Lab 9: Rotational Sensors and Trajectory Following

1. Introduction

In this lab, students will learn to use rotational sensors, specifically rotary encoders, to measure angular position. Students will also learn to use numerical differentiation to estimate velocity from the position measurements. Finally, students will learn how to use position feedback to make a motor follow a desired trajectory of position vs. time.

1.1 Lab Objectives

- Learn to use quadrature encoders
- Implement numerical differentiation
- Implement basic feedback control to follow a desired trajectory

1.2 Project Objectives

- Implement encoder wheel odometry and trajectory following
- Finalize sensing and control strategy

1.3 Lab Hardware

- Lab Robot

1.4 Project hardware

- Magnetic encoders on Pololu motors

2. Laboratory Concepts

2.1 Rotary Encoders

A rotary encoder is a type of sensor that encodes angular position information (either relative or absolute) onto a rotating disk using optical or magnetic stripes. In the case of an optical encoder, the disk contains of one or more tracks of alternating black and white/transparent stripes, and one or more pairs of LEDs/Photodetectors to measure the light reflected from or transmitted through the disk. In the case of a magnetic encoder, the disk contains a track of alternating magnetic stripes (north/south poles) and one or more Hall Effect sensors. The resolution of the encoder (smallest increment of angular position that can be detected) will be inversely proportional to the number of stripes.

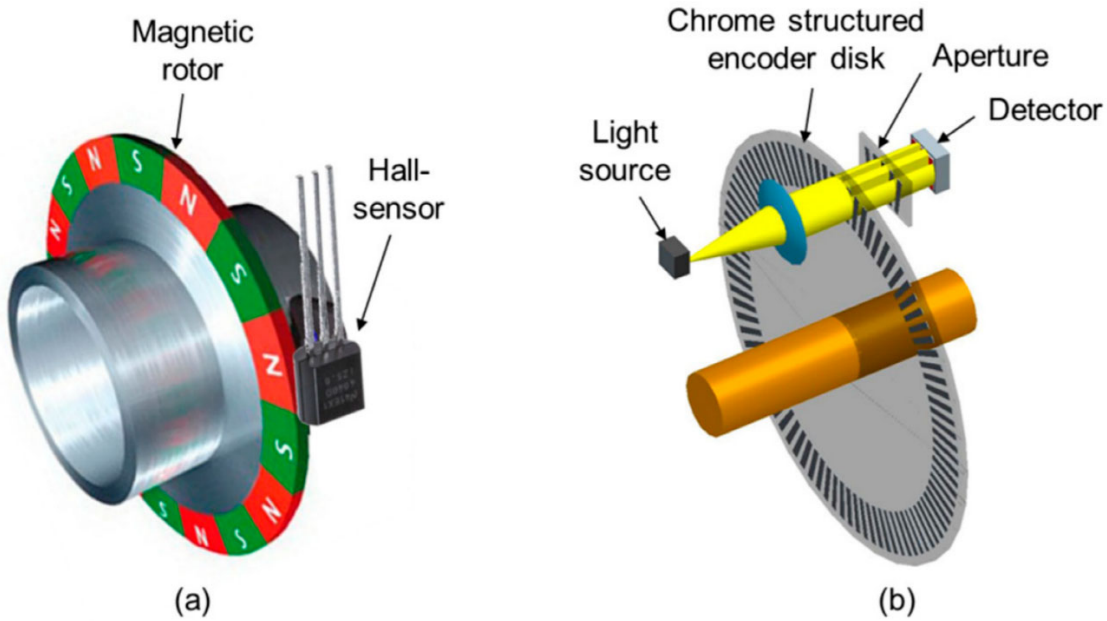


Figure 1. Diagram of Magnetic vs Optical encoders.

An encoder with a single detector would output a digital signal that looks like a square wave that jumps between HIGH and LOW as the stripes go by the detector. By counting pulses of the square wave, the relative or incremental angular position of the encoder can be tracked. However, in order to know whether to count up or down, there needs to be a way to detect which direction the disk is rotating. In order to accomplish this, a second detector is typically offset from the first detector, such that there is a 90° phase shift between the square wave from the two detectors, as illustrated in Figure 2. *Quadrature Signal from a 2-Channel Encoder*. Depending on whether the encoder is rotating clockwise (CW) or counterclockwise (CCW), the Channel 1/A signal will be leading or lagging behind the Channel 2/B signal by a quarter cycle as shown in Figure 3. By **decoding** these two signals, we can count up/down depending on whether the encoder is rotating CW or CCW.

