

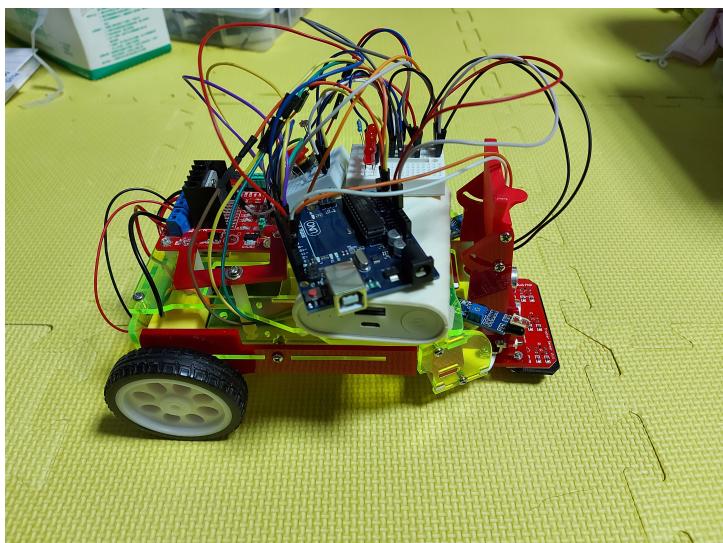


Arduino 跟隨照明自走車 - Report

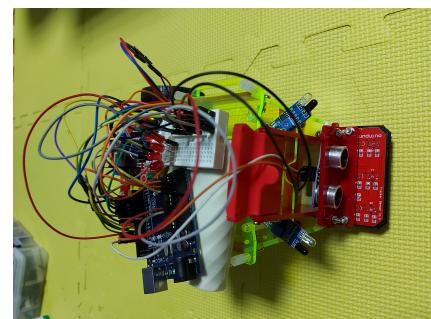
2020 Embedded System Final Project

107062228 陳劭愷

<https://github.com/justin0u0/arduino-auto-follow-car>



最終完成圖（側邊）



最終完成圖（正面）

Goal

Components

L298n 馬達驅動模組

HC-SR04 超聲波模組

紅外線避障模組

LED

Photo Resistor 光敏電阻

Implementation

Wiring

Code

變數宣告

Class Wheel

Class Car

[Class UltraSonic](#)
[Class Led](#)
[Class Infrared](#)
[Task sensorControlTask](#)
[Task lightControlTask](#)
[Task carControlTask](#)
[Interrupt Timer1](#)
[Interrupt wakeUp](#)
[Function initialSearch](#)
[Function setup & loop](#)

[Difficulty](#)

Goal

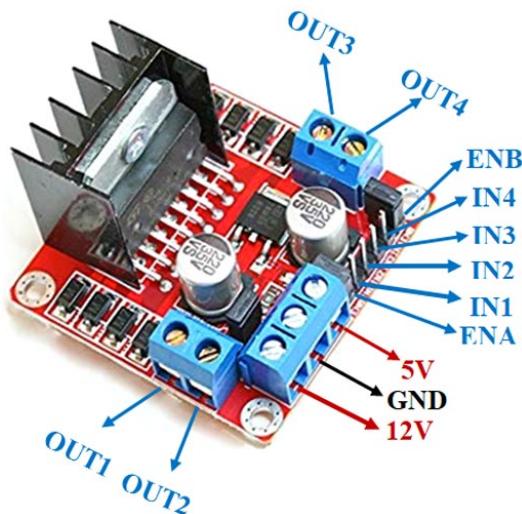
實作一台跟隨照明白走車，功能包括：

1. 當物體直線遠離車體時，自走車需要跟上物體的移動。
2. 當物體過於靠近車體時，自走車需要後退。
3. 當物體移動到車體右前方時，自走車需要右轉。
4. 當物體移動到車體左前方時，自走車需要左轉。
5. 在不同的亮度環境下提供不同強度的光線照明（光線越暗、照明越強）
6. 在剛啟動時，自動旋轉以找尋物體跟隨。
7. 如果超過 10 秒沒有偵測到範圍內有物體，則進入休眠模式。透過 Button Interrupt 的方式將其喚醒。

