CMD_CAMERA) {//Network control servo motor movement command
      Servo_1_Angle(paramters[1]);
      Servo_2_Angle(paramters[2]);
    }
```

```
    if (CmdArray[0] == CMD_LIGHT) { //Automatic car command
      if (paramters[1] == 1){
        Car_SetMode(0);  //1
        xEventGroupClearBits(xEventGroup, ALL_BIT);    // Clear all bit
        xEventGroupSetBits(xEventGroup, E1_BIT);    // Set E1 bit
      }else if (paramters[1] == 0){
        Car_SetMode(0);
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E6_BIT);  // Set E6 bit
      }
    }
    else if (CmdArray[0] == CMD_TRACK) { //Tracking car command
      if (paramters[1] == 1){
        Car_SetMode(0); //2
        xEventGroupClearBits(xEventGroup, ALL_BIT);    // Clear all bit
        xEventGroupSetBits(xEventGroup, (E2_BIT|TB_BIT));    // Set E2/TB bit
      }else if (paramters[1] == 0){
        Car_SetMode(0);
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E6_BIT);  // Set E6 bit
      }
    }
```

```
    if (CmdArray[0] == CMD_OVERRIDE) { //Override Mode
      xEventGroupClearBits(xEventGroup, ALL_BIT);    // Clear all bit
      xEventGroupSetBits(xEventGroup, E6_BIT);  // Set E6 bit
    }
    //Clears the command array and parameter array
    memset(CmdArray, 0, sizeof(CmdArray));
    memset(paramters, 0, sizeof(paramters));
  }
  Emotion_Show(emotion_task_mode);//Led matrix display function
  WS2812_Show(ws2812_task_mode);//Car color lights display function
  Car_Select(carFlag);//ESP32 Car mode selection function
```

Main Loop: Listen to Bluetooth command

```
  // Bluetooth Cmd
  if (SerialBT.available()){
    message=SerialBT.read();
    Serial.write(message);

    if(message==turnON){
      WS2812_Show(4);     //Car color lights display function
      Serial.println(F(" :LED ON"));
      SerialBT.println(F("LED ON"));
    }
    else if(message==turnOFF){
      WS2812_Show(0);        //Turn LED Off
      Serial.println(F(" :LED OFF"));
      SerialBT.println(F("LED OFF"));
    }
    else if(message==restart){
      xEventGroupClearBits(xEventGroup, ALL_BIT);    // Clear all bit
      xEventGroupSetBits(xEventGroup, E1_BIT);    // Set E1 bit
      Serial.println(F(" :RESTART"));
      SerialBT.println(F("RESTART"));
    }
    else if(message==stop){
      xEventGroupClearBits(xEventGroup, ALL_BIT);    // Clear all bit
      xEventGroupSetBits(xEventGroup, E5_BIT);    // Set E5 bit
      Serial.println(F(" :STOP"));
      SerialBT.println(F("STOP"));
    }
```

```
          else{
            WS2812_Show(5);
            Serial.println(F(" :Invalid Input"));
            SerialBT.println(F("Invalid Input"));
          }
        }
      }
```

Main Loop: Restart EPS32 if WiFi Client Not Connected

```
      client.stop();//close the connection:
      Serial.println("Command Client Disconnected.");
      ESP.restart();
    }
  }
```

Obstacle Avoidance Task

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      xEventGroupSetBits(xEventGroup, TB_BIT);  // Set TB bit
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        xSemaphoreTake(mutex, portMAX_DELAY);
        Motor_Move(0, 0, 0, 0);    //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        Serial.println("STOP");
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        xSemaphoreGive(mutex);
      }
      vTaskDelay(450);
    }
  }
}
```

Line Tracking Task

```
void start_Line_Track(void *pvParameters){
  Serial.println("start_Line_Track Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT|E2_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
      xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                              xBitsToWaitFor,
                                              pdFALSE,
                                              pdFALSE,
                                              portMAX_DELAY
                                              );
      if((xEventGroupValue & (TB_BIT)) != 0){
        Serial.println("Enter start_Line_Track");
        Track_Read();
        switch (sensorValue[3])
        {
          case 2:   //010
          case 5:   //101
            Emotion_SetMode(3);
            Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1);    //Move Forward
            break;
          case 0:   //000
```

```
            case 7:   //111
              Emotion_SetMode(6);
              vTaskDelay(100);
              Motor_Move(0, 0, 0, 0);                              //Stop
              xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
              xEventGroupSetBits(xEventGroup, E4_BIT);  // Set E4 bit
              Buzzer_Alert(1, 1);
              Emotion_SetMode(0);
              Emotion_Show(emotion_task_mode);
              break;
            case 1:   //001
            case 3:   //011
              Emotion_SetMode(4);
              Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4);  //Turn Left
              break;
            case 4:   //100
            case 6:   //110
              Emotion_SetMode(5);
              Motor_Move(SPEED_LV4, SPEED_LV4 , - SPEED_LV3, -SPEED_LV3);//Turn Right
              break;

            default:
              break;
          }
          Emotion_Show(emotion_task_mode);//Led matrix display function
        }
        xSemaphoreGive(mutex);
      }else{
        continue;
      }
    }
}
```

