Programming Phase

Objectives:

This task aims to integrate knowledge from previous tasks by building a fully functional system, focusing on **servo motor** control, **ADC signal** reading from **Potentiometers** and **Joysticks**, and mapping analog inputs to servo angles while managing multiple operating modes.

Main Task:

Control 4 servo motors using 4 potentiometers and 2 joysticks, ensuring that the entire system is fully functional

Subtask 1:

Control the **4 servos** using the **4 potentiometers**, one potentiometer per servo. Using your **main robotic arm**, and the **potentiometers controller arm**.

- Map the potentiometer's analog input range (e.g. 0 - 4096) to the servo's angle range (e.g. 0°–180°). It is not necessary to use the full range of the potentiometers and servos, use only the range required for your design.

Subtask 2:

Control **the Main robotic arm** with **the Joystick controller**, where each joystick module will be controlling two servos:

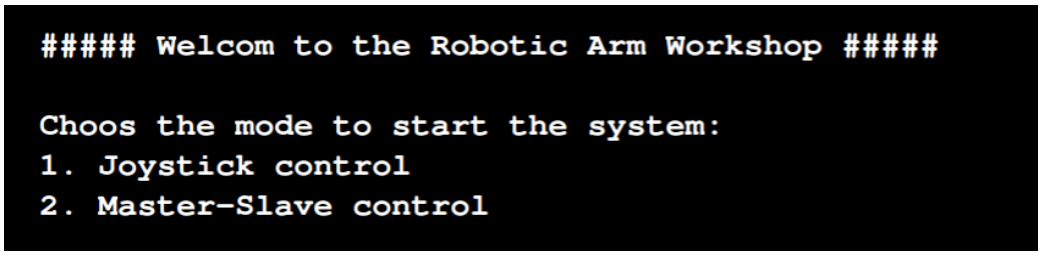
- X-axis → Servo 1
- Y-axis → Servo 2

Subtask 3:

Combine the two previous codes into one program, so the robotic arm can run in two modes, joystick control and master-slave (potentiometer) control. Use one of the programmable push-buttons to switch between the two modes, and use two programmable LEDs to show which mode is active.

Optional Task:

After you complete the main task and have your push-button ready to switch the modes, in addition to the button, you may use your phone and the Bluetooth module (HC-06) to switch the mode as well, as the system starts, none of the modes will be chosen, a message will be transmitted to the user to choose between the two modes (you should declare the two modes in the message), the user then will replay with the chosen mode to start the system.



```
##### Welcom to the Robotic Arm Workshop #####

Choos the mode to start the system:
1. Joystick control
2. Master-Slave control
```

Extra Task: (No tokens, just for your knowledge)

Using FreeRTOS, create 3 tasks to blink the 3 LEDs on the main board:

- Task 1: Blink LED1 at 2 Hz
- Task 2: Blink LED2 at 5 Hz
- Task 3: Blink LED3 at 10 Hz

Use ***HAL_Delay()*** to control the blinking frequency.

Hints:

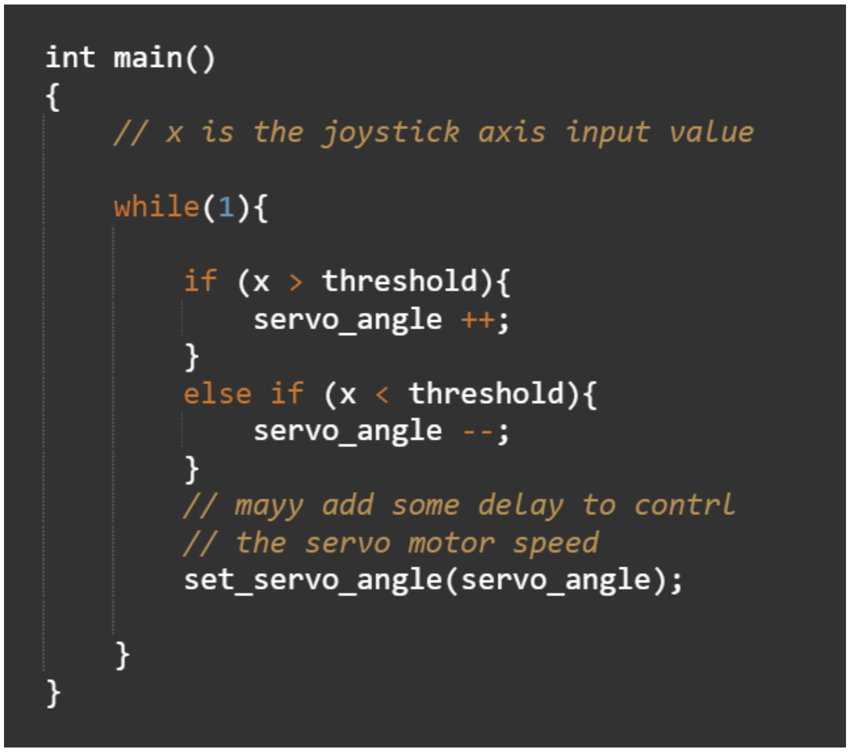
- You may use the general mapping formula:

$$\text{mappedValue} = \text{outMin} + \frac{(value - inMin) \cdot (outMax - outMin)}{(inMax - inMin)}$$

  Where the ***inRange*** is the constant range value of the input sensor (potentiometer), and the ***outRange*** is the constant range value of the output actuator (servo motor). ***Value*** is the reading of the input sensor (potentiometer), ***mappedValue*** is the value to write to the output actuator (servo motor).