Components

L298n 馬達驅動模組

L298n 上提供了一些 Pin 來方便的對兩顆馬達 Motor A、Motor B 進行操作。



L298n 示意圖

Image source: <https://components101.com/modules/l293n-motor-driver-module>

下表介紹如何透過 L298n 控制 Motor A、Motor B：

L298n Pin 腳

Aa 名稱	≡ 介紹
<u>IN1, IN2</u>	控制 Motor A 的轉向
<u>ENA</u>	可以利用 PWM 對 Motor A 控制轉速
<u>IN3, IN4</u>	控制 Motor B 的轉向
<u>ENB</u>	可以利用 PWM 對 Motor B 控制轉速
<u>12V</u>	如果電壓大小在 9V ~ 35V 則使用這個 PIN 對 L298n 供電
<u>5V</u>	可以用 Arduino 的 5V 對 L298n 供電
<u>GND</u>	接地
<u>OUT1, OUT2</u>	由 L298n 輸出訊號到 Motor A
<u>OUT3, OUT4</u>	由 L298n 輸出訊號到 Motor B

接下來介紹如何使用 IN1、IN2、ENA 控制 Motor A 的轉速、轉向：

IN1, IN2, ENA 控制 Motor A

Aa No.	► ENA 電位	► IN1 電位	► IN2 電位	≡ Motor A 行為
1	HIGH	HIGH	LOW	逆時針旋轉
2	HIGH	LOW	HIGH	順時針旋轉

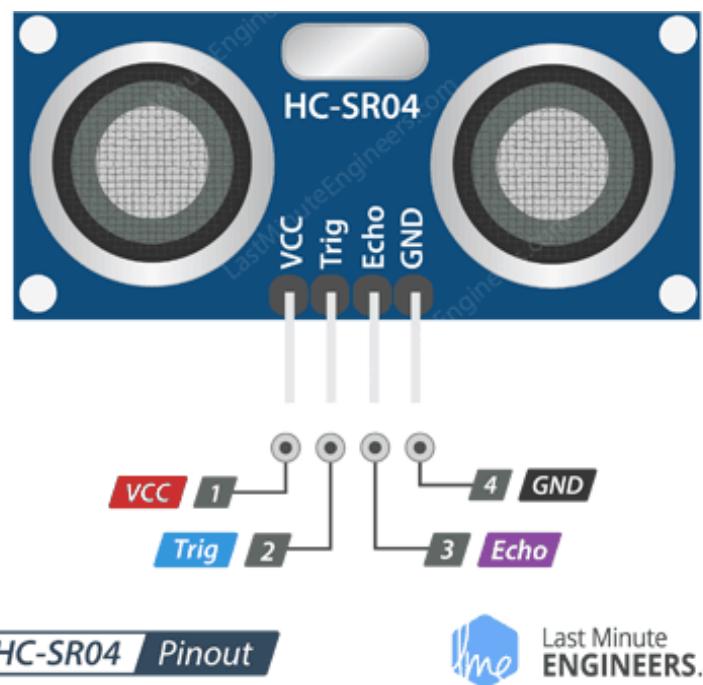
Aa No.	ENa 電位	IN1 電位	IN2 電位	Motor A 行為
3	HIGH	HIGH	HIGH	快速停止
4	HIGH	LOW	LOW	快速停止
5	LOW	Any	Any	馬達自由停止

而事實上，ENA 是可以透過電壓大小來控制 MotorA 轉速，也就是 ENA 的電壓越高，則 Motor A 轉速越高。因此可以透過 PWM 的方式控制電壓大小來控制馬達轉速。