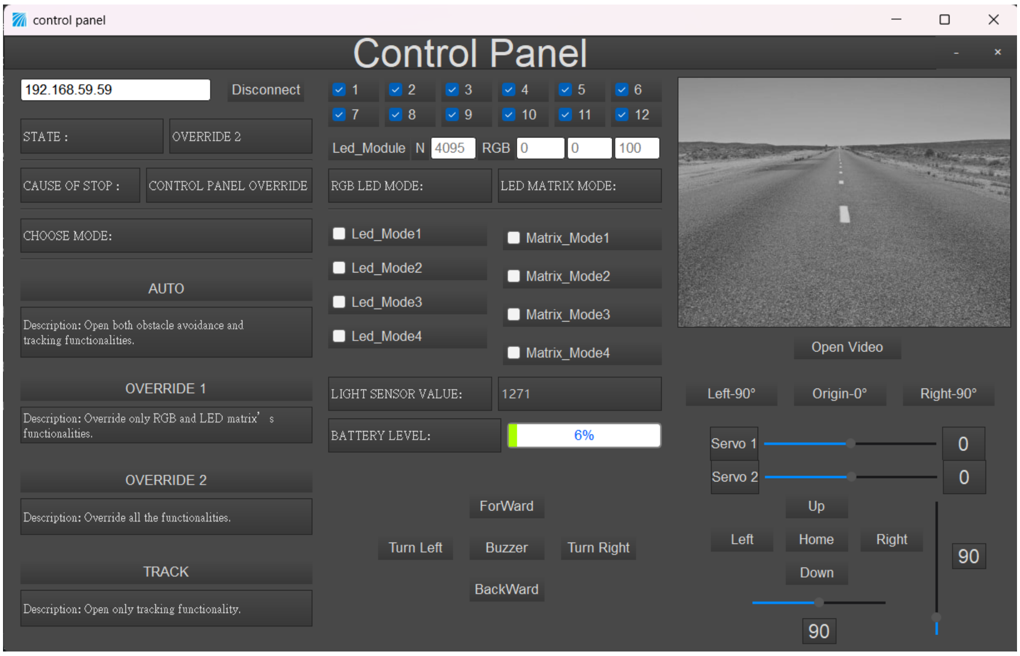
WiFi Connection Watch Dog

```
int wtdFlag = 0;
void loopTask_WTD(void *pvParameters) {
  while (1)
  {
    if (WiFi_MODE == 0)
    {
      if ((WiFi.isConnected() == 0) && wtdFlag == 0)
      {
        delay(100);
      }
      else if ((WiFi.isConnected() != 0) && wtdFlag == 0)
      {
        wtdFlag = 1;
        delay(100);
      }
      else if ((WiFi.isConnected() == 0) && wtdFlag == 1)
      {
        wtdFlag = 0;
        ESP.restart();
      }
    }
```