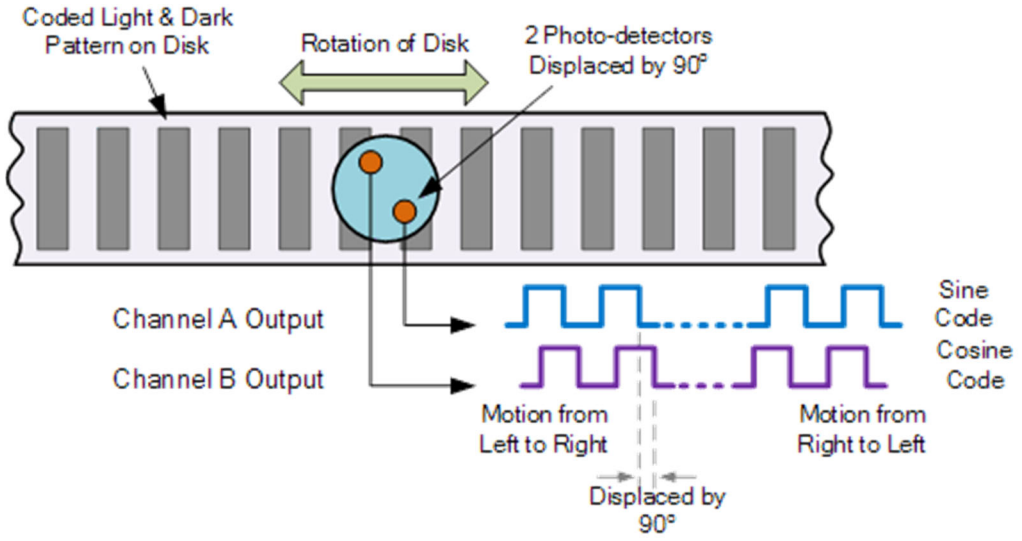


Figure 2. Quadrature Signal from a 2-Channel Encoder

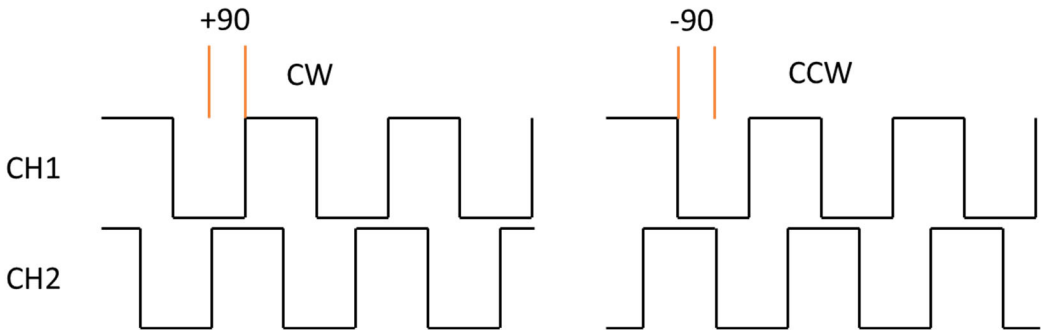


Figure 3. Example output of encoders going clockwise (CW) or counter-clockwise (CCW).

The two signals output by the encoder are referred to collectively as the **Quadrature** signals, since there are now 4 distinct countable events (square wave rises/falls) for each stripe that goes by. This can be exploited to effectively improve the resolution of the position measurements. As shown in Figure 4, a nominal strategy (X1 decoding) would be to just count the rising edges of Channel A, and only use Channel B to decide which direction to count. By counting both the rising and falling edges of Channel A (X2 decoding), the resolution can be improved by a factor of 2. However by counting the rising and falling edges of both channels (X4 decoding), the resolution can be improved by a factor of 4.

