工科專論<第九組>期末報告

程世鉉 B06505019 張哲浩 B06505027

牛繼緯 B06505039 黃日新 B06505048

2020/06/24

1. 分工表

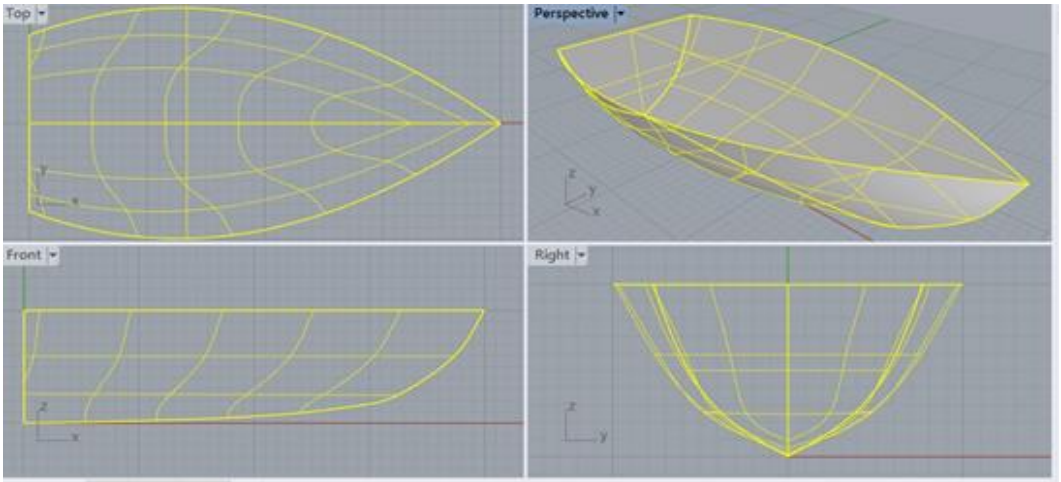
組員 張哲浩：策略擬定、硬體架設與電路板設定、船長、熬夜航向偉大航道、包辦大小事、控制程式討論

組員 黃日新：船體控制程式、電池軍火商、司機、舵手、甩尾、跑跑技術指導、跟屌一樣 like a dick

組員 程世鉉：控制程式修正、策略討論、賣萌、買飲料、腳痛

組員 牛繼緯：船模建置、軍火商、架設武裝

2. 船體設計

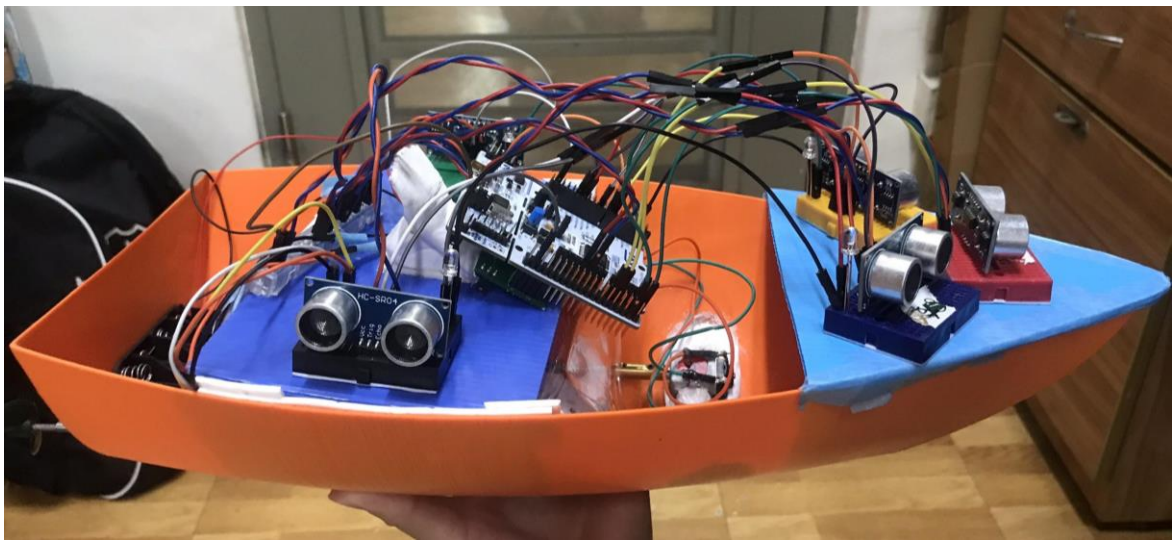


考慮到長度限制、吃水、賽道寬度和旋轉半徑，將船長設為 40cm，船寬設為 20cm，船高 10cm，盡量在符合規定的範圍內取得吃水深度與旋轉半徑兩者間的平衡，讓船不至於太長而造成轉彎不易或因為迴轉半徑過大而增加碰撞，也不會因為太短而需要增加高度來達到目標的排水量；船寬則因為原本賽道迴轉區寬度為 30cm，略寬的船體可以讓超音波感測器更快感測到牆壁而避免碰撞。