IN3、IN4、ENB 控制 Motor B 的方式與 IN1、IN2、ENA 控制 Motor A 的方式相同，就不再多介紹。

HC-SR04 超聲波模組

超聲波可以幫助偵測前方物體的距離，使用的是 HC-SR04 模組。



超聲波模組示意圖

Image source: <https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/>

HC-SR04 Pin 腳

Aa 名稱	介紹
VCC	5V 輸入

Aa	名稱	☰ 介紹
<u>GND</u>	接地	
<u>Trig</u>	用來觸發量測用。在 Trig Pin 輸出 $\geq 10\text{ms}$ 的高電位電壓後開始量測。	
<u>Echo</u>	用於接收訊號。用收到的高電壓的持續時間就是超聲波從送出到返回的時間。	

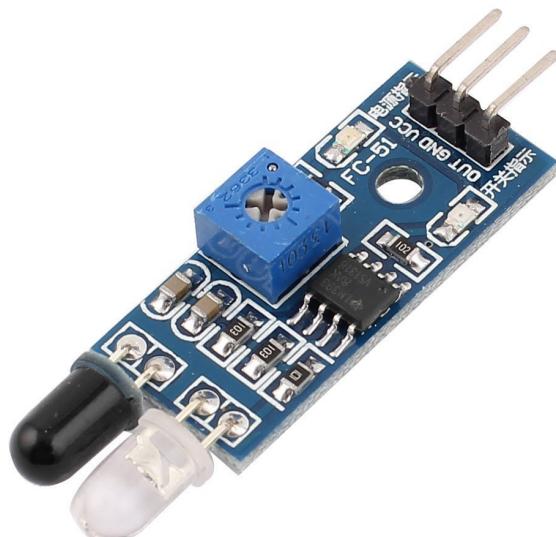
距離計算方式如下：

若設超聲波聲音速度 $340 \text{ m/s} = 0.034 \text{ cm/ms}$, 再乘以超聲波來回時間的一半即可得到距離。

因此距離 時間 $\times 0.5 \times 0.034 \approx \text{時間}/58 (\text{cm})$ 。

紅外線避障模組

在此 Project 中紅外線模組用於感應左前方、右前方有沒有物體存在，來判別應該要往哪一個方向轉向。



紅外線避障模組示意圖

Image source: <https://www.taiwansensor.com.tw/product/紅外線避障模組-避障-近接開關-感應距離可調/>

紅外線避障模組 Pin 腳

Aa	名稱	☰ 介紹
----	----	------

Aa 名稱	☰ 介紹
VCC	5V 輸入
GND	接地
OUT	感應到物體 OUT 為 LOW

紅外線模組上的旋鈕可以調整檢測距離，順時針轉可以增加檢測距離。

LED

在此 Project 中使用 LED 於照明功能。並使用 PWM 來控制 LED 亮度。

Photo Resistor 光敏電阻

在此 Project 中使用光敏電阻來接收照明顯度，以此控制 LED 亮度大小。

Implementation

Wiring

Arduino 與模組的接線

Aa Pin	☰ Pin 腳性質	⌚ 連接模組/設備	☰ 設備 Pin	⌚ 輸入/輸出	☰ 備註
2	Callback Digital	按鈕		輸入	因為要使用到 Interrupt Callback，所以只能挑 Pin 2、Pin 3 使用。
3	Callback Digital PWM	馬達驅動模組	IN2	輸出	
4	Digital	馬達驅動模組	IN1	輸出	
5	Digital PWM	馬達驅動模組	ENA	輸出	有 PWM 功能，所以給 ENA 使用。
6	Digital PWM	馬達驅動模組	ENB	輸出	有 PWM 功能，所以給 ENB 使用。

Aa Pin	≡ Pin 腳性質	◎ 連接模組/設備	≡ 設備 Pin	● 輸 入/輸出	≡ 備註
7	Digital	馬達驅動模組	IN3	輸出	
8	Digital	馬達驅動模組	IN4	輸出	
9	Digital PWM				因為有使用到 Timer1，怕 Pin 9 受到影響，因此不使用此 Pin。
10	Digital PWM				因為有使用到 Timer1，怕 Pin 10 受到影響，因此不使用此 Pin。
11	Digital PWM	LED		輸出	
12	Digital	超聲波	Echo	輸入	
13	Digital	超聲波	Trig	輸出	
A0	Analog	PhotoResistor		輸入	要使用 <code>analogRead</code> 來讀取類比訊號，因此使用 Analog Pin。
A1	Analog	紅外線	OUT	輸入	當作 Digital Pin 來使用。左側紅外線。
A2	Analog	紅外線	OUT	輸入	當作 Digital Pin 來使用。右側紅外線。

