Exercise Manual
For

SC2104/CE3002
Sensors, Interfacing and Digital Control

Practical Exercise #5:
A Self-Levelling Platform

Venue: SCSE Labs

COMPUTER ENGINEERING COURSE

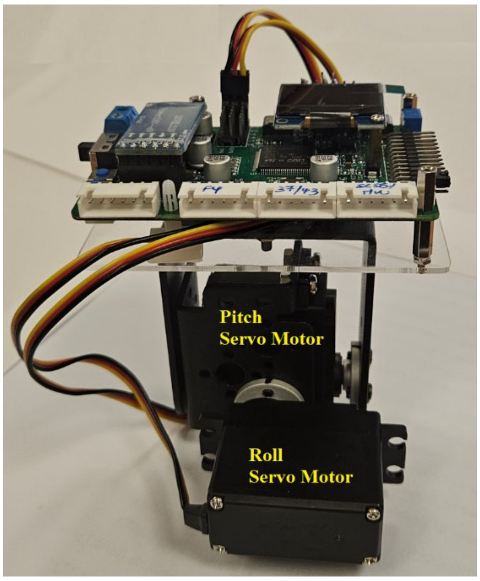
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

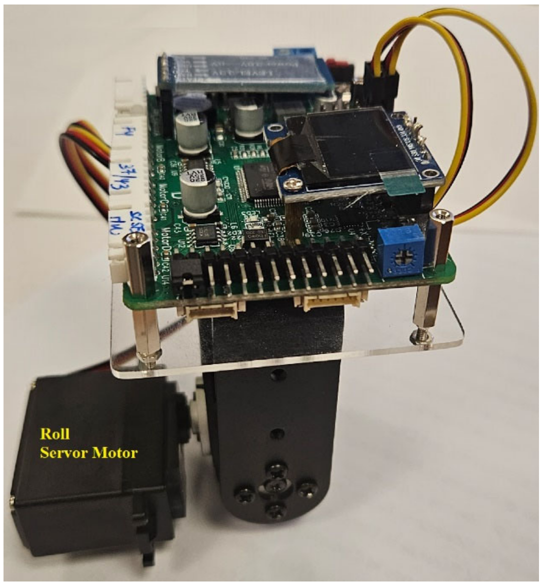
These exercises are to enable student to apply the knowledge that they have learnt during the previous practical lessons to implement a self-levelling platform. Student will use the IMU to measure the roll and Pitch angles of a platform, and continuously adjust the servo motors to maintain the platform level using digital PID control strategy.

Equipment and accessories required

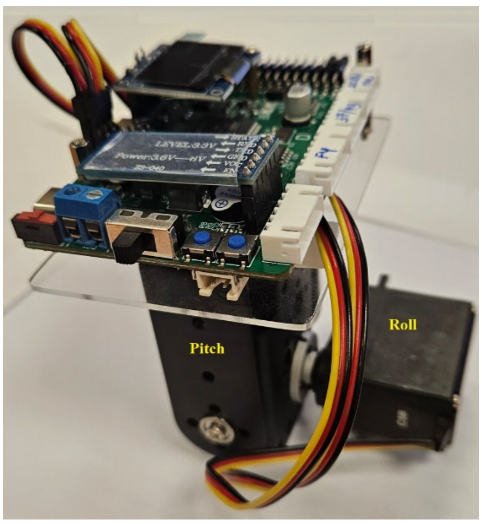
i) One desktop computer installed with STM32CubeIDE and SerialPlot.
ii) One STM32F4 board (Ver D)
iii) One ST-LINK SWD board
iv) Two servo motors assembled with levelling platform
v) USB-C cable for serial communication and power for the platform