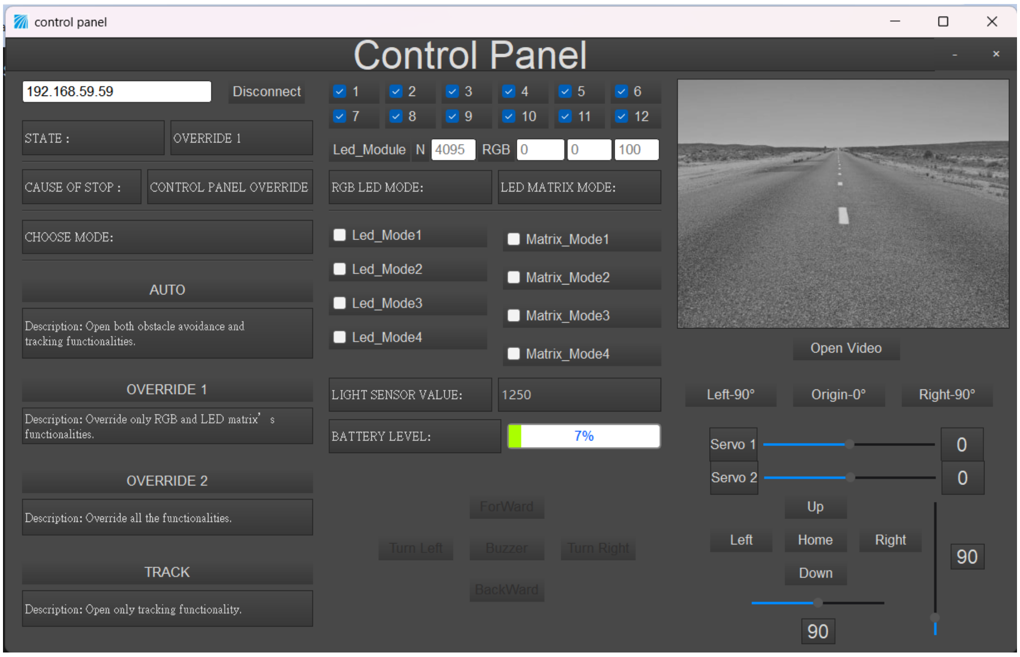
```
    else
    {
      if ((WiFi.softAPgetStationNum() == 0) && wtdFlag == 0)
      {
        delay(100);
      }
      else if ((WiFi.softAPgetStationNum() != 0) && wtdFlag == 0)
      {
        wtdFlag = 1;
        delay(100);
      }
      else if ((WiFi.softAPgetStationNum() == 0) && wtdFlag == 1)
      {
        wtdFlag = 0;
        ESP.restart();
      }
    }
  }
}
```