L298n 的供電使用 9V 鹼性電池供電，因為若使用 Arduino 的 5V 來供電，雖然 L298n 上的指示燈可以正常運作，但是馬達沒有足夠的力度轉動。

Arduino 板則是使用行動電源供電，因為車體用 USB 連接著電腦就無法四處移動。

Code

這次 Project 採用 OOP 的方式來撰寫 Code，因此由上而下，先介紹各個 Class 的用途，再介紹三個 FreeRTOS 的 Task，最後是一些 ISR、功能函數、和 Arduino 的 Setup、Loop。

變數宣告

```

// Author: justin0u0<mail@justin0u0.com>

#include <Arduino_FreeRTOS.h>
#include <semphr.h>
#include <stdarg.h>

// Class prototypes
class Wheel;
class Car;
class UltraSonic;
class Led;
class Infrared;

// Defines
// Sensor states
#define NOT_FOUND 0
#define TOO_FAR 1
#define TOO_CLOSE 2
#define APPROPRIATE 3
#define TOO_LEFT 4
#define TOO_RIGHT 5
// Car states
#define CAR_FORWARD 0
#define CAR_BACKWARD 1
#define CAR_STOP 2
#define CAR_LEFT 3
#define CAR_RIGHT 4

// Global shared variables
int8_t sensorState;
int8_t timerCounter = -1;
const int8_t wakeUpPin = 2;
TaskHandle_t sensorControlTaskHandle = NULL;
TaskHandle_t carControlTaskHandle = NULL;
TaskHandle_t lightControlTaskHandle = NULL;
bool suspendCarFlag = false;
bool suspendLightFlag = false;

// Locking
SemaphoreHandle_t sensorStateUpdated;

```

在此我們宣告了的 Class 的 Prototype，定義了一些 state、全域共享變數以及 Locking 用的變數。

Class Wheel

```

class Wheel {
private:
    int8_t enablePin;
    int8_t in1Pin;
    int8_t in2Pin;
public:
    Wheel(int8_t enable, int8_t in1, int8_t in2): enablePin(enable),
in1Pin(in1), in2Pin(in2) {
        pinMode(enablePin, OUTPUT);
        pinMode(in1Pin, OUTPUT);
        pinMode(in2Pin, OUTPUT);

        // Default
        this->Stop();
        this->SetSpeed(140);
    }

    // Wheel::RotateFront()
    // Make wheel spin clockwise
    void RotateFront() {
        digitalWrite(in1Pin, LOW);
        digitalWrite(in2Pin, HIGH);
    }

    // Wheel::RotateBack()
    // Make wheel spin anti-clockwise
    void RotateBack() {
        digitalWrite(in1Pin, HIGH);
        digitalWrite(in2Pin, LOW);
    }

    // Wheel::Stop()
    // Stop the wheel spinning immediately
    void Stop() {
        digitalWrite(in1Pin, LOW);
        digitalWrite(in2Pin, LOW);
    }

    // Wheel::SetSpeed(uint8_t)
    // RotateSpeed should set in range [0, 255]
    void SetSpeed(uint8_t rotateSpeed) {
        analogWrite(enablePin, rotateSpeed);
    }
};

```

Wheel 控制 Motor 的轉動方向以及轉速，初始化時我們帶入兩個 IN 和一個 Enable 的 Pin 腳

位，預設 Motor 為停止狀態以及轉速 140，並提供以下函數：

- `Wheel::RotateFront` → 輪子往前轉動。
- `Wheel::RotateBack` → 輪子往後轉動。
- `Wheel::Stop` → 輪子快速停止。
- `Wheel::SetSpeed` → 使用 PWM (`analogWrite`) 控制 Enable Pin 的輸出電壓，因此速度的範圍是在 0 ~ 255 之間，用來控制輪子的轉速。

