Lab 4 - Encoders

ME 451 - Introduction to Instrumentation and Measurement Systems, Spring 2019

## Lab Objectives

* Use quadrature encoders to measure rotation.
* Drive a motor with a motor driver.
* Gain more experience with interrupts.
* Get exposed to bang-bang setpoint control method.
* Gain more independence with Arduino coding.

**Lab Sensor for Report:** Encoders. *This is a 3 day lab.*

*Note:* Groups will use motors that belong to the class for this lab. These motors will be kept in the lab room at all times. If a lab group breaks a motor, they will have to borrow motors from other groups.

# Section 1: Encoders and Interrupts

1. Wire up the encoder based on the documentation on the [product page](https://www.pololu.com/product/2824).
   1. *Note:* We have bought several different motor/encoder combinations and therefore your readings can be different from other groups. Although your specifications might be different from the product page, you can trust the wiring diagram shown in the ‘using the encoder’ section.
      1. Do not connect the motor power wires yet, that will go to the motor driver board in your kit. You will set up the motor in Section 3.
      2. You must hook up encoder channel A to pin 2.
   2. *Note:* Make sure that you have a view of the actual encoder. If there is a black cover on the back, carefully remove it from the motor assembly.
2. Encoders work in principle similarly to a button. The way we read encoders is with interrupts: we wait for it to trigger (we will call them ticks) and count how many ticks we experience. From there, we can deduce how much the encoder was rotated.
   1. Coding an interrupt for an encoder is straightforward: Use your code for *Lab 3 Signoff 3*. Encoder sensing works a lot like button counting, but without the need for debouncing.
      1. *Note:* Make sure that in your pinMode line you use INPUT instead of INPUT\_PULLUP.
      2. *Note:* Do not put any Serial.print statements into your interrupt, it will mess up your readings.
3. Based on how the code runs, your code will count up regardless of what way you rotate the encoder.
   1. The encoder we are using in lab is a quadrature encoder. Each encoder has two channels built into the sensor which we use to determine direction.
      1. Each encoder channel outputs high or low. \*\*\*
   2. For lab, implement a RISING interrupt to sense the encoder state of Channel A. Inside the interrupt, check the state of Channel B. Add a tick to the counter variable if going forwards, and subtract a tick if going backwards.
   3. **Signoff 1:** Demonstrate your direction sensing encoder implementation.
      1. Output your tick count to the Serial Monitor. It should count up when spun clockwise, count down when spun counterclockwise.
      2. *Note:* Many examples online will implement reading an encoder and sensing direction in the loop() function, however we do not want you to do this for lab. You will not get credit for Signoff 1 if you implement encoder reading in the loop() function.

## Section 1 Discussion Questions

**Discussion Question 1:** Take a picture of the encoder sensor on the back of the motor (underneath the cap). Annotate the picture for the encoder wheel and both hall effect sensors. What does a hall effect sensor sense?

**DQ 2:** How does a quadrature encoder work? How much out of phase is each channel in the quadrature encoder? Describe the advantages of reading the encoders with interrupts, as opposed to in the loop?

# Section 2: Making Encoders Useful

1. Reading the encoder is only half the battle. Now we will take our tick count and get some more useful data.
2. Something important to know about an encoder is how many counts per revolution there are. Unfortunately, our suppliers sent us various motors with encoders that have different counts per revolution. This means we will have to find it manually.
   1. Rotate the motor shaft one spin. Be as accurate as you can when measuring the ticks.
      1. Repeat this five times. Record the average ticks of all of your trials.
   2. Now spin the encoder wheel on the back, not the motor shaft, and count the ticks per rotation. Record the average number of ticks after five trials.
   3. Using the counts per revolution, we should be able to determine the angle of the motor shaft (from our initial starting position).
      1. Do this now. At 360 degrees, reset the rotation to 0 degrees.
      2. Output your calculated degrees alongside your tick count variable in the Serial Monitor.
   4. We can also use the counts per revolution to determine distance travelled.
      1. Do this now. Assume that we have a wheel attached to the motor that has a diameter of 9/π cm.
3. Finally, we will calculate the velocity at which your motor is moving, based on your encoder.
   1. We will need to calculate the instantaneous velocity.
      1. To keep the interrupt simple, we will implement instantaneous velocity calculations in the loop() function.
      2. To do that, we will need to find the displacement and dt of the system.
         1. The easiest way to do this is to set our dt with a delay at the end of loop. Then we will calculate our net displacement by comparing the current angle with the angle that we were previously at, the last time we checked.
            1. Use a value of 100 milliseconds for dt.
         2. Then calculate displacement over our dt also in loop, and thus calculate the instantaneous velocity. Output the velocity calculations to Serial Monitor in csv format, alongside your other calculations.
4. **Signoff 2:** Working encoder system.
   1. Outputs the number of ticks, the motor shaft angle, the distance travelled, and velocity calculations to the Serial Monitor.

## Section 2 Discussion Questions

**DQ 3:** Using your ticks per revolution at the motor shaft and at the encoder wheel, calculate what the gear ratio of your motor is. Compare it to the encoder product page gear ratio. (*Note:* Not all of the motors will match the product page).