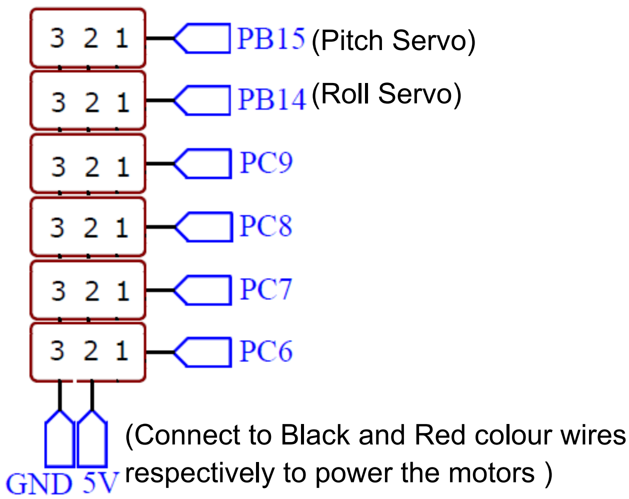
(a)



(b)



(c)



(d)

Introduction

In this practical exercise, you are given a STM32F board mounted on a platform that is driven by two servo motors which can be rotated to adjust the pitch and roll orientations of the platform. The rotations of the two servo motors are in turn controlled by the STM32F board, which issues the adjustment commands based on the readings it obtains from its IMU sensors. (See figure a, b, and c on page 2)

The aim of the exercises is to implement a program using the STM32F board that measures the roll and pitch angles of the platform, and continuously adjust the servo motors such that the platform is maintained at certain target angles; E.g., for horizontally level, the pitch angle and roll angle are to be maintained as close to 0 degree as possible.

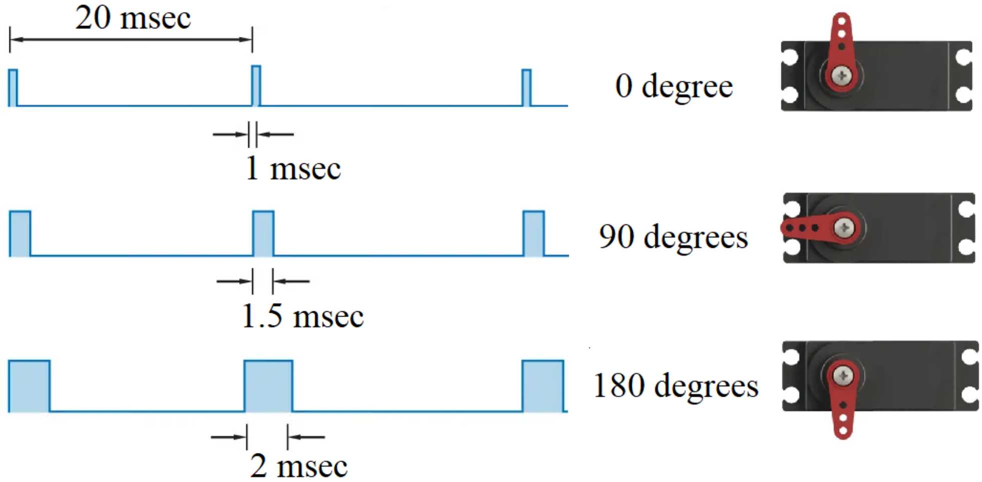
The two servo motors are driven by PWM waveforms generated by the STM32F board through its GPIO header pins PB14 (for the roll servo motor) and PB15 (for the pitch servo motor) as shown in figure d on page 2.

You will be using the STM32F4 Ver D board (used in Lab 1 to 4 exercises) and the STM32CubeIDE for this lab exercises, together with the SerialPlot software to observe the movement of the platform. The program code required for the exercises are provided in a zip file available on NTULearn course site, which you will use to setup a new project.