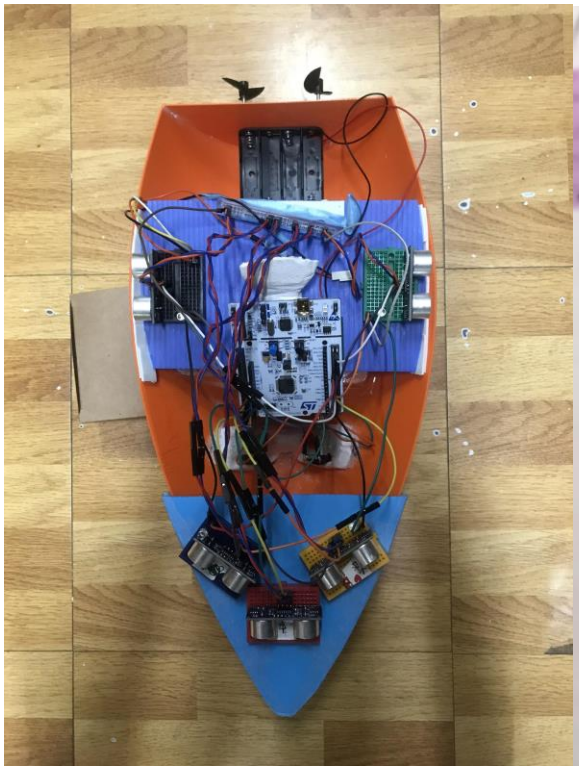
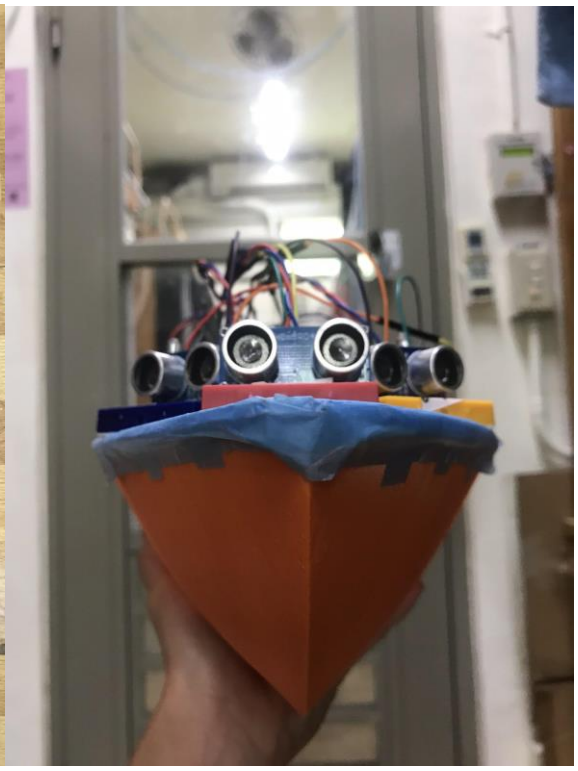
船體設計為了避免翻船或晃動過多導致進水，整體以穩度為優先考量，因此使用倒三角形的船型來設計來增加 GM 值，最後的效果也不錯(1.42 cm)，**同時倒三角形也能有效地增加預備浮力；船底的部分收成直角，可以比圓弧型的船底增加更多吃水面積，船的高度也能控制在一定範圍內，使重心不會偏高；船身的部分盡量把曲率拉得連續，讓整體形狀更加流線以降低水的阻力；船尾選擇切齊而不做尖尾是為了能讓船體在有限的長度內增加吃水面積，讓水線不要太高，同時也能增加船體的實際長度，但在這裡因為速度較低所以影響應該不大。**

螺槳裝在船的正後方以發揮最大動力，高度約在船底到水線的中間，使兩者不會因為距離太近而互相影響，也足夠提供轉彎所需的較大力矩。馬達因為考慮到軸的長度設置在船體的偏前方處，同時也因船體後方較深、前方較淺，因此將主電路板和電池裝設在後方，使整體重心能與空船時保持一致。