# Section 3: Running the Motor

1. Before we put together the motor driver circuit, connect your motor power wires to 5V (red) and GND (black).
   1. Record the velocity that the motor runs at.
2. It’s now time to control the motor. Following your findings from pre lab, set up your motor driver to control a DC motor.
   1. Implement analogWrite to control your motor on the control pin of the motor driver. For now, just have it continuously output one analogWrite speed.
   2. Plug in the 12V power supply into the motor driver (use the DC connector to help). Keep the motor driver powered by 12V for the rest of the lab.
      1. You can control the speed of the motor by putting a PWM signal to the EN1 pin. The IN1 and IN2 pins are used for direction setting.
         1. *Note:* **DO NOT** let both IN1 and IN2 be high at the same time.
   3. Write code to run the motor until it reaches 1000 encoder ticks, then run the motor in the other direction until it reaches 0. Then switch the directions again and repeat.
      1. Add a timestamp to your Serial Monitor. Record two cycles of the direction switching.
      2. Plot the data you collected. You should expect the tick values to overshoot a little over 1000 and a little under 0 ticks.
      3. Make a tweak to your code which will eliminate the overshoot in the tick value. You are free to do whatever you want to the code to do this. For those unsure of what to do, we offer this suggestion:
         1. As you get closer to the tick value, lower the speed. You can do this simply or elegantly, your choice.
         2. Your system must have zero offshoot. Regarding undershoot, your system can only be 10 off from the setpoint (so from 990 to 1000 or 0 to 10).
         3. *Note:* Your freedom has one exception: Lab 5 is about PID, so you are not allowed to use PID control for this section.
      4. Record the data with your implementation that eliminates the overshoot.
         1. **EXTRA CREDIT:** The group (or groups) which have the fastest direction switching time will receive 10 points extra credit on their lab report.
            1. This will be determined when the labs are graded.
            2. Remember, you must plot two cycles of direction switching for it to count.
      5. **Signoff 3:** Show off your system.
         1. Demonstrate your zero-offshoot system.
3. Motor speed characterization.
   1. We will also determine the relationship between the motor speed variable and the actual no-load motor velocity.
      1. Use your velocity code from Section 2, but modify it so that it will drive the motor forwards continuously with analogWrite.
      2. Record the average motor velocity at the motor speed variable values of 0, 25, 50, 75, 100, 125, 150, 175, 200, 225, 255. Plot the response in MATLAB.

## Section 3 Discussion Questions

**DQ 4:** Describe how you removed the overshoot from your code. Provide both plots you made: the initial plot and the improved plot. How fast were both trials?

*Note:* If you wish to be considered for the Extra Credit, zoom in at the crest of the peaks and show that there is no overshoot and it is within the bounds we give in the directions.

**DQ 5:** Provide the plot of your motor characterization. What is the lowest analogWrite value that makes the motor move? Why is it like this? What is the no-load speed of the motor?

# Section 4: Preparation for PID

1. PID is a control method to bring a system nicely to a setpoint. In preparation for PID, we will try to build a system that goes to a setpoint without PID.
   1. The most basic control strategy to get to a setpoint is to tell the motor to go full speed in the direction towards the setpoint.
   2. Implement this control strategy.
      1. Make the setpoint 1000 ticks.
         1. *Note:* Make sure to always drive the motor towards the setpoint depending on what side of the setpoint your are on.
      2. Count ticks using your quadrature encoder code.
      3. Print a timestamp and the number of ticks to your Serial Monitor.
      4. *Note:* You should not expect this to work that well. There is only a low chance that it will settle.
   3. **Signoff 4:** Demonstrate a working, bang-bang system.
      1. Demonstrate your Arduino trying to get to 1000.
      2. Your system does not need to work well or even settle. Just show it working.
   4. Record your implementation trying to get to the setpoint for 15 seconds.
      1. Plot this response in MATLAB.
         1. Also plot the setpoint on this graph as a solid line.
2. Just like the end of Section 3, let’s edit the code to make the Arduino settle near 1000 +/- 50.
   1. Record your implementation and show how it settles near the setpoint of 1000 ticks.
   2. *Note:* The advice from Section 3 also applies here.
   3. *Note:* Since Lab 5 is about PID (and since this is prep for PID), you are not allowed to use PID here. You must use some sort of tweaked bang-bang control.
   4. *Note:* No extra credit for this implementation.

## Section 4 Discussion Questions

**DQ 6:** Describe how a standard bang-bang implementation works, and then describe the tweaks you made to it to make it settle. Provide both plots you made: the initial plot and the improved plot.

# Post-Lab Questions

**Post-lab Question 1:** Come up with a test to generate a torque-speed curve. Why do we care about the torque-speed curve of a motor?

**PLQ 2:** An alternative to an encoder is a rotary potentiometer. What are the advantages that encoders have over rotary potentiometers?

**PLQ 3:** Let’s say we have a flag on our motor just like in lab.

* 1. Are the encoders able to tell you the exact position of the flag?
     1. If not, what do you need to know before you can figure that out?
  2. What if the Arduino were powered off and powered on again?
     1. Again, if not, what do you need to know before you can figure that out?
  3. What type of encoder can give you an absolute position? How do they work?

**PLQ 4:** What is odometry? How can encoders be used for it? Only using encoder odometry is not good practice. Why? Briefly describe what SLAM is (no need to get too technical, just describe as simply as possible what it is used for) and how it improves upon encoder odometry.

### 

### Group Names:

(in pen)

# Lab 4 Signoffs

1. \_\_\_\_\_\_\_ Implemented quadrature encoder reading.
2. \_\_\_\_\_\_\_ Demonstrated a working encoder system with angle, distance, and velocity calculations.
3. \_\_\_\_\_\_\_ Implemented zero offshoot, direction switching system.
4. \_\_\_\_\_\_\_ Showed working bang-bang setpoint control.

**TA Signature**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Date**:\_\_\_\_\_\_\_\_\_\_\_

### 