Exercise Manual
For

SC2104/CE3002
Sensors, Interfacing and Digital Control

Practical Exercise #4:
Digital Control
of
DC Motor

Venue: SCSE Labs

COMPUTER ENGINEERING COURSE

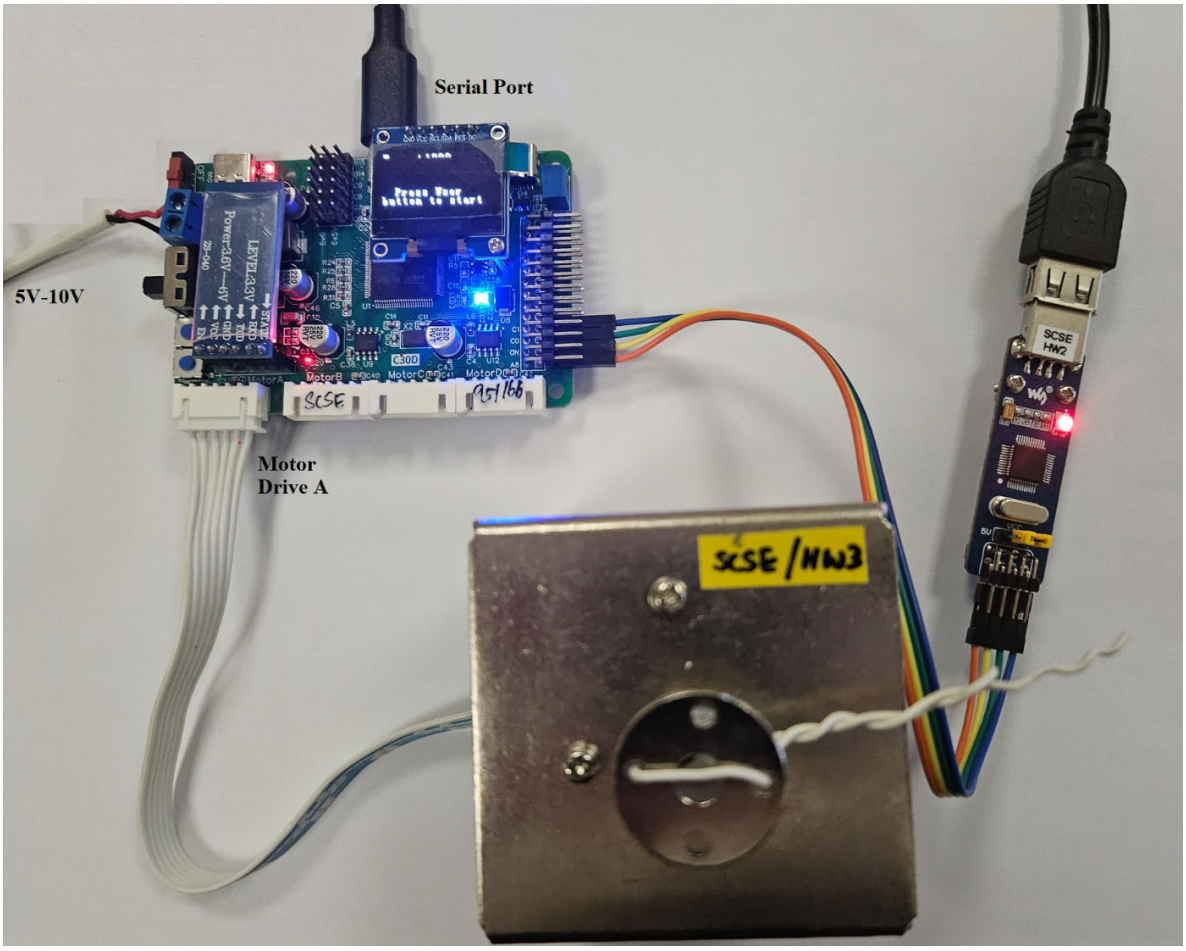
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

These exercises are to introduce student to the operation of a digital control system (using the STM32F4 board) to control the rotation of a DC motor. Student will observe the control system's effects on the DC motor operation based on the P, I and D control algorithms.

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) One 20:1 gear DC Motor (attached with 26-poles/13 pulses Hall Effect Encoder)
v) External DC power supply (5V to 10V)



Introduction

Digital control is now the predominant mean for many control applications. In this exercise, you will go through a sequence of tasks designed to let you be familiarized with the concepts of digital PID control and use it to control the position of a DC motor and observe its behaviours with different parameter settings.

1. Setup

You will be using the STM32F4 Ver D board (used in Lab 1 to 3 exercises) and the STM32CubeIDE for this lab exercises, together with the SerialPlot software to observe the step response of the motor. 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.