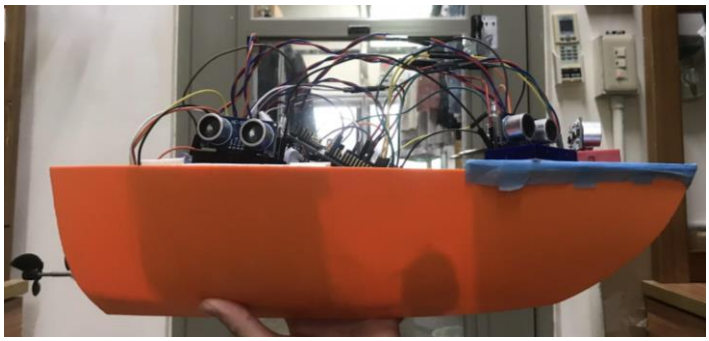
4. 放入你船體的實際圖片



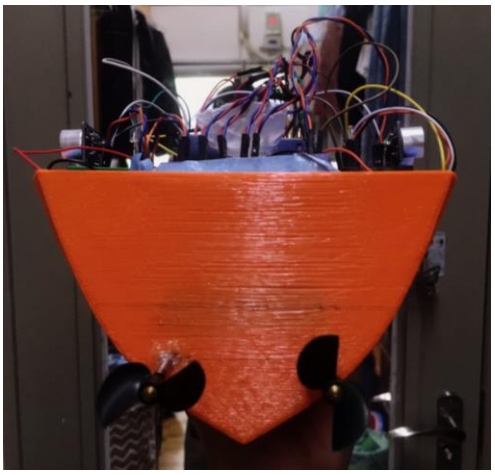
fg. 跟屌一樣號

fg. 俯視圖/正視圖

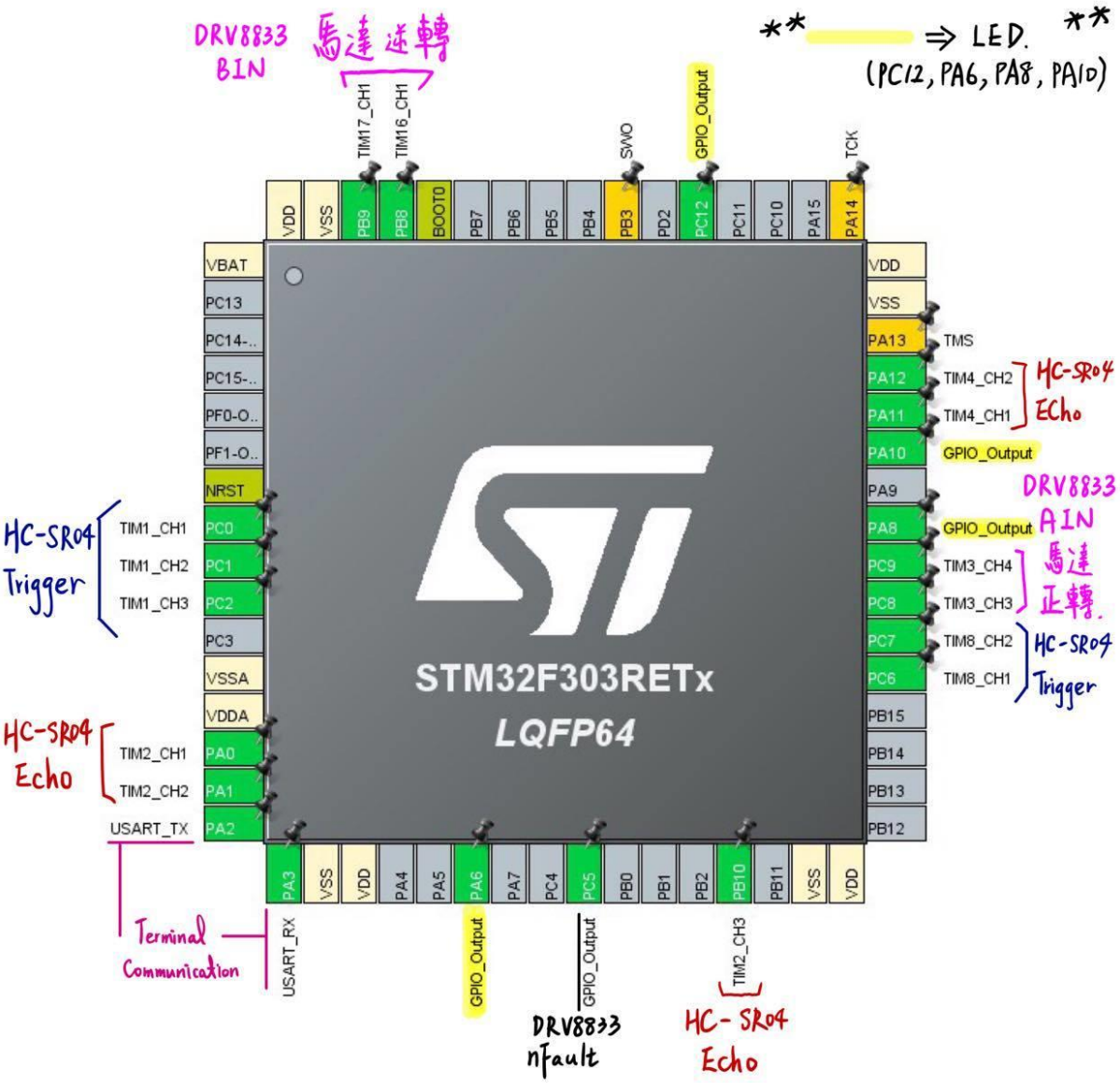


fg. 側視圖/美尻圖



3. 電路設計

控制硬體(stm32)設置：



TIM/GPIO 的使用：

TIM 總共使用 7 個

TIM1 & TIM8 負責超音波 Trigger，TIM2 & TIM4 負責超音波 Echo，TIM3 & TIM16 & TIM17 負責控制馬達。

GPIO Output 總共使用 5 個

PC5 控制馬達晶片的開啟。
PA10/PA6/PA8/PC12 分別控制 左前/右前/左邊/右邊 的 LED 燈。

USART：

PA2/PA3 分別控制 USART_TX & USART_RX 與電腦終端機做溝通，確認超音波測距狀況。

馬達驅動：

我們將馬達的正負極接到 4 個 Channel 分別設為 PWM Generation，將 counter period 設為 100，如此一來可以將轉速分級成百分比制，控制車輪的正逆轉。

