LABORATORY EXERCISE 2: MOTION CONTROL

OBJECTIVES

The objectives of this exercise are:

- To test the Arduino programs (sketches) that you have written to implement closed-loop control and the control of stepper motors
- To give experience of the practical meaning of proportional and derivative control in the context of a simple motion control system

APPARATUS

- An Arduino Mega 2560 microcontroller and a prototyping board ("breadboard")
- A stand-alone servomotor incorporating incremental encoder
- An LS7366R up-down quadrature counter IC mounted on a board (the "Counter Click" board)
- Stepper motor
- A bipolar, micro-stepping stepper motor driver
- 0-35 V dual power supply
- GWINSTEK digital storage oscilloscope ("scope")

SAFETY ISSUES

In practice this is a low hazard laboratory, but it does involve the rotating shafts on two small motors. To avoid any risk of entrapment, **sleeves should be rolled up and long hair tied back**. Your personal computer is required to connect with the Arduino.

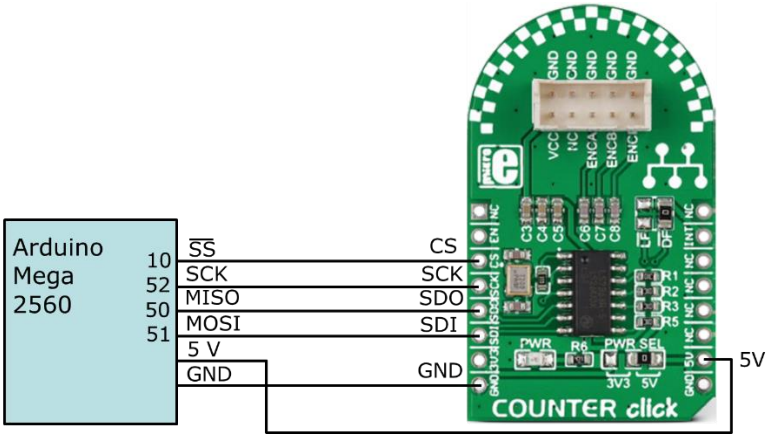
PROCEDURE

EXERCISE 1: RUNNING IN OPEN LOOP TO CHECK THE FOUNDATIONS FOR CLOSED LOOP CONTROL

This part of the laboratory is effectively a re-run of part of Lab 1 exercise 3 but is necessary to ensure all is in place for the main experiments.

1. Ensure the large bench power supply unit (PSU) is connected to the mains.
2. Switch the PSU on and ensure the ON/OFF button is in OFF. When it is OFF, the screen displays the set-point voltage and current limits.
3. Adjust the voltage to read 5V and the current limiter to read 0.5 A.
4. Turn ON the supply. The displays now read the actual voltage and current. The voltage should remain around 5V, but the current should be approximately zero as no load is connected.
5. Turn OFF the supply.
6. Plug the Counter Click quadrature counter board into the breadboard, around halfway along the breadboard.

7. Plug the encoder connection wires from the Counter Click board into separate transverse strips in the breadboard and connect the appropriate coloured wires from the motor to the Counter Click leads as follows, matching the colours:
    - Counter board **Blue** - Encoder **Blue (5V power supply)**
    - Counter board **Green** - Encoder **Green (power ground)**
    - Counter board **Yellow** - Encoder **Channel A**
    - Counter board **White** - Encoder **Channel B**
8. Connect the following (we will use the encoder output directly with the Arduino using the FSM you have developed and the individual channel timer):
    - Encoder **Channel A** – Arduino **2**
    - Encoder **Channel B** – Arduino **3**
    - Encoder **Channel A** – Arduino **47 (timer/counter input pin T5)**
9. Make the power and SPI serial communication connections between the Arduino and the Counter Click board as follows:
    - Counter **VCC (5V supply)** – Breadboard **vertical rail for 5V**
    - Counter **GND** – Breadboard **vertical rail for GND** (not the same as GND rail)
    - Counter **SCK (system clock)** – Arduino **52**
    - Counter **CS (chip select)** – Arduino **10**
    - Counter **SDI (slave data input)** – Arduino **51 (MOSI – Master Output Slave Input)**
    - Counter **SDO (slave data output)** – Arduino **50 (MISO – Master Input Slave Output)**



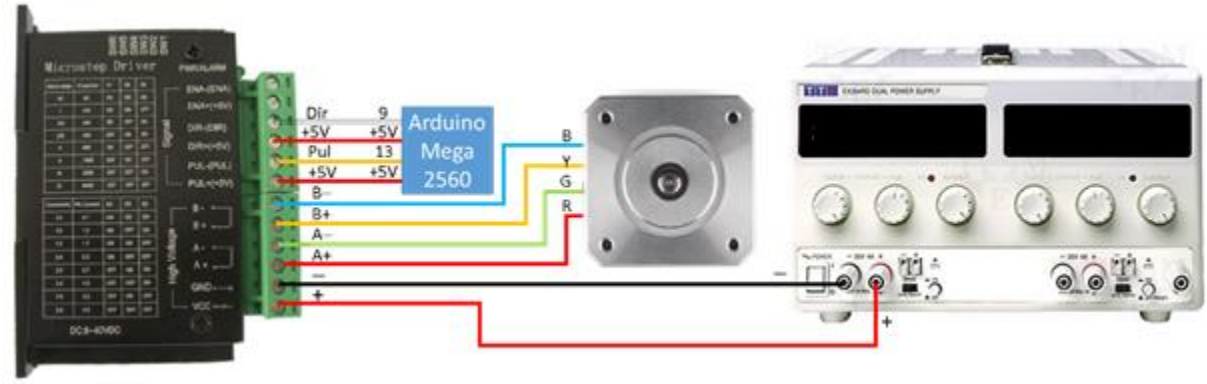
10. Leave the scope ground clip connected and connect Channel 1 scope probe to Channel A of the encoder (yellow lead) and Channel 2 scope probe to Channel B (white lead).
11. **Get a demonstrator to check your wiring to avoid expensive damage!**
12. Re-connect the Arduino.
13. Load the program you have modified from **MotorEncoderSkeleton.ino** into the Arduino IDE; it should have already been verified in your programming sessions.