1.1. Obtain the files

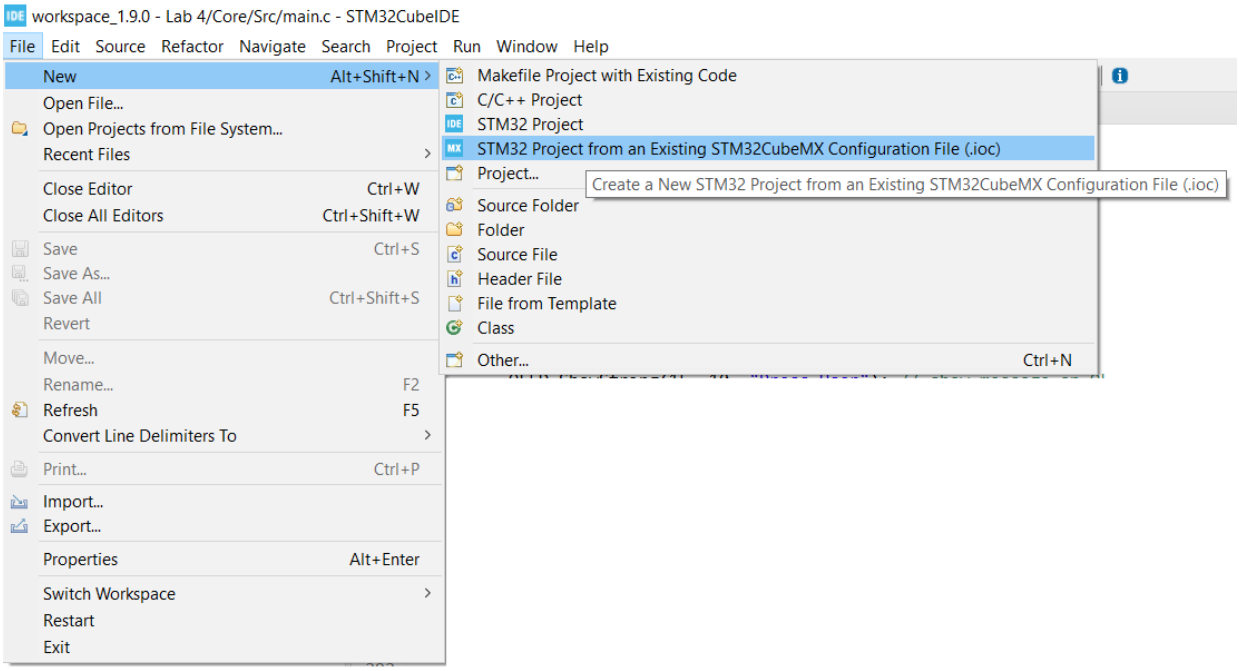
Download the zip file that contains the following files from NTULearn course site, extract and store them on the computer (or your thumb drive).

Lab 4.ioc
main.c
oled.c
stm32f4xx_it.c
main.h
oled.h
oledfont.h

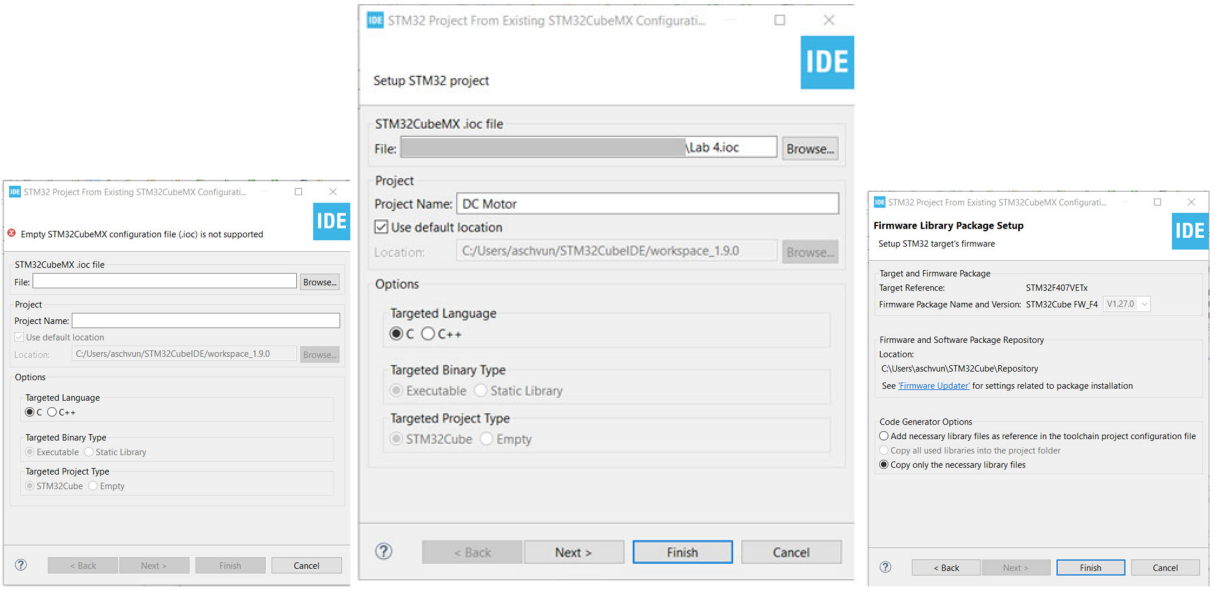
You will import these files into the project for this exercise.