分別開啟 TIM3_CH3 & TIM3_CH4 設為控制馬達正轉，TIM16_CH1 & TIM17_CH1 控制逆轉。

超音波測距：

超聲波我們總共使用五組，設置三組測量前方（正前、左前 40 度、右前 40 度），兩組後方測量左右方與船身平行的牆面距離。

<Trigger>

TIM 配置：左前方 TIM1_CH1 / 前方 TIM1_CH2 / 右前方 TIM1_CH3 / 左後方 TIM8_CH1 / 右後方 TIM8_CH2。

超音波的 Trigger 我們統一將 TIM 的 channel 設為 PWM Generation，因為 TIM 的頻率是 8MHz，所以 Prescaler 我們設為 79，這樣一來一個 tick 所需的時間是 10us，counter period 設為 6999。透過以上設定，就可以透過 PWM 自動且規律地發出有效訊號：每 70ms 中有 10us 高電位，進而發出訊號。

為了避免干擾，我們按照 左前、正前、右前、左後、右後 感測器的順序，依序開啟 trigger 發波進行測距，測量完畢就關閉 PWM 換開啟下一個感測器的 PWM 發送訊號測距。

<Echo>

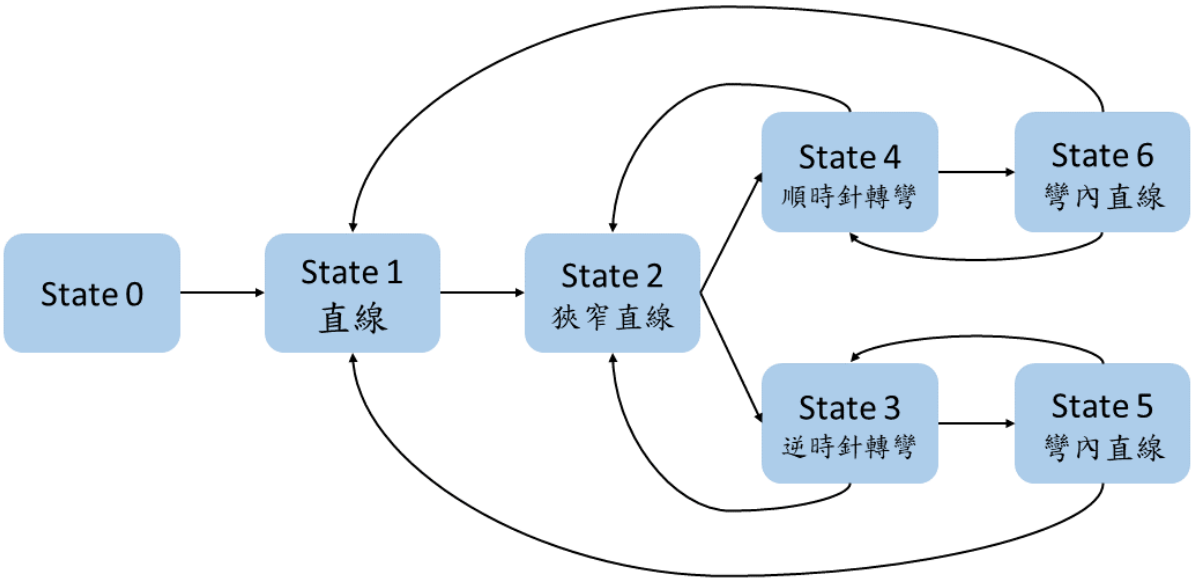
TIM 配置：左前方 TIM2_CH1 / 前方 TIM2_CH2 / 右前方 TIM2_CH3 / 左後方 TIM4_CH1 / 右後方 TIM4_CH2。

超音波的 Echo 我們將 TIM 的 channel 設定為 input capture direct mode，將 counter period 設為最大，polarity selection 設為 Both Edges，如此一來便可以抓到從高低電位間轉換的訊號。在 rising edge 將 counter 歸零，falling edge 記數，記錄 echo 在高電位的總記數，將記數轉換為時間，再計算成距離。

TIM 頻率皆設為 8MHz，prescaler 設為 7，每 1us 記數一次，聲速每 29us 移動 1cm (1cm/(340m*10^2) = 29us)。所以超音波來回距離等於(總記數 cnt*10^-6)/(29*10^-6)，我們所要求的單一趟距離為：cnt/29/2。

控制演算法：

首先定義各個狀態（state），用來代表船不同的行進狀況，各狀態間的轉變條件會在後面說明。



程式會記錄下感測器最近五次測得的距離，利用上述距離，以及目前的狀態，判斷船所在位置、姿態和即將面臨的障礙物，透過切換狀態或改變馬達轉速，做出相對應的動作，以閃避障礙物並通過賽道。不論在任何狀態下，程式都會先判斷船的前方是否離障礙物太近，若是則會控制馬達逆轉使船後退。