Class Car

```

class Car {
private:
    Wheel* leftWheel;
    Wheel* rightWheel;
    int8_t carState = CAR_STOP;
public:
    Car(int8_t leftEnable, int8_t leftIn1, int8_t leftIn2, int8_t
rightEnable, int8_t rightIn1, int8_t rightIn2) {
        leftWheel = new Wheel(leftEnable, leftIn1, leftIn2);
        rightWheel = new Wheel(rightEnable, rightIn1, rightIn2);

        this->Stop();
    }

    // Car::Forward()
    // Move car forward
    void Forward() {
        if (carState != CAR_FORWARD) {
            carState = CAR_FORWARD;
            leftWheel->RotateFront();
            rightWheel->RotateFront();
        }
    }

    // Car::Backward()
    // Move car backward
    void Backward() {
        if (carState != CAR_BACKWARD) {
            carState = CAR_BACKWARD;
            leftWheel->RotateBack();
            rightWheel->RotateBack();
        }
    }

    // Car::Left()
    // Turn car left
    void Left() {
        if (carState != CAR_LEFT) {
            carState = CAR_LEFT;
            leftWheel->Stop();
            rightWheel->RotateFront();
        }
    }

    // Car::Right()
    // Turn car right
    void Right() {
        if (carState != CAR_RIGHT) {
            carState = CAR_RIGHT;
            leftWheel->RotateFront();
            rightWheel->Stop();
        }
    }

    // Car::Stop()
    // Stop car moving immediately
    void Stop() {
        if (carState != CAR_STOP) {
            carState = CAR_STOP;
            leftWheel->Stop();
            rightWheel->Stop();
        }
    }

    // Car::SetSpeed()
    // Set car running speed
    // rotateSpeed should in range [0, 255]
    void SetSpeed(uint8_t rotateSpeed) {
        leftWheel->SetSpeed(rotateSpeed);
        rightWheel->SetSpeed(rotateSpeed);
    }
};

```

`Car` 控制車體的運行，因此可以知道一個車體必須有兩個 `wheel`，初始化時，帶入兩個輪子的兩個 IN 以及一個 Enable 的腳位，將兩個 `wheel` 進行初始化。提供以下函數方便車體運行的操作：

- `Car::Forward` → 前進即是兩個輪子都前進。
- `Car::Backward` → 後退即是兩個輪子都後退。
- `Car::Left` → 向左轉即右輪前進左輪不動。
- `Car::Right` → 向右轉及左輪前進右輪不動。
- `Car::Stop` → 停止即兩個輪子快速停止。
- `Car::SetSpeed` → 設定車速即設定兩輪轉速。

對於上述的前 5 個函數，為了避免連續呼叫兩次相同函數而還要再次多呼叫函數，使用一個私有變數 `carState` 來記錄上一次被呼叫的是哪一個函數，並只有在呼叫了與 `carState` 不同的狀態時才執行函數。

Class UltraSonic

```

class UltraSonic {
private:
    int8_t trigPin;
    int8_t echoPin;

public:
    UltraSonic(int8_t trig, int8_t echo): trigPin(trig), echoPin(echo) {
        pinMode(trigPin, OUTPUT);
        pinMode(echoPin, INPUT);
    }

    // UltraSonic::Measure()
    // Measure the distance from ultra sonic
    // Will block CPU for total 12 micro seconds
    float Measure() {
        digitalWrite(trigPin, LOW);
        delayMicroseconds(2);
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);

        float distance = pulseIn(echoPin, HIGH) / 58.0;
        return distance;
    }

    // UltraSonic::GetSensorState()
    // Measure 10 times, get average distance, then return with new sensorState
    int8_t GetSensorState() {
        float average = 0.0;
        for (int8_t i = 0; i < 10; i++) {
            average += this->Measure();
        }
        average /= 10;

        int8_t state;
        if (average < 12) {
            state = TOO_CLOSE;
        } else if (average < 18) { // average > 12
            state = APPROPRIATE;
        } else if (average < 35) {
            state = TOO_FAR;
        } else {
            state = NOT_FOUND;
        }
        return state;
    }
};