1.2. Setting up the STM32 Project

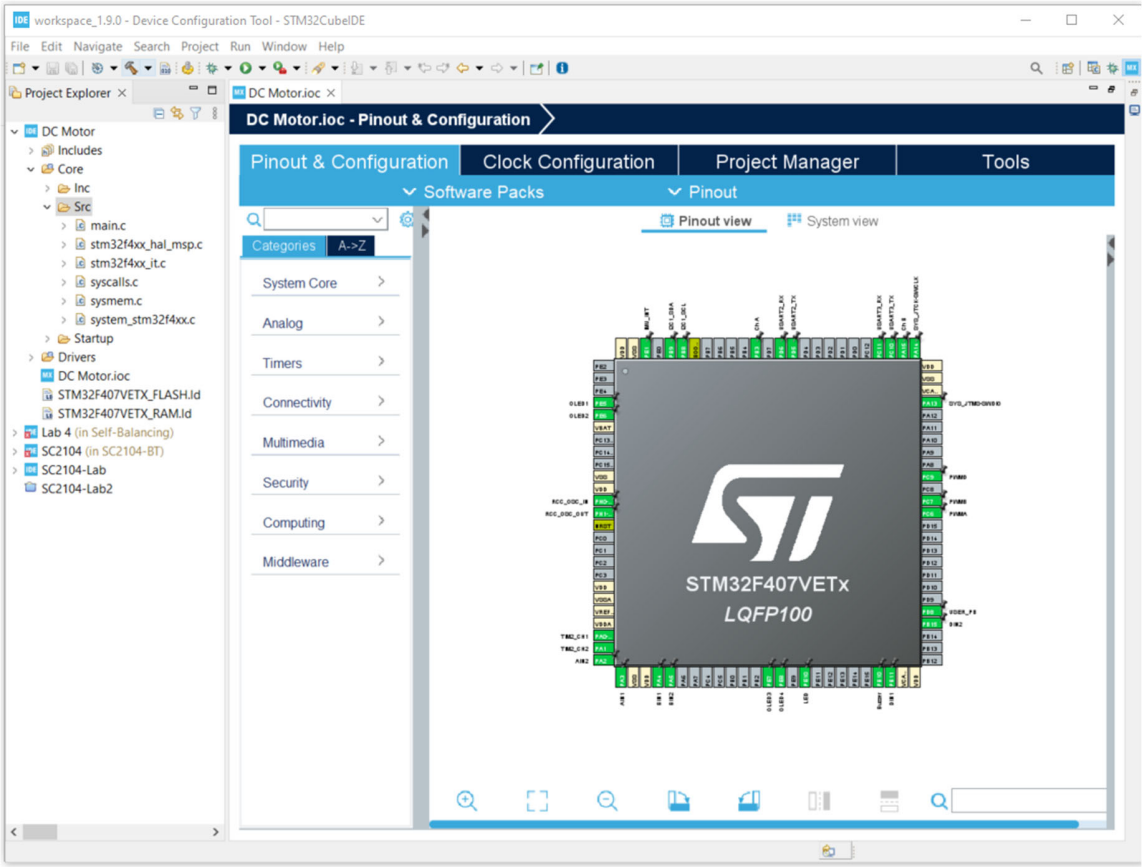
Launch the STM32CubeIDE on the computer and create a new STM32 project by choosing the option to import an existing configuration file.



- Import the “Lab 4.ioc” configuration file as shown below to setup a new project “DC Motor”.

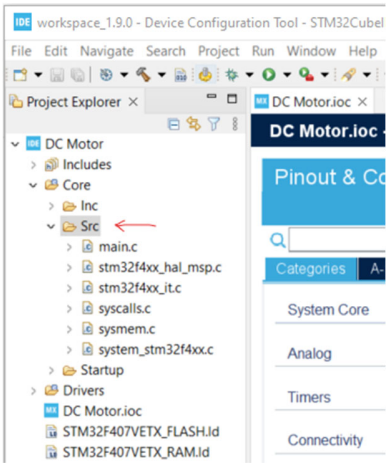


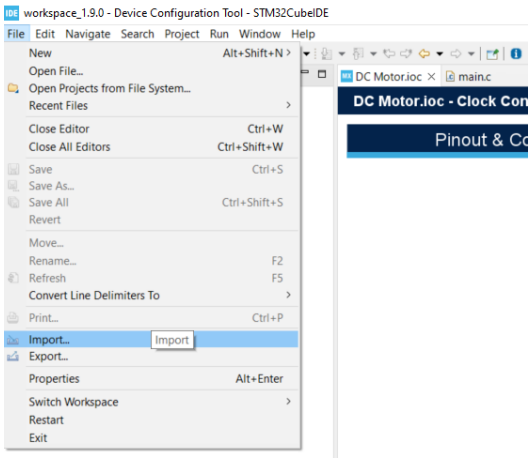
Once the configuration file has been imported and executed, the configured Microcontroller will be similar to as shown below, together with the auto generated program code in the various files (on the left panel) corresponds to the configuration.