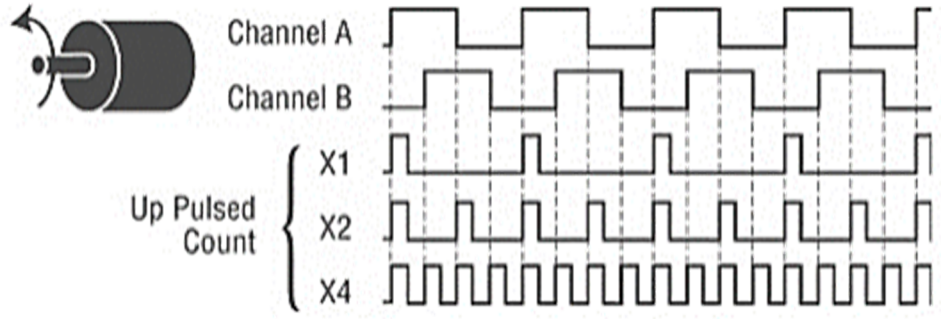


Figure 4. Examples of quadrature decoding.

As an example, consider the Pololu 37 mm diameter gearmotors used in this lab (see Figure 5), which come with magnetic encoders already installed. These encoders have 16 pairs of magnetic poles, so using full X4 quadrature decoding produces 64 counts per revolution, resulting in a resolution of 360°/64 = 5.6° as the smallest increment of angle that can be measured. This may not seem very impressive, but consider the effect of the gearbox with gear ratio of 1:N. The encoder is attached to the motor shaft, which will make N revolutions for each revolution of the output shaft. Therefore the encoder will produce Nx64 counts per revolution of the output shaft. For the motors in this lab the gear ratio N=70, which means the encoders will produce 4480 counts per revolution, which equates to a resolution of 360°/4480 = 0.08°, which is quite impressive. This means if we attach a wheel or arm to the output shaft, we can measure (and control) its angular position to within a tenth of a degree.

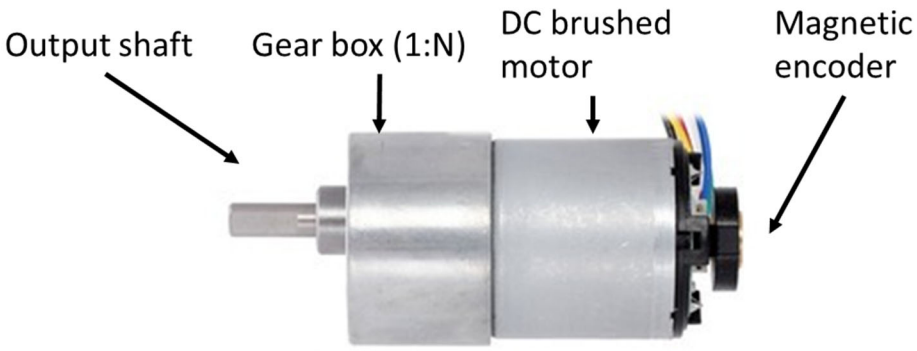


Figure 5. Pololu 37 mm diameter gear motor.

2.2 Quadrature Decoding/Counting

In general, there are two options for decoding/counting the quadrature signals from an encoder: 1. Software decoding/counting and 2. Hardware decoding/counting.

Software Decoding/Counting

To perform software decoding/counting, one connects the two encoder output channels to a pair of digital input pins (at least one of which are interrupt capable, preferably both) on the Arduino. For the Arduino Mega, the dedicated interrupt pins are: 2, 3, 18, 19, 20, and 21. Interrupts are special user-defined functions on the Arduino that are tied to a specific digital pin, such that when that pin changes states (HIGH to LOW or vice versa), the function is immediately and automatically executed, interrupting whatever else the program is doing at the time. When defining the interrupt function, one can put appropriate code inside, such that whenever the function executes (meaning a quadrature event occurred), it checks the states of both quadrature signals and increments the count up or down appropriately. Fortunately, an Arduino library named *Encoder* has been developed by the company PJRC (www.pjrc.com/teensy/td_libs_Encoder.html), which handles all the programming of the interrupts for you. To use the library, you must first create an instance of the Encoder object, specifying which pins to use. and then you can simply use the *.read()* function at any time to find the current count:

```
#include "Encoder.h"                          //includes library
Encoder myEnc(interupt_pin_1, interupt_pin_2); //Create an encoder object
current_count = myEnc.read();                 //Read the current counts
```

It will be up to you the user to convert the count into an angular position in radians or degrees based on the resolution of the encoder and the gear ratio. Note that this will be a **relative angular position**, relative to whatever position your wheel or mechanism was when you first turned on your Arduino. Therefore, you may want to use the *.write(0)* command to reset the count to zero after homing your mechanism or starting a new trajectory for your wheel.