II. Contol Panel (run on PC)

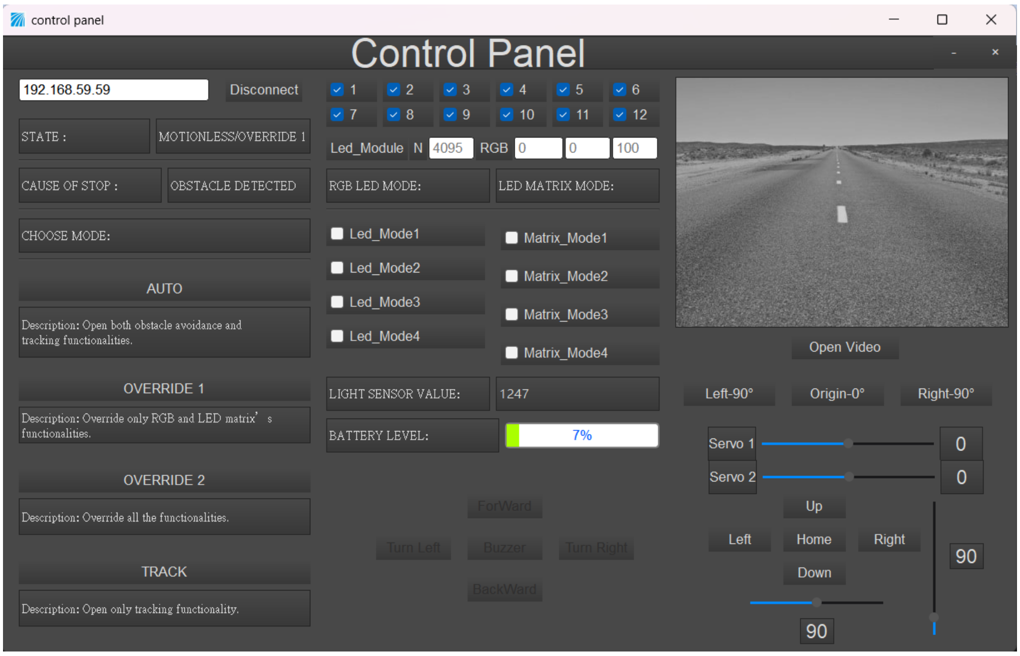
Control panel in OVERRIDE 2 STATE



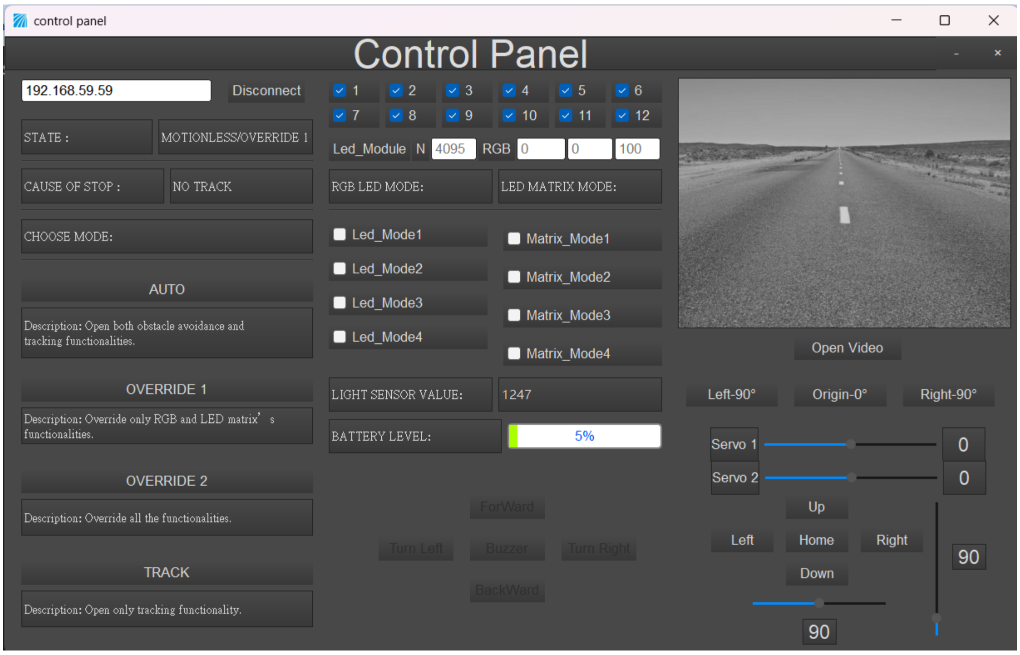
Control panel in OVERRIDE 1 STATE



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. <u>In our case, race condition occurs when both Line Tracking task and Obstacle Avoidance task try to control the wheel motor.</u>

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

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      xEventGroupSetBits(xEventGroup, TB_BIT);  // Set TB bit
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        Motor_Move(0, 0, 0, 0);    //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        Serial.println("STOP");
      }
      vTaskDelay(450);
    }
  }
}
```

Line Tracking Task

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

```
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdFALSE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & (TB_BIT)) != 0){
      Serial.println("Enter start_Line_Track");
      Track_Read();
      switch (sensorValue[3])
      {
        case 2:   //010
        case 5:   //101
          Emotion_SetMode(3);
          Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1);    //Move Forward
          break;
        case 0:   //000
        case 7:   //111
          Emotion_SetMode(6);
          vTaskDelay(100);
          Motor_Move(0, 0, 0, 0);                                    //Stop
          xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
          xEventGroupSetBits(xEventGroup, E4_BIT);  // Set E4 bit
          Buzzer_Alert(1, 1);
          Emotion_SetMode(0);
          Emotion_Show(emotion_task_mode);
          break;
        case 1:   //001
        case 3:   //011
          Emotion_SetMode(4);
          Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4);  //Turn Left
          break;
        case 4:   //100
        case 6:   //110
          Emotion_SetMode(5);
          Motor_Move(SPEED_LV4, SPEED_LV4 , - SPEED_LV3, -SPEED_LV3);//Turn Right
          break;

        default:
          break;
      }
      Emotion_Show(emotion_task_mode);//Led matrix display function
    }
  }
```

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 occure when the Line Tracking task is resumed.

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      xEventGroupSetBits(xEventGroup, TB_BIT);  // Set TB bit
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        if (line_Track_Handle != NULL){
          vTaskSuspend(line_Track_Handle);
        }
        Motor_Move(0, 0, 0, 0);    //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        Serial.println("STOP");
        vTaskResume(line_Track_Handle);
      }
      vTaskDelay(450);
    }
  }
}
```