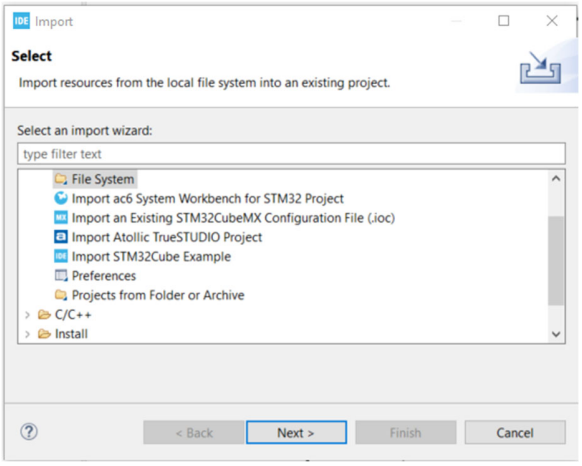


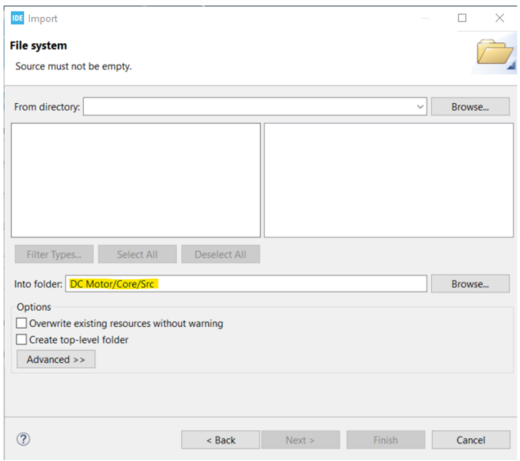
You will now import the program files extracted from the zip folder into this project.

- Select the 'Src' folder to import the given files main.c, oled.c and stm32f4xx_it.c