```

`UltraSonic` 控制超聲波的運作，在初始化時傳入 Trig、Echo 的腳位。提供兩個函數方便操作：

- `UltraSonic::Measure` → 進行一次超聲波量測並回傳結果。

在前面有介紹到，要開始量測需要對 Trig Pin 輸出 $\geq 10\text{ms}$ 的連續高電壓，因此先將 `trigPin` 設成 `LOW`，經過 2ms 後即輸出 10ms 的高電壓，利用 `pulseIn` 函數得到 `echoPin` 上連續高電壓的時間，計算出距離(cm)並回傳。

- `UltraSonic::GetSensorState` → 多次計算距離後取平均並回傳四種結果：太近 `TOO_CLOSE`、適中 `APPROPRIATE`、太遠 `TOO_FAR` 以及找不到物體 `NOT_FOUND`。

因為超聲波的量測會有誤差，因此我採用量測 10 次取平均值的方式，平均後再回傳結果。

Class Led

```
class Led {
private:
    int8_t pin;
public:
    Led(int8_t pin): pin(pin) {
        pinMode(pin, OUTPUT);

        // Default
        this->SetBrightness(0);
    }

    // Led::SetBrightness()
    // Set brightness of LED in range [0, 255]
    void SetBrightness(uint8_t brightness) {
        analogWrite(pin, brightness);
    }
};
```

`Led` 控制 LED 亮度，在初始化時傳入 LED 腳位。提供一個函數方便操作：

- `Led::SetBrightness` → 設定 LED 亮度，使用的是 `analogWrite` 來做到 PWM，因此亮度的範圍在 $0 \sim 255$ 之間。

Class Infrared

```
class Infrared {
private:
    int8_t pin;
public:
    Infrared(int8_t pin): pin(pin) {
        pinMode(pin, INPUT);
    }

    // Infrared::Detect
    bool Detect() {
        return (digitalRead(pin) == LOW) ? true : false;
    }
};
```

Infrared 控制紅外線，在初始化時傳入紅外線腳位。提供一個函數方便操作：

- **Infrared::Detect** → 偵測紅外線是否有偵測到物體。

Task sensorControlTask

```

// sensorControlTask
// Ultra sonic, infrared control task
// signal carControlTask if sensorState changed
void sensorControlTask(void* pvParameters) {
    // Setup
    UltraSonic* ultraSonic = new UltraSonic(13, 12);
    Infrared* leftInfrared = new Infrared(A2);
    Infrared* rightInfrared = new Infrared(A1);
    int8_t newState = -1;

    // Loop
    for(;;) {
        if (suspendCarFlag) {
            xSemaphoreGive(sensorStateUpdated);
        }

        // Detect by ultra sonic
        newState = ultraSonic->GetSensorState();

        // Not found by ultra sonic, then detect by infrareds
        if (newState == NOT_FOUND) {
            if (leftInfrared->Detect()) {
                newState = TOO_LEFT;
            } else if (rightInfrared->Detect()) {
                newState = TOO_RIGHT;
            }
        }

        // Signal carControlTask
        if (sensorState != newState) {
            sensorState = newState;
            xSemaphoreGive(sensorStateUpdated);
        }
    }
}

```

在 `sensorControlTask` 中，首先偵測超聲波前是否有物體，使用 `ultraSonic::GetSensorState` 來取得新的 state。若超聲波並沒有找到任何物體，則物體可能是移動到車體的左前或是右前方，因此對兩個紅外線都進行偵測，如果有偵測到的話則知道新的狀態可能是物體在左前側 `TOO_LEFT` 或是在右前側 `TOO_RIGHT`。最後取得的新狀態 `newState` 若與上一個狀態 `sensorState` 不同，則告知讓 `carControlTask` 運行。

因為 `carControlTask` 中會等待 `sensorStateUpdated` 這個 binary semaphore 有資源時才執行，因此決定何時讓 `carControlTask` 運行是 `sensorControlTask` 的重要工作。可