From the following output, we can discover that the automatic car stops upon detecting an obstacle and the EventGroup bits are succesfully 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

```
void start_Line_Track(void *pvParameters){
  Serial.println("start_Line_Track Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT|E2_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    Serial.println("Enter start Line Track");
    if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
      xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                              xBitsToWaitFor,
                                              pdFALSE,
                                              pdFALSE,
                                              portMAX_DELAY
                                              );
      if((xEventGroupValue & (TB_BIT)) != 0){
        Track_Read();
        switch (sensorValue[3])
        {
          case 2:   //010
          case 5:   //101
            Emotion_SetMode(3);
            Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1);    //Move Forward
            break;
          case 0:   //000
          case 7:   //111
            Emotion_SetMode(6);
            vTaskDelay(100);
            Motor_Move(0, 0, 0, 0);                                    //Stop
            xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
            xEventGroupSetBits(xEventGroup, E4_BIT);  // Set E4 bit
            Buzzer_Alert(1, 1);
            Emotion_SetMode(0);
            Emotion_Show(emotion_task_mode);
            break;
          case 1:   //001
          case 3:   //011
            Emotion_SetMode(4);
            Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4);  //Turn Left
            break;
```

```
          case 4:   //100
          case 6:   //110
            Emotion_SetMode(5);
            Motor_Move(SPEED_LV4, SPEED_LV4 , - SPEED_LV3, -SPEED_LV3);//Turn Right
            break;

          default:
            break;
        }
        Emotion_Show(emotion_task_mode);//Led matrix display function
      }
      xSemaphoreGive(mutex);
    }else{
      continue;
    }
  }
}
```

Obstacle Avoidance Task

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      xEventGroupSetBits(xEventGroup, TB_BIT);  // Set TB bit
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        xSemaphoreTake(mutex, portMAX_DELAY);
        Motor_Move(0, 0, 0, 0);     //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        Serial.println("STOP");
        xSemaphoreGive(mutex);
      }
      vTaskDelay(450);
    }
  }
}
```

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 garantee 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

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        xSemaphoreTake(mutex, portMAX_DELAY);
        Motor_Move(0, 0, 0, 0);     //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        Serial.println("STOP");
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        xSemaphoreGive(mutex);
      }
      xSemaphoreGive(binary_sem, portMAX_DELAY);
      vTaskDelay(450);
    }
  }
}
```

Line Tracking Task

```
void start_Line_Track(void *pvParameters){
  Serial.println("start_Line_Track_Created");
  xSemaphoreTake(binary_sem, portMAX_DELAY);
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT|E2_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
      xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                              xBitsToWaitFor,
                                              pdFALSE,
                                              pdFALSE,
                                              portMAX_DELAY
                                              );
      Serial.println("Enter start_Line_Track");
      Track_Read();
      switch (sensorValue[3])
      {
        case 2:   //010
        case 5:   //101
          Emotion_SetMode(3);
          Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1);    //Move Forward
          break;
        case 0:   //000
        case 7:   //111
          Emotion_SetMode(6);
          vTaskDelay(100);
          Motor_Move(0, 0, 0, 0);                                    //Stop
          xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
          xEventGroupSetBits(xEventGroup, E4_BIT);  // Set E4 bit
          Buzzer_Alert(1, 1);
          Emotion_SetMode(0);
          Emotion_Show(emotion_task_mode);
          break;
        case 1:   //001
        case 3:   //011
          Emotion_SetMode(4);
          Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4);  //Turn Left
          break;
```

```
            case 4:   //100
            case 6:   //110
              Emotion_SetMode(5);
              Motor_Move(SPEED_LV4, SPEED_LV4 , - SPEED_LV3, -SPEED_LV3);//Turn Right
              break;

            default:
              break;
          }
          Emotion_Show(emotion_task_mode);//Led matrix display function

          xSemaphoreGive(mutex);
        }else{
          continue;
        }
      }
    }
```

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 Aoidance (O)
1	1	1	1	1	1

Here, we use the first two bits of the EventGoup 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

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      xEventGroupSetBits(xEventGroup, E2_BIT);  // Set E2 bit
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        xSemaphoreTake(mutex, portMAX_DELAY);
        Motor_Move(0, 0, 0, 0);    //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        Serial.println("STOP");
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        xSemaphoreGive(mutex);
      }
      xSemaphoreGive(binary_sem, portMAX_DELAY);
      vTaskDelay(450);
    }
  }
}
```

Line Tracking Task