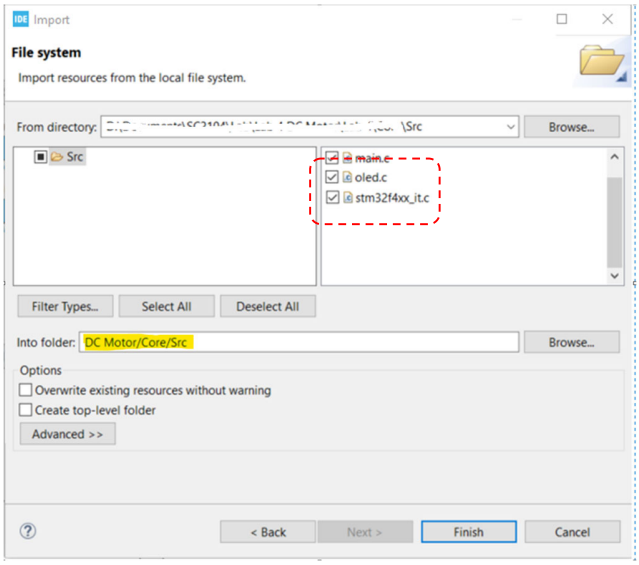




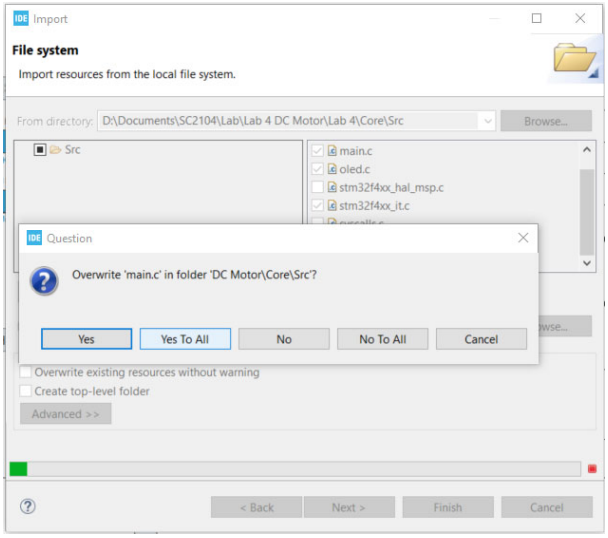




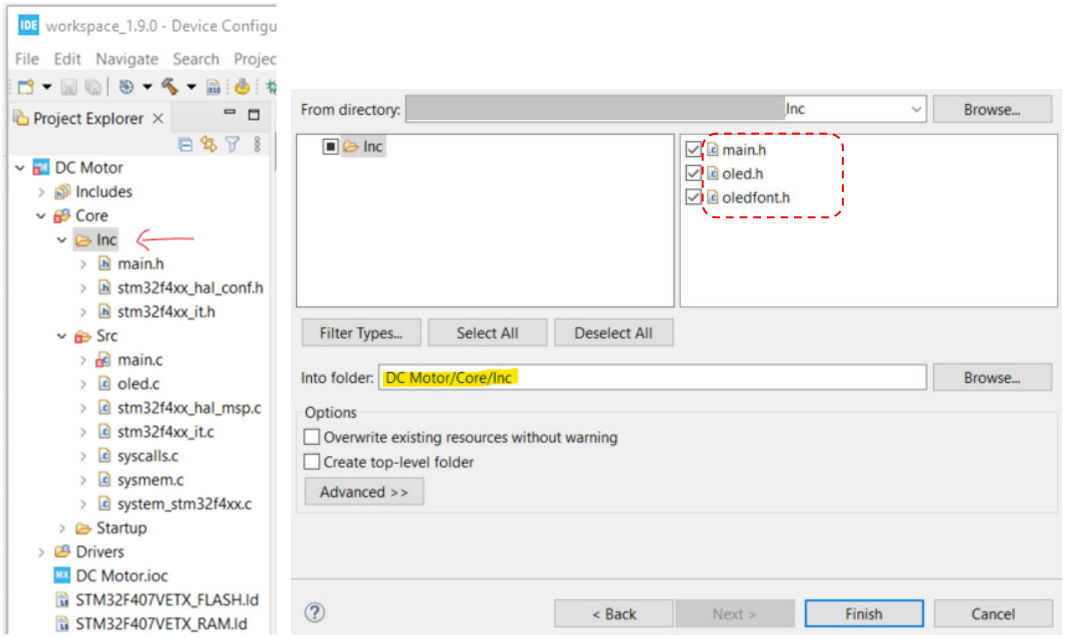
- Select the directory where the files are kept, and import them into the folder 'Src'



Note that the default main.c file is to be replaced by the provided version.



- Select the 'Inc' folder and similarly import the three header files provided as shown below.

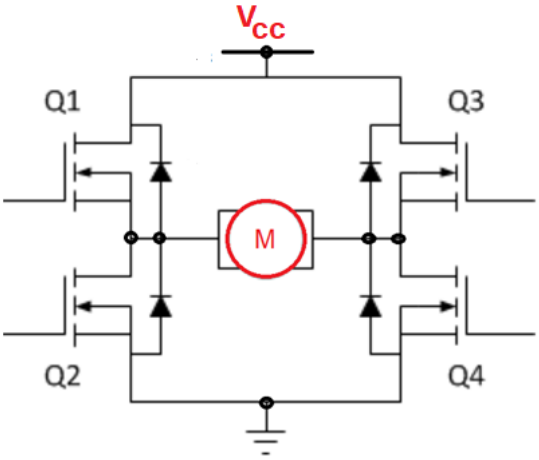


- Build/compile the project and download the code onto the STM32F4 board. (The code should be compiled with no error). The DC motor should start to rotate if the program executes successfully. (If the download is not successful, check the Run -> Debug Configurations -> Debugger setup, which should use 'ST-Link GDB server' for the Debug probe setting.)

- Connect the USB serial port (the middle USB port) to the computer and run the SerialPort program (with 115200 baud rate and using 'comma' column delimiter) to observe the data sent by the program.

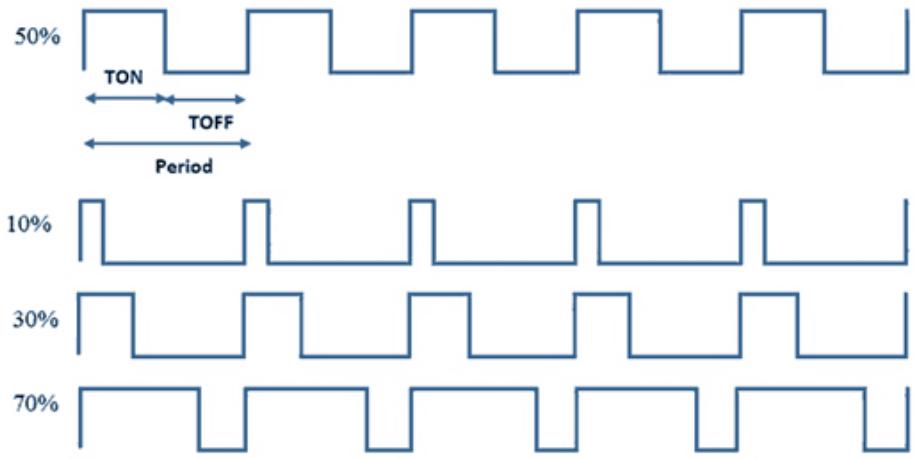
2. DC Motor Control

The operation of the DC motor will be affected by the voltage **V_{CC}** applied to the board. To observe meaningful responses with feedback control loop, V_{CC} will need to be of sufficient voltage level. In this lab exercise, you will use a 10V DC voltage (See appendix for the power supply setup.)