各狀態判斷演算法：

state 0：此為初始狀態，船預設會快速前進。

- 若船有向前移動（前方測距連續兩次降低），即會進入 state 1。

state 1：船預設會中速前進。

- 若偵測到船進入狹窄區（左後方及右後方測距皆小於 30 cm），即會進入 state 2。

state 2：船預設會中速前進。

- 若船有向左右偏移（利用左後方或右後方測距的差值，以及其測距之變化），程式會控制馬達轉速修正方向，使船回到賽道中央。
- 若偵測到前方有障礙物（前方測距小 80 cm 且連續下降小於 10 cm 三次），程式會根據後方超音波測距，判斷船的左右位置，決定要進入 state 3 或 state 4。

state 3：程式會根據前方三個感測器的測距大小，並透過左後方感測器的測距輔助，定出船身位於彎內的姿態與位置。

- 若判斷船在彎道內側（前方及右前方測距皆過小），則會控制馬達進行左轉。
- 若判斷船在彎道外側（前方測距尚有空間），包含剛從直線區進入或是已經可以從迴轉區離開，則會控制馬達進行右轉。
- 若前方尚有空間（前方測距或左前方測距大於 70 cm），則會進入 state 5。
- 若前方淨空（前方測距連續三次大於 100 cm），則會回到 state 2。

state 4：此狀態與 state 3 相似，僅將左右側距和左右轉相反。

- 若判斷船在彎道內側（前方及右前方測距皆過小），則會控制馬達進行右轉。
- 若判斷船在彎道外側（前方測距尚有空間），包含剛從直線區進入或是已經可以從迴轉區離開，則會控制馬達進行左轉。
- 若前方尚有空間（前方測距或右前方測距大於 70 cm），則會進入 state 6。
- 若前方淨空（前方測距連續三次大於 100 cm），則會回到 state 2。

state 5：船預設會慢速前進。

- 若船有向左右偏移（利用右後方測距之變化），程式會控制馬達轉速修正方向。

- 若偵測到前方有障礙物（前方測距小於 45 cm），則會回到 state 3 繼續左轉。
- 若前方淨空（前方測距連續三次大於 100 cm），則會回到 state 1。

state 6：此狀態與 state 5 相似，僅將左右側距和左右轉相反。

- 若船有向左右偏移（利用左後方測距之變化），程式會控制馬達轉速修正方向。
- 若偵測到前方有障礙物（前方測距小於 45 cm），則會回到 state 4 繼續右轉。
- 若前方淨空（前方測距連續三次大於 100 cm），則會回到 state 1。

各狀態代碼與燈號表：

※燈號代碼：暗燈(0)、慢閃(1)、快閃(2)、亮燈(3)。燈號順序為「左前、右前、左側、右側」

※燈號表詳見 <4.額外設計>

控制程式碼

//全域變數

//「dis_F、dis_rF、dis_lF、dis_rR、dis_lR」分別代表「前方、右前方、左前方、右後方、左後方」超聲波目前的測距

//「F、rF、lF、rR、lR」代表過去五次的測距

//「LED_rF、LED_lF、LED_right、LED_left」分別代表「右前方、左前方、右側、左側」LED 的燈號代碼

```
double dis_F = 0.0;
double dis_rF = 0.0;
double dis_lF = 0.0;
double dis_rR = 0.0;
double dis_lR = 0.0;
```

```
double F[5], rF[5], lF[5], rR[5], lR[5];
int state = 0;
int posture = 0;
int LED_rF = 0;
int LED_lF = 0;
int LED_right = 0;
int LED_left = 0;
```

//將輸入數字轉換成 char 字元陣列（僅在 debug 時使用）

```
char* distances2char(double* input) {
    int digit = 0;
    for (int i = 0; i < sizeof(input) + 1; i++) {
        int decimal = (int) input[i];
        int fraction = (int) ((input[i] - decimal) * 100);
        for (int j = 0; j < 20; j++) {
            if ((decimal / (int) pow(10, j)) == 0) {
                digit += j;
                break;
            }
        }
        if (fraction != 0)
            digit += 3;
    }
    digit = digit + 2 * sizeof(input) + 3;
    char* output = (char*) malloc(digit * sizeof(char));

    int current = sizeof(input);
    int decimal = (int) input[current];
    int fraction = (int) ((input[current] - decimal) * 100);
    for (int i = 0; i < digit - 3; i++) {
        if (fraction != 0) {
            for (int j = 0; j < 2; j++) {
                output[digit - i - 4] = fraction % 10 + '0';
                fraction = fraction / 10;
                i++;
            }
            output[digit - i - 4] = '.';
            i++;
        }

        output[digit - i - 4] = decimal % 10 + '0';
        decimal = decimal / 10;
        if (decimal == 0) {
            current--;
            if (current >= 0) {
                i++;
                output[digit - i - 4] = ' ';
                i++;
                output[digit - i - 4] = ',';
                decimal = (int) input[current];
                fraction = (int) ((input[current] - decimal) * 100);
            }
        }
    }

    output[digit - 1] = '\0';
    output[digit - 2] = '\r';
    output[digit - 3] = '\n';
    return output;
}
```

//將資料回傳至 terminal 並 print 出（僅在 debug 時使用）

```
void print_data() {

    char* s;

    s = " F distances = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = distances2char(F);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "rF distances = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = distances2char(rF);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "lF distances = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = distances2char(lF);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "rR distances = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = distances2char(rR);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "lR distances = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = distances2char(lR);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "state = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = int2char(state);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "posture = ";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
    s = int2char(posture);
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    s = "\n";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);
}
```