```
void start_Line_Track(void *pvParameters){
  Serial.println("start_Line_Track Created");
  xSemaphoreTake(binary_sem, portMAX_DELAY);
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E2_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
      xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                              xBitsToWaitFor,
                                              pdFALSE,
                                              pdFALSE,
                                              portMAX_DELAY
                                              );
      if((xEventGroupValue & (E2_BIT)) != 0){
        Serial.println("Enter start_Line_Track");
        Track_Read();
        switch (sensorValue[3])
        {
          case 2:   //010
          case 5:   //101
            Emotion_SetMode(3);
            Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1);    //Move Forward
            break;
          case 0:   //000
          case 7:   //111
            Emotion_SetMode(6);
            vTaskDelay(100);
            Motor_Move(0, 0, 0, 0);                                   //Stop
            xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
            xEventGroupSetBits(xEventGroup, E4_BIT);  // Set E4 bit
            Buzzer_Alert(1, 1);
            Emotion_SetMode(0);
            Emotion_Show(emotion_task_mode);
            break;
          case 1:   //001
          case 3:   //011
            Emotion_SetMode(4);
            Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4);  //Turn Left
            break;
          case 4:   //100
          case 6:   //110
            Emotion_SetMode(5);
            Motor_Move(SPEED_LV4, SPEED_LV4 , - SPEED_LV3, -SPEED_LV3);//Turn Right
            break;

          default:
            break;
        }
        Emotion_Show(emotion_task_mode);//Led matrix display function
      }
      xSemaphoreGive(mutex);
    }else{
      continue;
    }
  }
}
```

However, when using only two bits of the EventGoup 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

```
void start_Obst_Avoid(void *pvParameters){
  Serial.println("start_Obst_Avoid Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                            xBitsToWaitFor,
                                            pdFALSE,
                                            pdTRUE,
                                            portMAX_DELAY
                                            );
    if((xEventGroupValue & E1_BIT) != 0){
      Serial.println("Enter start_Obst_Avoid");
      xEventGroupSetBits(xEventGroup, TB_BIT);  // Set TB bit
      float distance = get_distance();
      if (distance <= OBSTACLE_DISTANCE){
        xSemaphoreTake(mutex, portMAX_DELAY);
        Motor_Move(0, 0, 0, 0);    //Stop the car to judge the situation
        Buzzer_Alert(2, 1);
        Emotion_SetMode(0);
        Emotion_Show(emotion_task_mode);
        Serial.println("STOP");
        xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
        xEventGroupSetBits(xEventGroup, E3_BIT);  // Set E3 bit
        xSemaphoreGive(mutex);
      }
      vTaskDelay(450);
    }
  }
}
```

Line Tracking Task

```
void start_Line_Track(void *pvParameters){
  Serial.println("start_Line_Track Created");
  // define a variable which holds the state of events
  const EventBits_t xBitsToWaitFor  = (E1_BIT|E2_BIT);
  EventBits_t xEventGroupValue;
  while(1){
    if (xSemaphoreTake(mutex, 1/portTICK_PERIOD_MS) == pdTRUE){
      xEventGroupValue  = xEventGroupWaitBits(xEventGroup,
                                              xBitsToWaitFor,
                                              pdFALSE,
                                              pdFALSE,
                                              portMAX_DELAY
                                              );
      if((xEventGroupValue & (TB_BIT)) != 0){
        Serial.println("Enter start_Line_Track");
        Track_Read();
        switch (sensorValue[3])
        {
          case 2:   //010
          case 5:   //101
            Emotion_SetMode(3);
            Motor_Move(SPEED_LV1, SPEED_LV1, SPEED_LV1, SPEED_LV1);    //Move Forward
            break;
          case 0:   //000
          case 7:   //111
            Emotion_SetMode(6);
            vTaskDelay(100);
            Motor_Move(0, 0, 0, 0);                                     //Stop
            xEventGroupClearBits(xEventGroup, ALL_BIT);  // Clear all bit
            xEventGroupSetBits(xEventGroup, E4_BIT);  // Set E4 bit
            Buzzer_Alert(1, 1);
            Emotion_SetMode(0);
            Emotion_Show(emotion_task_mode);
            break;
```

```
                case 1:   //001
                case 3:   //011
                    Emotion_SetMode(4);
                    Motor_Move(-SPEED_LV3, -SPEED_LV3, SPEED_LV4, SPEED_LV4);  //Turn Left
                    break;
                case 4:   //100
                case 6:   //110
                    Emotion_SetMode(5);
                    Motor_Move(SPEED_LV4, SPEED_LV4 , - SPEED_LV3, -SPEED_LV3);//Turn Right
                    break;

                default:
                    break;
            }
            Emotion_Show(emotion_task_mode);//Led matrix display function
        }
        xSemaphoreGive(mutex);
    }else{
        continue;
    }
  }
}
```

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.