1. Operation of the Levelling System

1.1 Servo Motor

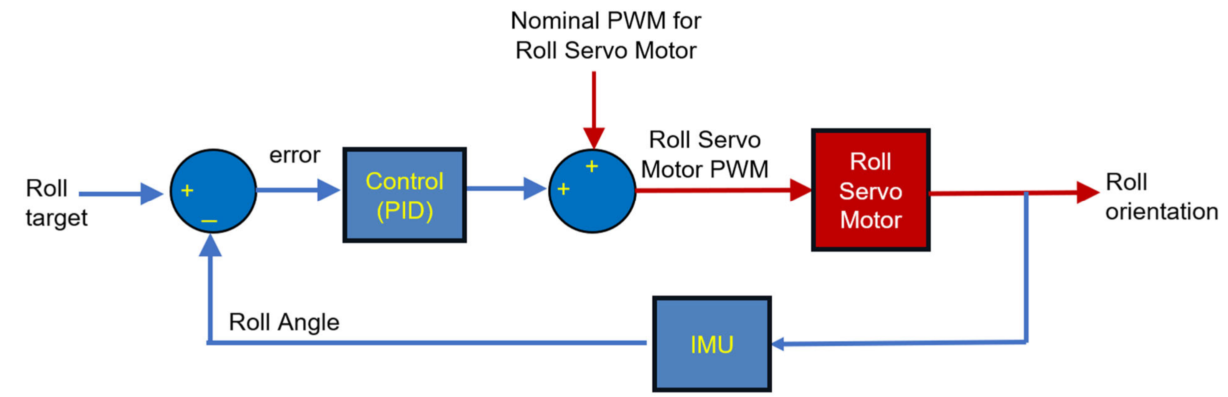
Servo motor is controlled by a PWM waveform applied to its terminal as shown below (refer to lecture note topic 7 for detail)

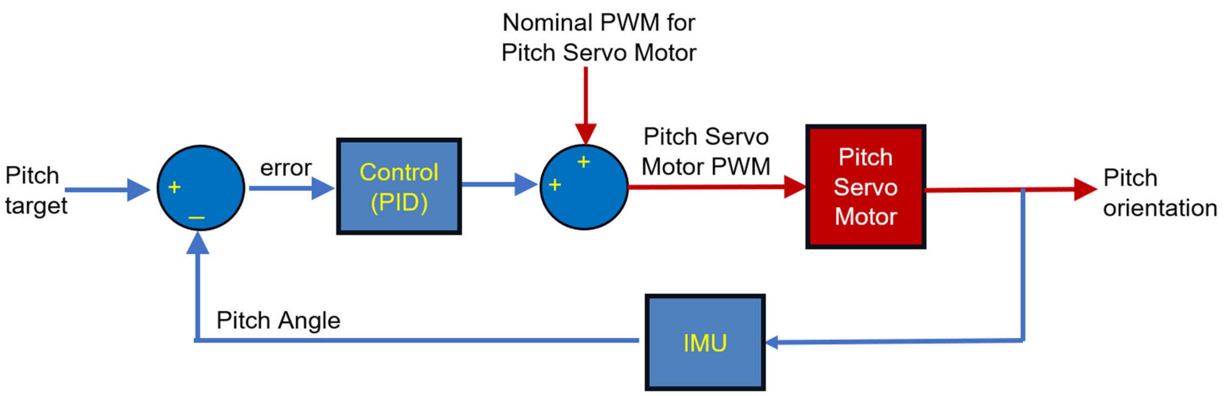


As such, there must be a nominal PWM applied to the servo motor at all times in order for it to maintain its position.

1.2 Levelling platform Control Loop Model

The following show the model of the system that is used in the implementation of the platform levelling using feedback control.





A nominal PWM is applied to the servo motor upon start up to orient its shaft to position the platform at a specific angle (the target levelling angle).