//超音波的 Echo 測距分別回傳至全域變數 dis_lF, dis_F, dis_rF, dis_lR, dis_rR

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim) {
    if (htim->Instance == TIM2) {
        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1) {
            __HAL_TIM_SET_COUNTER(&htim2, 0);
        } else {
            int cnt = __HAL_TIM_GET_COUNTER(&htim2);
            dis_lF = cnt / (double) 29 / (double) 2;
        }

        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1) == 1) {
            __HAL_TIM_SET_COUNTER(&htim2, 0);
        } else {
            int cnt = __HAL_TIM_GET_COUNTER(&htim2);
            dis_F = cnt / (double) 29 / (double) 2;
        }

        if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_10) == 1) {
            __HAL_TIM_SET_COUNTER(&htim2, 0);
        } else {
            int cnt = __HAL_TIM_GET_COUNTER(&htim2);
            dis_rF = cnt / (double) 29 / (double) 2;
        }
    }
    if (htim->Instance == TIM4) {
        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11) == 1) {
            __HAL_TIM_SET_COUNTER(&htim4, 0);
        } else {
            int cnt = __HAL_TIM_GET_COUNTER(&htim4);
            dis_lR = cnt / (double) 29 / (double) 2;
        }

        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_12) == 1) {
            __HAL_TIM_SET_COUNTER(&htim4, 0);
        } else {
            int cnt = __HAL_TIM_GET_COUNTER(&htim4);
            dis_rR = cnt / (double) 29 / (double) 2;
        }
    }
}
```

//進階控制函式，針對不同賽道環境做變化，判斷船的位置與姿態

```
int determine_posture() {

    int posture = 0;

    // Initialize
    if (state == 0) {
        if (F[1] - F[0] > 1 && F[2] - F[1] > 1)
            posture = 1001; // into state 1
        else
            posture = 0; // default(halt)
    }

    // Straight line
    if (state == 1) {
        if (lR[0] < 30 || rR[0] < 30)
            posture = 1012; // into state 2
        else
            posture = 10; // default(straight)
    }

   // Narrow straight line
   if (state == 2) {
       if (F[0] < 80 && F[1] - F[0] < 10 && F[2] - F[1] < 10 && F[3] - F[2] < 10) {
           if (rR[0] - lR[0] > 10)
               posture = 1023; // into state 4
           else
               posture = 1023; // into state 3
       }
       else if (rR[0] - lR[0] > 10) { // left-sided
           if (lR[0] < lR[1] && lR[0] < lR[2])
               posture = 26; // left-sided and towards left wall
           else if (lR[0] > lR[1] && lR[0] > lR[2])
               posture = 25; // left-sided and towards right wall
           else
               posture = 24; // left-sided and straight
       } else if (lR[0] - rR[0] > 10) { // right-sided
           if (rR[0] < rR[1] && rR[0] < rR[2])
               posture = 22; // right-sided and towards right wall
           else if (rR[0] > rR[1] && rR[0] > rR[2])
               posture = 23; // right-sided and towards left wall
           else
               posture = 21; // right-sided and straight
       } else
           posture = 20; // default(straight)
   }
```

```
// Counterclockwise turning
if (state == 3) {
    if (F[0] > 100 && F[1] > 100 && F[2] > 100)
        posture = 1032;
    else if (lR[0] > 85) { // Inner turn
        if (F[0] < 35 && rF[0] < 35)
            posture = 34; // near wall
        else if (F[0] > 70 || lF[0] > 70)
            posture = 1035; // into state 5(finish left turn)
        else
            posture = 35; // default(middle angle)
    } else { // Outer turn
        if (F[0] > 50 && lF[0] > 60)
            posture = 31; // middle angle
        else if (F[0] < 30 && rF[0] < 30 && lF[0] < 40)
            posture = 33; // large angle and near wall
        else if (F[0] < 50 && rF[0] < 50)
            posture = 32; // large angle
        else
            posture = 30; // default(small angle)
    }
}

// Counterclockwise small straight
if (state == 5) {
    if (F[0] > 100 && F[1] > 100 && F[2] > 100)
        posture = 1051; // into state 1
    else if (F[0] < 45)
        posture = 1053; // into state 3
    else if (rR[1] - rR[0] > 1 && rR[0] < rF[1] && rR[0] < rR[2])
        posture = 51; // towards right wall
    else if (rR[0] - rR[1] > 1 && rR[0] > rF[1] && rR[0] > rR[2])
        posture = 52; // towards left wall
    else
        posture = 50; // default(straight)
}
```

```
    // Clockwise turning
    if (state == 4) {
        if (F[0] > 100 && F[1] > 100 && F[2] > 100)
            posture = 1042;
        else if (rR[0] > 85) { // Inner turn
            if (F[0] < 35 && lF[0] < 35)
                posture = 44; // near wall
            else if (F[0] > 70 && rF[0] > 70)
                posture = 1046; // into state 6(finish right turn)
            else
                posture = 45; // default(middle angle)
        } else { // Outer turn
            if (F[0] > 50 && rF[0] > 60)
                posture = 41; // middle angle
            else if (F[0] < 30 && lF[0] < 30 && rF[0] < 40)
                posture = 43; // large angle and near wall
            else if (F[0] < 50 && lF[0] < 50)
                posture = 42; // large angle
            else
                posture = 40; // default(small angle)
        }
    }