2.1 Operating range of the PWM value

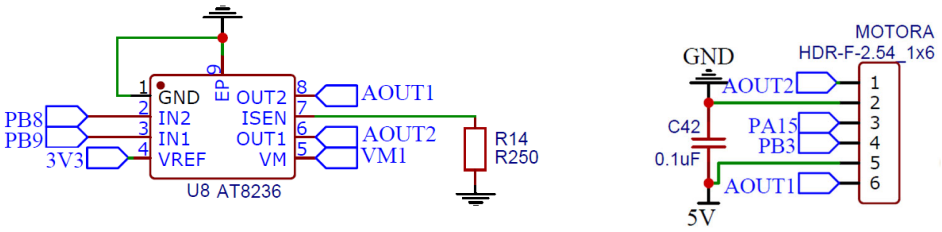
A PWM waveform is used to operate the DC motor through the H-Bridge circuit as shown in the previous diagram. By varying the duty ratio of the PWM waveform, we can control the rotation step and rotation speed of the motor. The maximum PWM value is to be limited to 7200 (i.e., its Period), but there is a minimum PWM value that is needed to make the motor starts to rotate (why?).



2.2 STM32F Board Motor Driver

The following table shows the control logic of the motor driver chip AT8236 that is used in version D of the STM32F4 board.

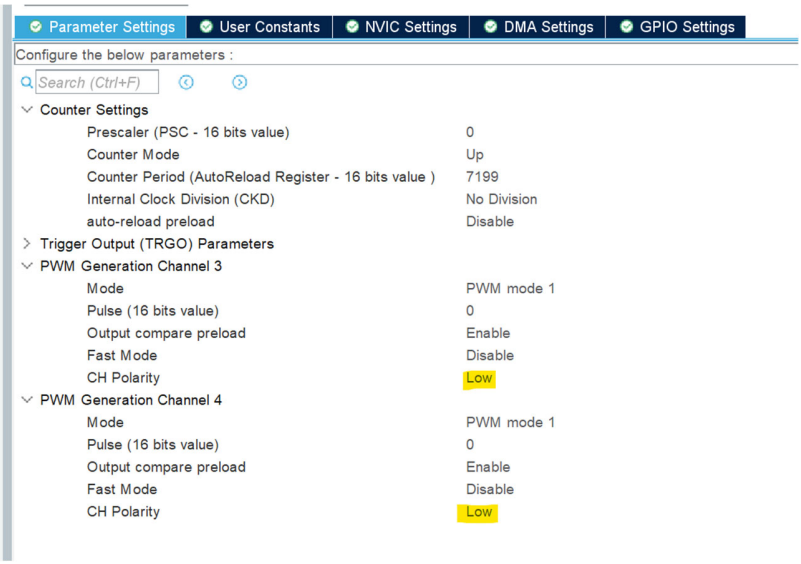
IN1	IN2	PWM Control
1	PWM	Forward/Clockwise
PWM	1	Reverse/Counter Clockwise
1	1	Brake/Stop



PB8 and PB8 pins have been configured to generate the PWM signals required to drive the H-Bridge:

PB8 = IN2 = Timer 4 Ch 3 ⇒ OUT2 = AOUT1
PB9 = IN1 = Timer 4 Ch 4 ⇒ OUT1 = AOUT2

Due to the inversion of the signal, the polarity of each channel has to be set to low, while logic '1' will require the pwm value to be set to 0 (instead of full-scale value 7199).



The DC motor utilizes a 26 poles Hall Sensor (i.e. 13 pulses) based encoder interfaced to PA15 and PB3 pins that are configured as Timer 2 'Encoder' mode. With a 20:1 gear ratio, the number of pulses generated per revolution of the output shaft is hence 260 pulses.

3. Practical Exercises

The DC Motor is to be driven by the STM32F board through Motor Driver A connector (Refer to photo on Page 1).

DC voltage applied to the STM32 board should be set at about 10V.

3.1 PWM-Speed characteristics of the DC Motor

In this part of the exercises, you will record the relationship of the PWM input vs the rotation speed of the DC motor in term of RPM (Revolution Per Minute).

i) Comment off the function call at line 441 - to skip the PID function call

```
440
441       //pwmVal = PID_Control(); // call the PID control loop calculation
442       pwmVal = 1000;           // this will overwrite PID control above
443       error = 5;               // to overwrite control loop checking
444
```

ii) Set the pwmVal at line 442 in the main.c file to a PWM values (e.g., 1000).

iii) Build and download the program onto the STM32F board that is connected to the Motor through Motor Driver A connector (See photo on Page 1).

iv) Run the program and check the RPM value, which is displayed on the OLED screen as well as been sent through the serial port (which can be displayed using the SerialPlot program – see the **`serial_uart()`** function for the detail).

```
253
254 void serial_uart(){
255       // send various values to serial port @ usart 3 for display
```

v) Change the value of the pwmVal value between 250 to 6000 and repeat the measurements from step ii) above.

Comment whether the output speed is linearly proportional to the PWM values. (E.g. plot your measurements using Excel file). This characteristic will determine the controllable range of the DC motor.

Q: What is the minimum PWM value that need to make the motor starts to rotate? (You will find that this minimum value may affect the proper operation of the motor control as will be observed later).