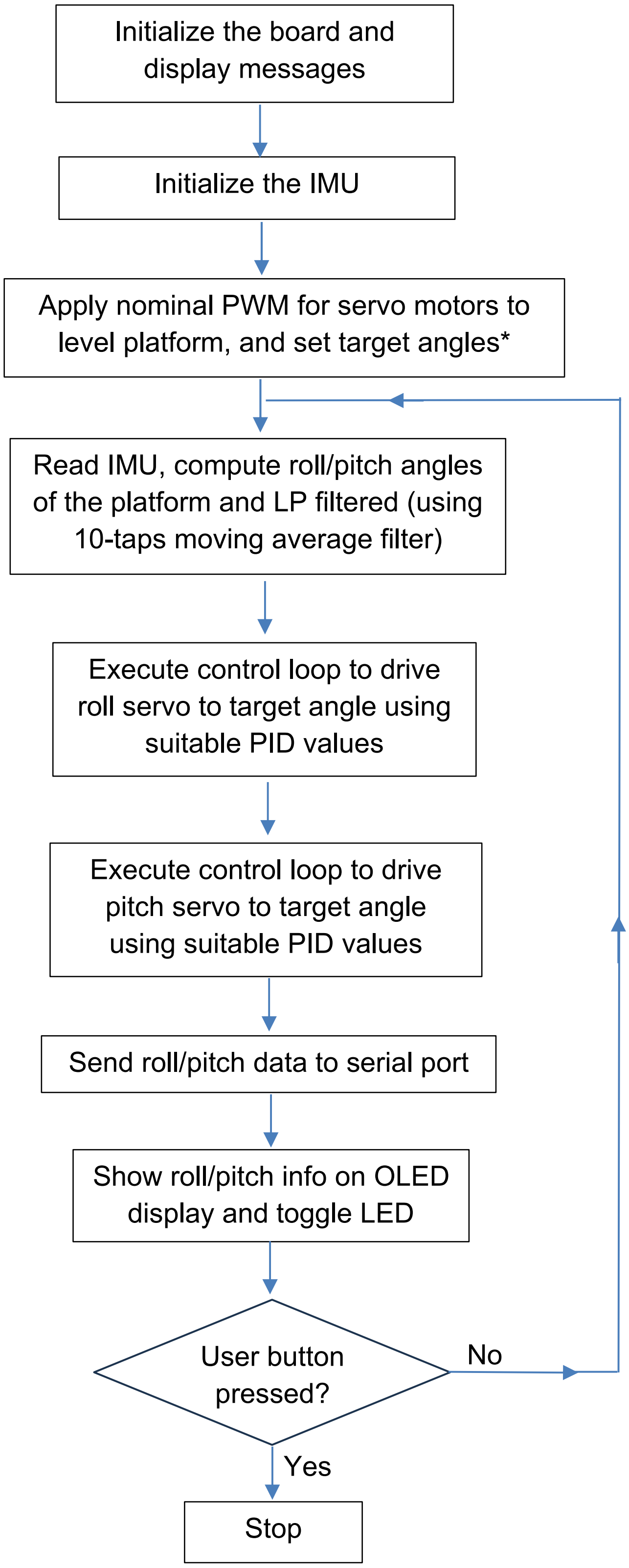
If the platform is tilted, addition PWM waveform is needed to adjust the servo motor to bring the platform back to the target angle (i.e. the levelling angle).

In a feedback loop system, the additional PWM waveform is generated by the control system based on the error between the measure angle and the roll angle.

Food for thought: Once the tilted platform is adjusted by the control system toward the target angle, the error become smaller and eventually becomes zero. This causes the addition PWM component to also reduce to zero, and make the platform drops away from the target angle?

1.3. Program logic

The following is the overview of the flowchart of the program provided.



* To set the levelling target angles, hold the platform at the angles you want it to level at (E.g. roll and pitch angle to be 0 degrees for horizontally level)

2. Sample Program

Download the Lab5 zip file that contains the following files from NTULearn course site, extract and store them on the computer (or your thumb drive).

Lab 5.ioc
c files: main.c, MPU6050.c, oled.c, stm32f4xx_it.c
header file: main.h, oled.h, oledfont.h

Import these files into a new STM32 project for this exercise. (Refer to Practical Exercise 4 manual for the steps if you cannot remember.)

- Build/compile the project and download the code onto the STM32F4 board. (The code should be compiled with no error). The servo motors should start to rotate to the positions as shown in Figure a, b, and c (such that the roll and pitch adjustments are orthogonal to each other) if the program executes successfully.