EXERCISE 2: CLOSING THE LOOP: PROPORTIONAL CONTROL AND PID CONTROL

1. Now load **ProportionalClosedLoop.ino** and set `Kp` to 0.001. Compile and upload it to the Arduino. You should be able to enter a target position and watch the motor move approximately to that position.
2. What happens as you make the proportional gain `Kp` take different values in the range 0.001 to 0.1? If it oscillates wildly, de-activate the PSU, insert a smaller gain, and try again. If it runs continuously, you have probably got the "difference" calculation wrong way around.
3. De-activate the power supply and load **PIDClosedLoop.ino** Initially try `Kp=0.02, Ki=0.0` and `Kd=0.0`. Re-activate the power supply confirm that the result is the same as for your proportional gain example.

4. **Use the video guide which demonstrates how to tune a PID controller.** What happens when you add some `Ki` value?
5. Switch off the power supply, disconnect the red and black wires from the binding posts and dismantle the circuit, putting the Counter Click and other items back in the box so that you have a bare Arduino and breadboard.

EXERCISE 3: OPEN LOOP POSITION CONTROL – ONE STEP AT A TIME



1. Leave the power supply settings on 12V, 0.5A.
2. With the power supply switched off, connect the + power terminal on the stepper driver unit (confusingly labelled Vcc) to the red (+) terminal on the PSU. Similarly connect the – terminal of the driver to the – terminal of the PSU.
3. Connect the red, green, yellow and blue wires of the motor to the A+, A–, B+ and B– terminals of driver respectively (just plug in the connector).
4. Connect the terminals on the driver labelled PUL+ and DIR+ to the +5V pins on the Arduino Mega (near pin 22).
5. Using additional leads connected via the breadboard, connect the PUL– (step) terminal on the driver to pin 13 on the Arduino Mega, and connect the DIR– (direction) terminal on the driver to pin 9 on the Arduino Mega.
6. Connect the ground clip of the scope to a ground pin on the Arduino, and clip the probe for Channel 1 of the scope to an additional lead connected via the breadboard to pin 13 and Channel 2 to an additional lead connected via the breadboard to pin 9.
7. Plug the Arduino into the PC and load up **SimplisticRampStepperXXX.ino**. Comment out the "testing at home" parameters for the ramp and uncomment the "testing in lab" parameters.
8. Open the Serial Monitor, ensure it is set to NL&CR, 9600 baud.
9. Compile and upload the sketch and enter a move to 5000 steps. The ramping seems to be very slow, so will the motor manage to complete the move? Does it manage to keep up with the required acceleration?
10. Now load, compile and upload **LeibRampStepperXXX.ino**, and again comment out the home testing parameters and uncomment the lab testing parameters. Test with movements in the positive and negative directions and observe the signals on the scope. What happens to the pulse width as the motor speeds up and slows down? What happens to the pulse spacing? Does the motor keep up with the desired movements? Is something interfering with the smooth movements of your motor?
11. Switch off the power supply and unplug the Arduino from the PC, disconnect the red and black wires from the binding posts. Unplug the connectors from the stepper driver unit but do not unscrew the wires from the terminals. Remove the wires from the Arduino and put all connectors back in the box. and dismantle the circuit, putting the Counter Click and other items back in the box so that you have a bare Arduino and breadboard.

COURSEWORK SUBMISSION

Well formatted and organised submission: A zip file containing the answers to your questions, ideally as a PDF, including listings of the programs you have written and submitted as preparatory work, including any corrections made in class. Also please include in the zip file all the programs themselves as Arduino source (.ino) files. **[20%].**

Questions:

1. Explain what happens as you increase the gain in the proportional control situation. **[10%]**
2. Explain the effect of introducing derivative action to the situation with a gain of 0.02. **[10%]**
3. Explain why we are not concerned about the presence of mathematically complex operations and functions in the section of `loop()` where parameters such as *R* are set. **[10%]**
4. Explain **very briefly** why, by contrast, we have gone to considerable effort to avoid mathematically complex operations and functions in `computeNewSpeed()`. **[10%}**
5. Explain what undesirable behaviour was noticed when running the stepper example based on timed loops (hint: it was something to do with how smoothly your program operated, and something that might have distracted the microprocessor from the task of generating the pulses). Hence explain how an interrupt was used in conjunction with a hardware timer to overcome this problem. You need to identify:
    a. What triggers the interrupt
    b. What happens when the interrupt is triggered, both in terms of making something physical happen and changing an important value in a register
    c. What is the purpose of the lengthy block of if-statements in the interrupt service routine?

    [15% for telling the story here]
6. What happens when you try to run the stepper motor too fast? **[5%]**
7. Explain how stepper motors and servo motors use very different approaches to achieving motion control. **[20%]**