You can account for this by adding an offset to the pwmVal through the pwmMin parameter at line 93:

```
92 int16_t pwmMax = (7200 -200); // Maximum PWM value = 7200 keep the maximum value to 7000
93 int16_t pwmMin = 50;      // offset value to compensate for deadzone
```

3.2 Step response

In this part of the exercises, you will study and compare the effect of using various parameters to operate the control loop.

First, the program is coded with the target angle set at 1000 degrees as shown below. During operation, the motor will try to rotate to this target angle.

```
388
389   start = 0;
390   angle = 0;
391   target_angle = 1000; // rotate 1000 degree
392   error = target_angle - angle;
393   error_old = 0;
394   error_area = 0;
395
```

The angle value rotated will be shown on the OLED display. This angle value, together with other data of interest are also sent through the serial port for display on the computer using the SerialPlot program. (Again, study the **serial_uart()** function for the detail)

a) Constant PWM control loop

To start with, you will set the PWM with constant value and observe the performance of the control loop (which will stop sending the PWM signal when the rotated angle reaches the target angle)

- First, set the program statement at 441 to 443 as shown below (Study the code in the main loop to understand the purpose)

```
440
441       pwmVal = PID_Control(); // call the PID control loop calculation
442       pwmVal = 1000;          // this will overwrite PID control above
443       //error = 5;              // to overwrite control loop checking
444
```

- Set the pwmVal at line 442 to various values as shown below and observe their respective step response – i.e., the shape and the 10%-to-90% rise time taken to approach the target angle.
  i) 500
  ii) 1000
  iii) 1500
  Note: You can also try some other values to observe the behaviours of the response.

- Run the SerialPlot program to display the Motor's rotated Angle and Target angle sent by the microcontroller through the `serial_uart()` function
  (Note: Screen capture the waveforms such that you can compare the responses with different parameters settings.)

- Try turning the motor shaft by hand and observe the response of the system. This is equivalent to introducing a disturbance to the motor.

b) P, I and D control loop

In this part of the exercises, you will compare the effect of using different values of P, I and D to operate the control loop for the DC motor.

To enable the PID function for the control loop, you will need to comment off the constant pwmVal code at line 442 and 443 as shown below.

```
440
441       pwmVal = PID_Control(); // call the PID control loop calculation
442       //pwmVal = 1500;          // this will overwrite PID control above
443       //error = 5;              // to overwrite control loop checking
444
```

Program logic of Position Control Loop

The logic of the program code is as follows:

1. Perform all the necessary initializations
2. Setup interrupt to detect User push button
3. Set the target angle (e.g., 1000 degrees)
4. a. Execute the control loop
   b. Send the data to the serial port to be displayed in SerialPort program
   c. Repeat from 'a' until output angle has settled at/near the target angle
   d. Exit the control loop and stop the motor
   e. Wait for User push button to be pressed and go to 'a'

The data that are sent through the serial port are as follows (using 'comma' as column delimiter)

i) Output angle (Display this value on the SerialPlot)
ii) Target angle(Display this value on the SerialPlot)
iii) Error - difference between i) and ii)
iv) PWM value used to drive the motor (Maximum value is 7200, minimum value is ?)
v) Value of error integration - for PI control
vi) Number of times the output angle has settled within the target angle region (set to 5 to trigger 'd' above)
vii) Value of error rate - for PD control
viii) RPM of motor

The three parameters of the PID control loop are to be set through the code as shown below:

```
407
408   Kp = 1;    // range: 1 to 10
409   Ki = 0;    // range 0 to 3
410   Kd = 0;    // range: 0 to 3
411
```

i) **Proportional Gain Kp**

Set the Kp (at line 408) to various values and observe the change on the step response of the system.

- Kp = 1 (Compare this response to those observed in 3.2a)
- Kp = 2
- Kp = 3
- Kp = 4

etc.

ii) **Integral Gain Ki – PI control**

Add the integral gain Ki (at line 409) to form a PI control loop and observe its effect on the step response of the system

- Kp = 1, Ki = 0
- Kp = 1, Ki = 1
- Kp = 2, Ki = 1

etc.

iii) **Differential Gain Kd – PD control**

Add the Differential gain Kd (at line 410) to form a PD control loop and observe its effect on the step response of the system

- Kp = 4, Kd = 0
- Kp = 4, Kd = 1
- Kp = 5, Kd = 2
- Kp = 7, Kd = 2

etc.

iv) **PID control**

You can set all three control parameters at the same time to form a PID control system. Typical steps used to set the three values as follows:

1. Increase Kp until ringing at the output start to occur
2. Add Kd and increase its value to remove the ringing
3. Add Ki which will gradually adjust the output value toward the target value