    // Clockwise small straight
    if (state == 6) {
        if (F[0] > 100 && F[1] > 100 && F[2] > 100)
            posture = 1061; // into state 1
        else if (F[0] < 45)
            posture = 1064; // into state 4
        else if (lR[1] - lR[0] > 1 && lR[0] < lR[1] && lR[0] < lR[2])
            posture = 61; // towards right wall
        else if (lR[0] - lR[1] > 1 && lR[0] > lR[1] && lR[0] > lR[2])
            posture = 62; // towards left wall
        else
            posture = 60; // default(straight)
    }

    return posture;
}
```

//由狀態決定馬達的轉速以及 LED 燈的燈號

```
void determine(int posture) {

    switch (posture) {
    case 0:
        fast_forward();
        set_LED(2, 2, 2, 2);
        break;
    case 10:
        medium_forward();
        set_LED(0, 0, 0, 0);
        break;
    case 20:
        medium_forward();
        set_LED(3, 3, 0, 0);
        break;
    case 21:
        medium_forward();
        set_LED(3, 3, 0, 0);
        break;
    case 22:
        left_turn();
        set_LED(3, 3, 2, 0);
        break;
    case 23:
        medium_forward();
        set_LED(3, 3, 0, 0);
        break;
    case 24:
        medium_forward();
        set_LED(3, 3, 0, 0);
        break;
    case 25:
        medium_forward();
        set_LED(3, 3, 0, 0);
        break;
    case 26:
        right_turn();
        set_LED(3, 3, 0, 2);
        break;
    case 30:
        right_turn();
        set_LED(0, 3, 0, 2);
        break;
    case 31:
        right_turn();
        set_LED(0, 3, 0, 2);
        break;
    case 32:
        left_turn();
        set_LED(0, 3, 2, 0);
        break;
    case 33:
        left_turn();
        set_LED(0, 3, 2, 0);
        break;
    case 34:
        left_turn();
        set_LED(0, 3, 2, 0);
        break;
    case 35:
        left_turn();
        set_LED(0, 3, 2, 0);
        break;
```

```
case 40:
    left_turn();
    set_LED(3, 0, 2, 0);
    break;
case 41:
    adjust_left();
    set_LED(3, 0, 2, 0);
    break;
case 42:
    right_turn();
    set_LED(3, 0, 0, 2);
    break;
case 43:
    right_turn();
    set_LED(3, 0, 0, 2);
    break;
case 44:
    right_turn();
    set_LED(3, 0, 0, 2);
    break;
case 45:
    right_turn();
    set_LED(3, 0, 0, 2);
    break;
case 50:
    slow_forward();
    set_LED(2, 2, 0, 0);
    break;
case 51:
    adjust_left();
    set_LED(2, 2, 2, 0);
    break;
case 52:
    adjust_right();
    set_LED(2, 2, 0, 2);
    break;
case 60:
    slow_forward();
    set_LED(1, 1, 0, 0);
    break;
case 61:
    adjust_left();
    set_LED(1, 1, 2, 0);
    break;
case 62:
    adjust_right();
    set_LED(1, 1, 0, 2);
    break;
```

```
    case 1001:
        state = 1;
        break;
    case 1012:
        state = 2;
        break;
    case 1023:
        state = 3;
        break;
    case 1024:
        state = 4;
        break;
    case 1032:
        state = 2;
        break;
    case 1035:
        state = 5;
        break;
    case 1042:
        state = 2;
        break;
    case 1046:
        state = 6;
        break;
    case 1051:
        state = 1;
        break;
    case 1053:
        state = 3;
        break;
    case 1061:
        state = 1;
        break;
    case 1064:
        state = 4;
        break;
    }

    if (F[0] < 15)
        reverse(500);
}
```

//各種馬達轉換函式

```
void fast_forward() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 80);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 88);
}

void medium_forward() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 75);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 83);
}

void slow_forward() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 72);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 80);
}
```

```
void left_turn() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 80);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 80);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 0);
}

void adjust_left() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 100);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 0);
}

void right_turn() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 0);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 80);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 100);
}

void adjust_right() {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 0);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 100);
}

void reverse(int t) {
    //  right wheel
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 0);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 100);
    //  left wheel
    __HAL_TIM_SET_COMPARE(&htim16, TIM_CHANNEL_1, 100);
    __HAL_TIM_SET_COMPARE(&htim17, TIM_CHANNEL_1, 0);

    HAL_Delay(t);
}
```

//LED 燈控制函式

```
void set_LED(int lf, int rf, int l, int r) {
    LED_lF = lf;
    LED_rF = rf;
    LED_left = l;
    LED_right = r;
}
void LED_control() {
    switch (LED_lF) {
    case 0:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 0);
        break;
    case 1:
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_10);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 1);
        HAL_Delay(10);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 0);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 1);
        break;
    }

    switch (LED_rF) {
    case 0:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, 0);
        break;
    case 1:
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, 1);
        HAL_Delay(10);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, 0);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, 1);
        break;
    }
```

