ISDN4000I 2021 Spring Lab 6

Part 1 Servo Control with PWM

This lab will have you utilize the MCU PWM peripheral to control the servo motors. As the Arduino built in PWM API (analogWrite etc.) does not support modifying of the PWM output frequency, we are going to use low level MCU register manipulation to write our own PWM servo motor driver.

This is an academic exercise to help you improve your understanding and practice of embedded systems programming. Later on in your project, you may resort to using the servo.h library for simplicity.

There are many types of registers like address register, control, flags, etc. That you have been familiarized with in the past lab. Sometimes, opting for register control is because of the need to control exact frequency or in general 20x faster than normal function call.

A servo is a device used in robotics where something has to be moved. Something, like the wheels that steer a car, the ailerons of a plane, or the fingers of a robotic hand. The control signal is a string of pulses of 50Hz; a pulse every 20ms. The servo will be at -90° if the pulse width is 1ms, at +90° if the pulse width is 2ms. and at zero, if the pulse width is 1.5ms.

TASK 1: Setting up the PWM timer

Set the PWM timer tick period to 1us, and use (REG_TCC0_PER * PWM timer tick period) to determine the period of the PWM output. Then set the frequency of the PWM to 50Hz by defining the value of the REG_TCC0_PER.

TASK 2: Creating register values for 3 servo angles

Learn more about from https://en.wikipedia.org/wiki/Servo_control.

Attach servo motors on pin 2 and pin 3.

Set the motor's servo angle to -90°g, 0° and 90° respectively, with 2 second delay in between. To do so, determine the CCBx time register value corresponds to the pulse width in microseconds (us).

TASK3: Writing setServo (int ccbCh, float angle) function

Based on your understanding of time registers implementation, write a function setServo (int ccbCh, float angle) where ccbCH is the CCB channel controlling the servo motor, and angle is the servo shaft angle in degrees.

TASK4: Test your function with the test code provided

Part 2 Closed Loop Motor Control

In the previous labs, the movement of wheels has been achieved. This time, the expected the wheel speed is controlled through encoder feedback.

PID control continuously evaluates the error between a setpoint and the variable being controlled and applies a correction based on proportional, integral, and derivative terms. The output speed of the motor from the encoder is compared to the setpoint and fed to the controller. The controller uses the PID control algorithm to determine a new output (PWM) if needed to reduce the error and a new output from the encoder starts the loop over again.

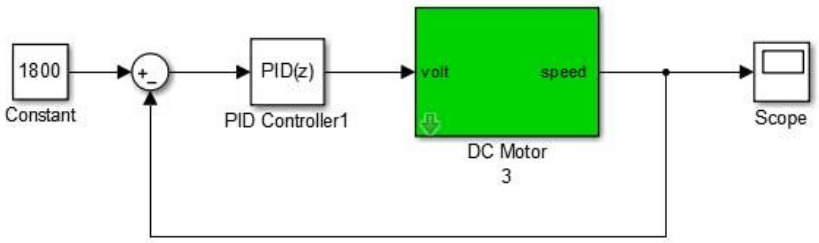
A PID controller produces an output signal consisting of three terms: one proportional to error signal, another one proportional to integral of error signal and third one proportional to the derivative of the error signal.

- The proportional controller stabilizes the gain but produces a steady state error.
- The integral controller reduces or eliminates the steady state error.
- The derivative controller reduces the rate of change of error.

PID controllers have higher stability, no offset and reduced overshoot.

$$\text{Output} = K_P e(t) + K_I \int e(t)\,dt + K_D \frac{d}{dt} e(t)$$

$$\text{Where : } e = \text{Setpoint - Input}$$



We will go through detailed discussions of PID controller next week. For the lab, use the skeleton code provided, and plot out the motor speed as it goes from rest to full speed (e.g. 30rpm) from t = 0 to t = 3 sec with PID gains applied on measured error by comparing the desired output with the measured encoder reading.

TASK 1: Show the TA plots with only P-control, another one with PI-control and discuss the 'system response' in terms of observable response time, overshoot and settling time.