While software counting with interrupts is easy to implement, there are limitations to be aware of. If the encoder is spinning at high speed, and the number of counts/second is sufficiently large, your programming could be overwhelmed by the frequency at which the interrupt functions are called, bogging down the execution of your main programming. We don't normally experience this limitation in this class, but it is something to be aware of.

Hardware Decoding/Counting

The alternative is to use hardware decoding/counting, which takes the burden off the Arduino programming. For example, one can use a LS7366 counter chip to directly interface with the quadrature signals and do all the decoding/counting offboard the Arduino. The LS7366 would then interface to the Arduino via the SPI (Serial Peripheral Interface), which are pins 50-53 on the Arduino Mega. One could put the LS7366 chip(s) on a breadboard, or purchase an Arduino shield-board such as the Robogaia, which has LS7366 chips for three encoders (see Figure 6). The Arduino program can then query the LS7366 chip to obtain the count whenever needed.

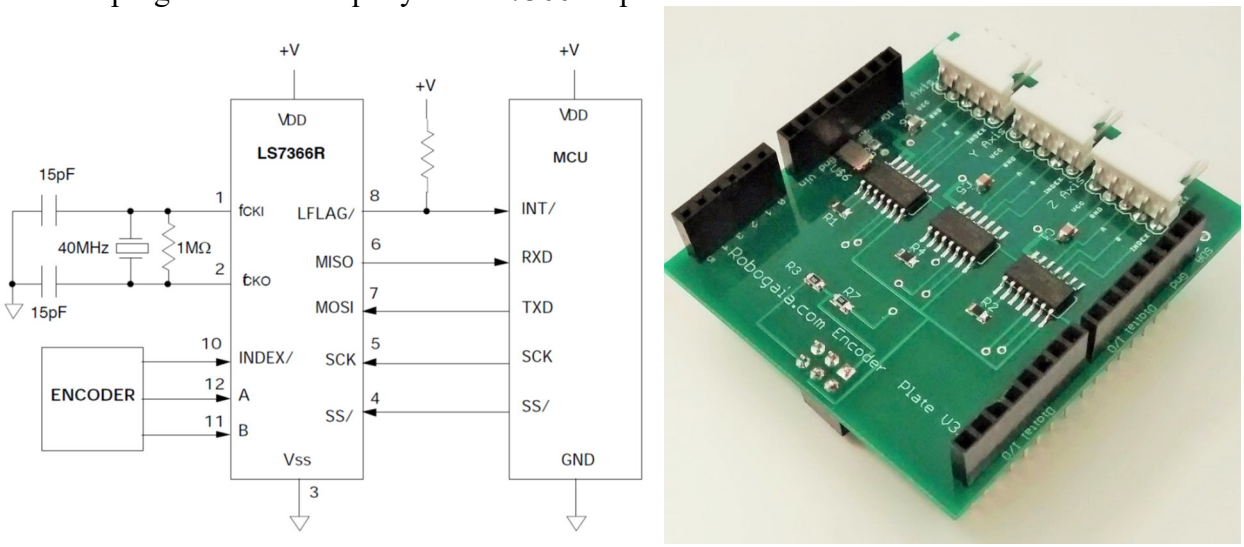


Figure 6. LS7366 Quadrature Counter Chip and Robogaia Shield Board

2.3 Velocity Measurement using Numerical Differentiation

It is often desirable to measure angular velocity in addition to angular position. There are sensors (tachometers) that directly measure velocity. However in the case where we already have a position sensor, an easier solution is to simply differentiate the angular position measurement. If the position sensor were an analog sensor (e.g. a potentiometer), we could use an analog circuit to differentiate the signal (i.e. differentiator op-amp circuit) before sampling, or we could compute the derivative using numerical/discrete differentiation via software. In the case where we are using an encoder to digitally measure position, then computing the derivative via software is the only

choice. A very simple way is to compute the derivative of a signal is to numerically estimate the slope of the signal. This method is shown in the figure and equation below.