  Recall from Lab 2:
  Floating-point support for `printf()` is not enabled by default in STM32CubeIDE. To do so, we need to add the flag `-u _printf_float` to its linker option. (-u flag is to force linking and loading of library module that is undefined, in this case, `_printf_float`)

- Connect the USB serial port (the middle USB port on the board) to the computer and run the SerialPlot program (with 115200 baud rate and using 'comma' column delimiter) to observe the data sent by the program, which are the current roll angle, roll target angle, current pitch angle and pitch target angle.

  With the default PID parameters used in the program, the step response of the roll angle (vs its target) should be similar to as shown in the following diagram when you roll the platform in both directions.

2.1 Program Code overview

The follow is the snippet of the main part of the program code, the while loop (starting at line 487, or nearby) that repeatedly calls the control loop to adjust the platforms.

```
484    /* Infinite loop */
485    /* USER CODE BEGIN WHILE */
486      //-------------------------------------------------------------------------------
487      while (start == 1){ //do until User button is pressed
488        get_angles();     //get roll and pitch angles from IMU
489
490        //filtering the angles using FIR moving average filter
491        roll_angle = mov_avr_roll(roll_KF);
492        pitch_angle = mov_avr_pitch(pitch_KF);
493
494        // Execute control loop for Roll angle with its PID parameters
495        Kp = 1;      // P
496        Ki = 0;      // I
497        Kd = 0;      // D
498        roll_PID(roll_angle, Kp, Ki, Kd);
499
500        // Execute control loop for Pitch angle with its PID parameters
501        Kp = 1;      // P
502        Ki = 0;      // I
503        Kd = 0;      // D
504        pitch_PID(pitch_angle, Kp, Ki, Kd);
505
506        Serial_tx(); //send roll & pitch angles, roll & pitch targets to serial port
507
508        OLED_disp_msg3(); //display information on OLED
509
510        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_12); // toggle LED for heart beat indicator
511      } // while(start == 1)
512
513    /* USER CODE END WHILE */
```

The following shows the control loop function **roll_PID()** (starting at line 376, or nearby) that is used to adjust the PWM for the roll servo motor.

```
366  //*************************************************************************************
367  // Control loop for Roll angle
368  float roll_angle;              // current Roll angle of platform
369  float roll_target;             // target Roll angle
370  float roll_error=0;            // error between target and actual
371  float roll_error_area = 0;     // area under error - to calculate I for PI implementation
372  float roll_error_old = 0;      // use these 3 variables to calculate D for PD/PID control
373  float roll_error_change = 0;
374  float roll_error_rate = 0;
375
376  void roll_PID(float angle, const float kp, const float ki, const float kd)
377  {   int roll_ServoVal;          // PWM servo value for roll servo motor
378      const int ServoMax = 220; // maximum server value
379      const int ServoMin = 50;  // maximum server value
380
381      roll_error = roll_target - angle;
382
383      roll_ServoVal = (int)(roll_servo_target-(roll_error*kp)); //   control loop
384
385      if (roll_ServoVal > ServoMax)  // Clamp the PWM value to maximum value
386         roll_ServoVal = ServoMax;
387      if (roll_ServoVal < ServoMin)  // Clamp the PWM value to minimum value
388         roll_ServoVal = ServoMin;
389
390      htim12.Instance->CCR1 = roll_ServoVal;       // send PWM duty cycle to servo motor
391
392      sprintf(temp[9], "rS:%3d", roll_ServoVal);  // show Servo PWM value on OLED display
393      OLED_ShowString(70, 20, &temp[9]);
394      OLED_Refresh_Gram();
395  }
```

Study these program code to understand how the instructions execute the Roll angle to level the platform. (Q: What type of control system is it implementing?)

3 Exercises: Step response of the system

With the USB cable connected to the computer, the values of the Roll and Pitch angles can be displayed on the computer using the SerialPlot programme for you to finetune the control parameters of the system.