Example: Compare the response of the three cases below

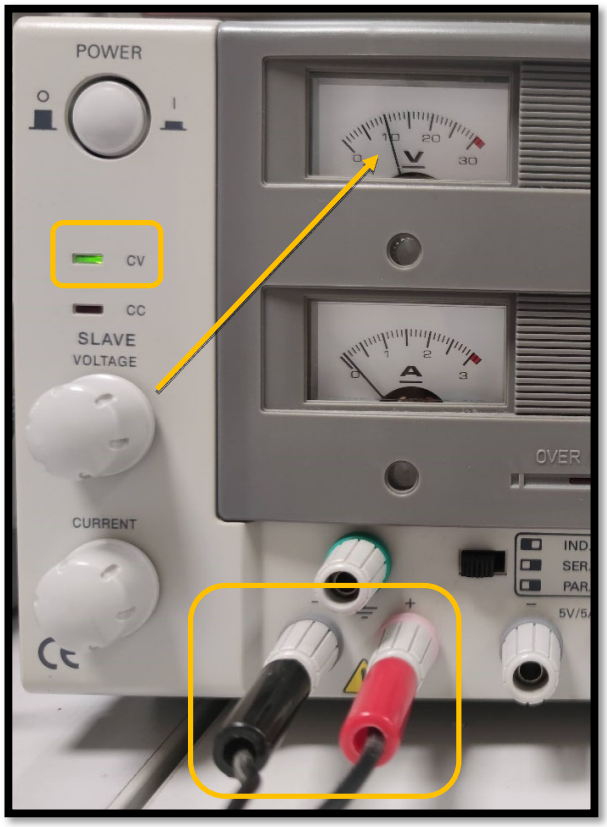
- Kp = 4
- Kp = 4 with Kd = 1
- Kp = 4 with Kd = 1 and Ki = 1

Case scenario for thought

Consider the situation that you are developing a program to control the MRT train when it approaches the station platform, such that the MRT train' doors are to be positioned correctly at the corresponding platform's barrier gates.

How will the P, I and D parameters each plays a part in such a design?

Appendix: 10V DC Power Supply setting

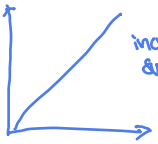


1. Adjust the “SLAVE VOLTAGE” Knob clockwise slowly to 10V. The ‘CV’ LED should remain green. (CV stands for Constant Voltage)

   Note: Do not adjust the “CURRENT” knob unless the CC LED is lighted up or you hear clicking sound during motor operation. (Then adjust it clockwise slightly until the CC LED is turned off. CC stands for Constant Current)

2. Connect the Red Lead to the +ve side of the Power Supply connector, the Black lead to the –ve side of the Power Supply connector as shown above.

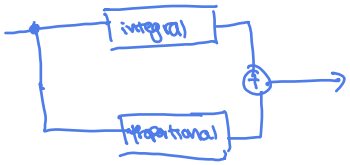
Error x Gain = Propeller Speed



increasing total sum = faster propeller / speed

Integral : Sum of Error

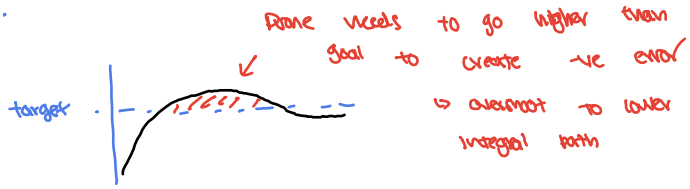
↳ Increasing Sum = increase speed



when very close to target

↳ Proportial ≈0 since error very small

consider:



Add derivative to detect how fast we are closing to our goal

↳ How fast error is growing / shrink

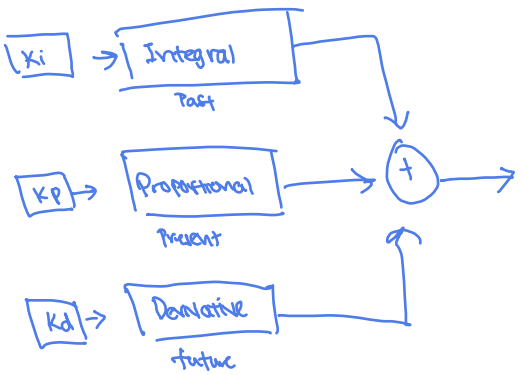
↳ Essentially, tells system that we are closing into our goal way too fast.

So, Kp → Speed

Ki → • improve steady state response
• (reduce steady state error)
• Adjusts output to target

Kd → reduce overshoot
↳ tells how fast we are approaching the target so we can slow down

↳ transient response.



Lab → 1) Kp ↑ until get ringing
2) Adjust Kd ↑ to remove ringing
3) Adjust Ki ↑ to gradually adjust.

$$error\_rate = error\_change \times \frac{1000}{dt}$$

PI

$$error\_area = error\_area + error \times dt$$

$$error = target - angle$$

$$pwmVal = error \times Kp + error\_area \times \frac{Ki}{10000}$$

→ Pushes system in the opp. direction of error.

↑ warm speed

↑ Integral of error

PD

$$pwmVal = pwmVal + error\_rate \times Kd$$

↑ Adjusts the calculated speed with Kd.

Smaller Kd = slower when approaching target

Kd slows down when approaching target.

↳ But when Kd is too high, when approaching target it will be super slow.

↳ Meanwhile, Integral is adding all this small error.

↳ When error accum. enough,