- Don’t use the same technique of mapping with the Joystick input. Instead you may use the ***angle increment/decrement*** technique. If the axis reading is larger than a threshold, the angle of the servo will keep incrementing, while if the reading is smaller, it will keep decrementing, when the joystick is in normal position, it will keep the last angle value.

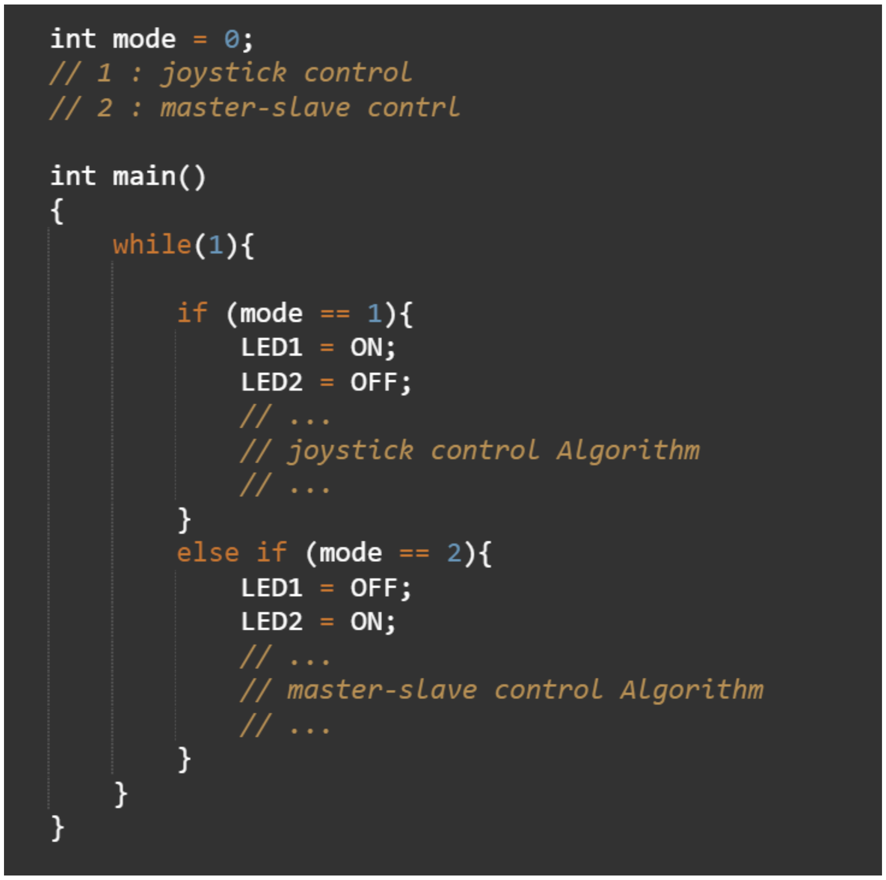
- When the joystick is at rest, it usually doesn’t stay fixed at one exact value but instead fluctuates slightly around it. To prevent these small changes from being misread as movement, you can define a ***dead zone*** around the resting point. For example, if the joystick’s range (for example) is 0–10 and the resting value is around 5, you can set a threshold so that any reading between 4 and 6 is ignored. Only values outside this range will be treated as actual movement.



```
int main()
{
    // x is the joystick axis input value

    while(1){

        if (x > threshold){
            servo_angle ++;
        }
        else if (x < threshold){
            servo_angle --;
        }
        // mayy add some delay to contrl
        // the servo motor speed
        set_servo_angle(servo_angle);

    }
}
```

This is not a real STM32 code, it is just for example

- For ***Subtask 3***, you may use a **global variable** that will store the mode value, then based on the mode variable value, the code will be executing a certain block of code. You may use **external interrupt** to check the button reading, and switch the mode when the button is pressed.



```
int mode = 0;
// 1 : joystick control
// 2 : master-slave contrl

int main()
{
    while(1){

        if (mode == 1){
            LED1 = ON;
            LED2 = OFF;
            // ...
            // joystick control Algorithm
            // ...
        }
        else if (mode == 2){
            LED1 = OFF;
            LED2 = ON;
            // ...
            // master-slave control Algorithm
            // ...
        }
    }
}
```

This is not a real STM32 code, it is just for example

Servo motor safety rules :

- **Avoid Manually Forcing the Servo Shaft:** Do not try to manually rotate or force the servo motor shaft or gears. Servo motors have delicate internal gears that can break easily when forced by hand.
- **IMPORTANT Do Not Block the Servo While Moving:** Avoid physically blocking the servo motor shaft during operation. This can cause the servo to draw its maximum current, potentially overheating and damaging the servo, and in some cases causing overcurrent that may harm the STM32 board.

Submission:

- Record a video of each subtask and send it to your mentor.
- Complete and submit this task in one PDF file for both main and optional (if done) tasks.
- Add a picture of your circuit connections to the PDF file.
- Add a screenshot of the configurations of IOC to the PDF file.
- Add a screenshot of the code of each subtask to the PDF file (only what you have modified in the generated code).
- Name the PDF files with task11_groupx_your_name, (replace x with your group number).
- This task should be submitted before 24th Sept 10:00pm (Malaysia time), 05:00pm (Makkah time).
- Your mentor must approve your task answer file before submission.
- Submit the PDF file to the Google form.