以看到只有兩個情況下要讓 `carControlTask` 運行，一是當 `sensorState` 被改變，因此車子的行為也應該要改變；二是在 `suspendCarFlag` 為 `true` 時，這個 flag 是用來告知要進入休眠模式的，詳細會在後面介紹。

Task lightControlTask

```
// lightControlTask
// Detect photo resistor, control LED lightness
void lightControlTask(void* pvParameters) {
    // Setup
    Led* led = new Led(11);
    int8_t value = 0;

    // Loop
    for (;;) {
        if (suspendLightFlag) {
            led->SetBrightness(0);
            suspendLightFlag = false;
            vTaskSuspend(NULL);
        }

        value = analogRead(A0);
        if (value >= 100) {
            led->SetBrightness(0);
        } else if (value >= 60) {
            led->SetBrightness(75);
        } else if (value >= 30) {
            led->SetBrightness(150);
        } else {
            led->SetBrightness(255);
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

在 `lightControlTask` 中，要偵測光敏電阻的亮度，並且設定 LED 的亮度。

`suspendLightFlag` 為要進入休眠模式時會被設成 `true`，當進入休眠模式時，將 LED 全數關閉並將自己這個 task 暫時 suspend。

Task carControlTask

```

// carControlTask
// Control car movement by sensorState
// wait until sensorState updated to execute
void carControlTask(void* pvParameters) {
    // Setup
    Car* car = new Car(6, 7, 8, 5, 4, 3);

    // Loop
    for (;;) {
        // Wait until sensorState got updated
        xSemaphoreTake(sensorStateUpdated, portMAX_DELAY);

        if (suspendCarFlag) {
            car->Stop();
            suspendCarFlag = false;
            vTaskSuspend(sensorControlTaskHandle);
            vTaskSuspend(NULL);
        }
        timerCounter = -1;

        switch (sensorState) {
            case T00_CLOSE:
                car->Backward();
                break;
            case T00_FAR:
                car->Forward();
                break;
            case APPROPRIATE:
                car->Stop();
                break;
            case T00_LEFT:
                car->Left();
                break;
            case T00_RIGHT:
                car->Right();
                break;
            case NOT_FOUND:
                // Start count 10 seconds, then sleep
                car->Stop();
                TCNT1 = 0; // Reset timer
                timerCounter = 0;
                break;
            default:
                car->Stop();
        }
    }
}

```

在 `carControlTask` 中，上面有提到，必須等待 `sensorControlTask` 告知偵測物體狀態有所改變時，才進行操作，因此使用 `xSemaphoreTake` 來等待 `sensorStateUpdated` 這個 signal。

接著就可以利用全域共享變數 `sensorState` 來得知現在物體與車體的狀態是如何的。根據每一種不同的狀態，即可對車子進行操作。例如：太靠近 `TOO_CLOSE` 的話可以 `Car::Backward` 來後退。這裡就不多贅述。

當物體沒有被偵測超過 10 秒應該要進入休眠模式，因此在 `NOT_FOUND` 時，將 Timer1 的計數器 Reset，並且將 `timerCount` 從 -1 變成 0。`timerCount` 在 -1 時代表不需要計時，所以平時 `timerCount` 應該要是 -1，這部分會在底下再說明。

當 `suspendCarFlag` 被設起，代表應該要進入休眠模式，因此將車子進入停止狀態，同時 suspend `sensorControlTask` 以及自己 `carControlTask`。

Interrupt Timer1