```
    switch (LED_right) {
    case 0:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, 0);
        break;
    case 1:
        HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_12);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, 1);
        HAL_Delay(10);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, 0);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, 1);
        break;
    }

    switch (LED_left) {
    case 0:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);
        break;
    case 1:
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_8);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 1);
        HAL_Delay(10);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 1);
        break;
    }
}
```

//以下為 main 函式內的程式碼

//初始化

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */


  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_TIM8_Init();
  MX_TIM16_Init();
  MX_TIM17_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
    char* s;
    s = "connection success.\n\r";
    HAL_UART_Transmit(&huart2, (uint8_t *) s, strlen(s), 0xFFF);

    int index = 0;

    for (int i = 0; i < 5; i++) {
        F[i] = 150;
        rF[i] = 150;
        lF[i] = 150;
        rR[i] = 150;
        lR[i] = 150;
    }
```

//開啟馬達 PWM，以及開啟 Echo 腳位 Timer 的 input capture

```
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);
HAL_TIM_PWM_Start(&htim16, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim17, TIM_CHANNEL_1);

HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_3);
HAL_TIM_IC_Start_IT(&htim4, TIM_CHANNEL_1);
HAL_TIM_IC_Start_IT(&htim4, TIM_CHANNEL_2);
```

//程式會記錄下感測器最近五次測得的距離，更新的距離會遞補上去

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
  while (1) {

      for (int i = 4; i > 0; i--) {
          F[i] = F[i - 1];
          rF[i] = rF[i - 1];
          lF[i] = lF[i - 1];
          rR[i] = rR[i - 1];
          lR[i] = lR[i - 1];
      }
```

//超音波 trigger 的 PWM 分別開啟關閉

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_Delay(70);
if (dis_lF > 150)
    lF[0] = lF[1];
else
    lF[0] = dis_lF;
HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
HAL_Delay(70);
if (dis_F > 120 && dis_F < 180)
    F[0] = F[1];
else
    F[0] = dis_F;
HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
HAL_Delay(70);
if (dis_rF > 150)
    rF[0] = rF[1];
else
    rF[0] = dis_rF;
HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);

HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);
HAL_Delay(70);
if (dis_lR > 150)
    lR[0] = lR[1];
else
    lR[0] = dis_lR;
HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_1);

HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_2);
HAL_Delay(70);
if (dis_rR > 150)
    rR[0] = rR[1];
else
    rR[0] = dis_rR;
HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);
```

//執行判斷函式

```
        index++;
        if (index <= 10)
            continue;

        posture = determine_posture();
        determine(posture);
        LED_control();

//      print_data();
    }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

  /* USER CODE END 3 */
}
```

4. 額外設計

由於我們的控制設計使用狀態切換會有不同的控制效果，所以需要確認船處在哪個狀態，故配置狀態燈以做分析判斷。

前後方左右處各兩顆 LED 燈，總共 4 顆燈製作信號。

燈的狀態有四種：暗燈(0)、慢閃(1)、快閃(2)、亮燈(3)。藉此開發出以下燈號表（共 2 頁），並確保每位船員熟稔此份燈號表。

電路配置：

設置控制燈泡的腳位為 GPIO Output，分別將 PA10/PA6/PA8/PC12 的腳位控制左前/右前/左邊/右邊的 LED 燈。

狀態	代碼	燈號	情況	動作
0	0	2222	靜止	快速前進
	1001		開始前進	進入 state 1
1	10	0000	走直線	中速前進
	1012		進入狹窄區	進入 state 2
2	20	3300	走直線	中速前進
	21		船身正且偏右側	中速前進
	22	3320	船身右斜且偏右側	左調整
	23	3300	船身左斜且偏右側	中速前進
	24		船身正且偏左側	中速前進
	25		船身右斜偏左側	中速前進
	26	3302	船身左斜且偏左側	右調整
	1023		偵測前方障礙物，且進入右轉點	進入 state 3
	1024		偵測前方障礙物，且進入左轉點	進入 state 4

3	30	0302	小角度	右轉
	31		中角度	
	32	0320	大角度	左轉
	33		大角度且離牆近	
	34		彎內接近牆壁	
	35		彎內呈中間角度	
	1032		完成轉彎且前方淨空	進入 state 2
	1035		完成轉彎	進入 state 5
4	40	3020	小角度	左轉
	41		中角度	
	42	3002	大角度	右轉
	43		大角度且離牆近	
	44		彎內接近牆壁	
	45		彎內呈中間角度	
	1042		完成轉彎且前方淨空	進入 state 2
	1045		完成轉彎	進入 state 5
5	50	2200	走直線	慢速前進
	51	2220	船身右斜	左調整
	52	2202	船身左斜	右調整
	1051		前方淨空	進入 state 1
	1053		偵測到前方牆壁，進入左轉點	進入 state 3
6	60	1100	走直線	慢速前進
	61	1120	船身右斜	左調整
	62	1102	船身左斜	右調整
	1061		前方淨空	進入 state 1
	1064		偵測到前方牆壁，進入右轉點	進入 state 4

---THE END---