3.1 Practical 1: Step response of the Platform

i) Tilt the platform abruptly <u>along the Roll axis</u> and observe the step response of the system using the `SerialPlot` program (which should be similar to the figure shown on page 6). You should also observe that the platform will try to return to the levelling position.

ii) Change the Kp value (at line 495) used for the Roll control loop and observe the step responses.

- Reduce the value (e.g. to 0.5 or smaller)
- Increase the value in steps of 0.5 and observe the change in the response.

Q: does the response behave as you expected?

3.2 Practical 2: Enhancing the control loop

You should realize that the control loop implemented is a proportion control system, which hence exhibit a couple of shortcomings. As such, the control system can be enhanced by adding the other parameters into the control loop.

i) Add the integration operation to the code in **`roll_PID()`** function to implement a PI control loop. (Hint – this can be done by using the variable **`roll_error_area`** (declared at line 371) and the sampling interval **`dt`**, which is a global variable that is continuously updated in the while loop during operation as shown below

```
// get the loop elapse time = sampling interval
millisNow = HAL_GetTick(); // store the current time for next round
dt = (millisNow - millisOld)*0.001; // time elapsed in millisecond
millisOld = millisNow; // store the current time for next round
```

Activate this integration function by setting the **Ki** parameter (at line 496) to a non-zero value (e.g. about value of 1 to 2) and observe its effect on the performance of the system. (Start with a small value and increase it at 0.5 interval)

ii) The performance of the system can be further improved by adding the derivative component to the control function to form a PID control loop. Use the variables **`roll_error_old,`** **`roll_error_change`** and **`roll_error_rate`** (declared at lines 372 to 374) to implement the D component and activate it by using a non-zero value (e.g. about 0.5) for the **Kd** parameter (at line 497) .

3.3 Practical 3: Activate the Pitch control loop

Once you have tuned the PID control loop for the Roll angle, you can duplicate the code for the Pitch angle through the function `pitch_PID()`, located after the `roll_PID()` function in the given program.

```
398 //------------------------------------------------------------------------------------
399 // Control loop for Pitch angle
400 float pitch_angle;              // current pitch angle of platform
401 float pitch_target;             // target angle
402 float pitch_error = 0;          // error between target and actual
403 float pitch_error_area = 0;     // area under error - to calculate I for PI implementation
404 float pitch_error_old = 0, pitch_error_change = 0, pitch_error_rate = 0;
405
406 void pitch_PID(float angle, const float kp, const float ki, const float kd)
407 {   int pitch_ServoVal;         // PWM servo value for Pitch servo motor
408     const int ServoMax = 220; // maximum server value
409     const int ServoMin = 50;  // maximum server value
410
411     return; // disable this function - remove this line to implement Pitch control loop
412     htim12.Instance->CCR2 = pitch_ServoVal;  // send duty cycle to Pitch servo motor
413
414     sprintf(temp[9], "pS:%3d", pitch_ServoVal);  // show Servo PWM value on OLED display
415     OLED_ShowString(0, 20, &temp[9]);
416     OLED_Refresh_Gram(); // refresh OLED display
417 }
```

Remove the return statement (at line 411) and add the code to implement a P control loop first. Tune its Kp parameter until it performs as required. Then add code to form a PI control loop, and eventually a PID control loop.

You should disable the Roll control loop while tuning the Pitch control loop, in order to remove the interaction between the two control loops that make tuning complicated. The roll control loop can be disabled by setting its PID parameters to 0 values.

```
    // Execute control loop for Roll angle with its PID parameters
    Kp = 0;      // P
    Ki = 0;      // I
    Kd = 0;      // D
    roll_PID(roll_angle, Kp, Ki, Kd);
```

3.4 Practical 4: A Self-Levelling Platform

After separately implemented and tested the PID code for both the Roll and Pitch control loops, activate both of them together to check that the platform can perform the self-levelling operation with satisfactorily performance. You may need to further adjust some of the parameters due to the interaction of the two loops during operation.

Endnote: Through the series of practical exercises you performed in this course, we hope that you can see and understand how smart sensors and digital control system can be useful for many real-life applications.