```
// Built-in ISR to count to 10 seconds then suspend all tasks
ISR(TIMER1_COMPA_vect) {
    if (timerCounter >= 0) {
        timerCounter++;
        if (timerCounter == 10) {
            // Sleep
            Serial.println(F("SLEEP"));
            suspendLightFlag = true;
            suspendCarFlag = true;
            timerCounter = -1;
        }
    }
}
```

本次 project 中使用 Timer1 來計時 10 秒沒有偵測到物體後進入休眠模式。因為 Timer1 的 prescaler 最大到 1024 而已，而 Arduino 的時脈為 16M，並且 Timer1 使用的 register 為 16 bits 的：

$$2^{16} \div (1.6 \times 10^6) \approx 4.16(s)$$

可以知道就算用上最大的 prescaler，一次的 Timer1 Overflow 都還是無法計算到 10 秒的，因此使用了 `timerCounter` 來計算 10 次的一秒取代，並且將一次的 Timer1 Interrupt 時間設為 1 秒。

上面有提到當 `timerCounter == -1` 時代表不計時，因此當 `timerCounter ≥ 0` 時將 `timerCounter` 加一，當加到 10 的時候，代表經過了 10 秒了，因此將

`suspendLightFlag` 以及 `suspendCarFlag` 都設成 `true`。

Interrupt wakeUp

```
// ISR to wake up all tasks from suspend
void wakeUp() {
    Serial.println(F("WakeUp"));
    xTaskResumeFromISR(sensorControlTaskHandle);
    xTaskResumeFromISR(carControlTaskHandle);
    xTaskResumeFromISR(lightControlTaskHandle);
}
```

`wakeUp` 函數用來當做按鈕的 Interrupt Service Routine，當進入休眠模式且按鈕被按下時，將被 `suspend` 的 task 都喚醒，因為是在 ISR 內使用，因此要用的函數是 `xTaskResumeFromISR` 而非 `vTaskResume`。

Function initialSearch

```
// initialSearch
// Make car turn right continuously until object detected
void initialSearch() {
    // Setup
    Car* car = new Car(6, 7, 8, 5, 4, 3);
    UltraSonic* ultraSonic = new UltraSonic(13, 12);

    car->Right();

    // Loop
    for (;;) {
        if (ultraSonic->GetSensorState() != NOT_FOUND) {
            Serial.println(F("Object detected! Start following!"));
            break;
        }
    }
}
```

在初始運行時，我希望車子可以旋轉以找尋物體，因此直接讓車子不停向右，並且直到超聲波偵測到物體後直接離開函數。

函數 `initialSearch` 即是用來幫助完成這件事。

Function setup & loop

```
void setup() {
    Serial.begin(9600);

    // Initialize Timer1, timer1 infect pin 9 and pin 10
    noInterrupts();
    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1 = 0; // Actual timer value
    OCR1A = 62500; // 16MHZ / 256 * 1
    TCCR1B |= (1 << WGM12); // Clean Timer on Compare Mode
    TCCR1B |= (1 << CS12); // Prescaler 256
    TIMSK1 |= (1 << OCIE1A); // Enable timer compare interrupt
    interrupts();

    // Setup external wakeup pin
    pinMode(wakeUpPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(wakeUpPin), wakeUp, CHANGE);

    sensorStateUpdated = xSemaphoreCreateBinary();

    // Make turn until object found
    initialSearch();

    xTaskCreate(sensorControlTask, "SensorControl", 64, NULL, 1, &
sensorControlTaskHandle);
    xTaskCreate(carControlTask, "CarControl", 128, NULL, 1, &
carControlTaskHandle);
    xTaskCreate(lightControlTask, "LightControl", 64, NULL, 1, &
lightControlTaskHandle);
}

void loop() {}
```

在 `setup` 中，首先設定 Timer1，設定方法在註解已經註明，就不多加贅述。再來設定按鈕是用來觸發 ISR `wakeUp` 函數用，完成後呼叫 `initialSearch` 函數開始找尋物體。最後物體發現後，建立三個 Tasks `sensorControlTask`、`carControlTask` 以及 `lightControlTask`。

Difficulty

1. 在一開始測試馬達運轉時，明明提供了 9V 的電池，但是馬達卻無法克服最一开始的靜摩擦力開始轉動。到最後將 **9V** 的碳鋅電池換成 **9V** 的鹼性電池 就成功的運轉了。
2. 原本設計的休眠模式是將 Arduino 放到 Low Power Mode，再使用 button interrupt 將其喚醒，但是沒想到 Arduino low-power 的 library 不能跟 FreeRTOS 的 library 一起使用，最後只好改了其他方式來呈現休眠模式。