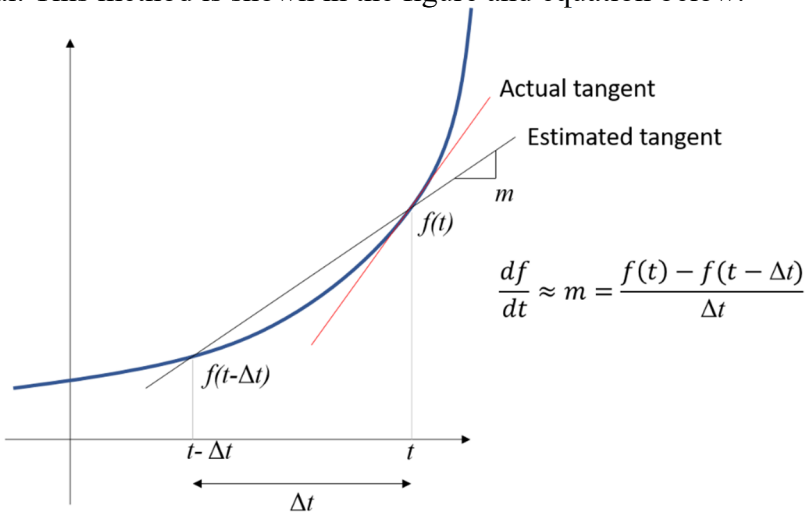


Figure 7. Numerical differentiation example.

The actual derivative of the function *f(t)* is the slope of the tangent line. One way to estimate this line is to find the slope of the line between the current point in time and the previous point in time *f(t-Δt)*. The smaller the time step Δ*t* (i.e the faster we sample the position), the more accurate this numerical approximation of the velocity becomes. However, one should be aware that the numerical differentiation process amplifies the noise in a signal. In the case of encoders, we don't have any noise in the traditional sense, however the resolution of the encoders produces a stair-step effect that is referred to as *Quantization Noise*. When taking the derivative, a step in position results in a spike in velocity. Therefore, it is often a good idea to use digital filtering to smooth the encoder signal before or after taking the derivative.

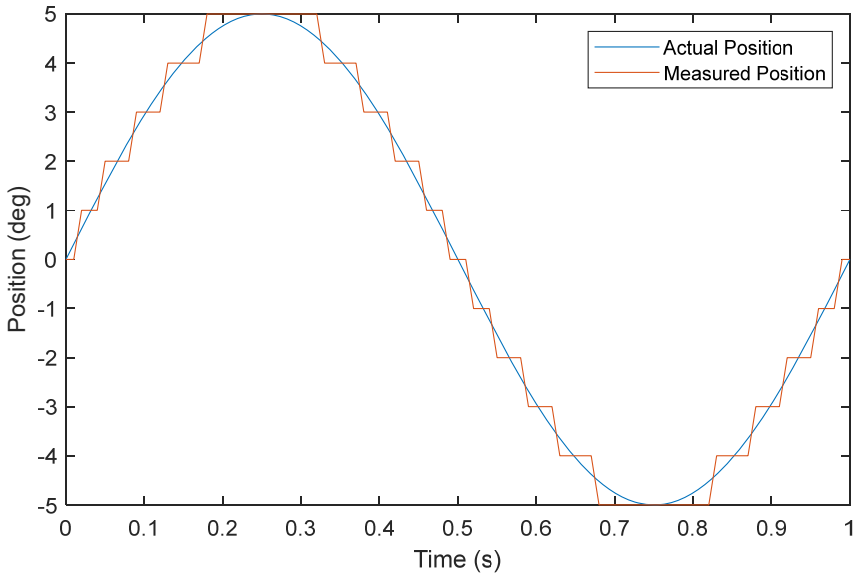


Figure 8. Encoder Reading with "Quantization Noise"

2.4 Trajectory Following

The final objective of this lab is to use the position measurements from an encoder to make a wheel follow a desired trajectory of angular position vs. time.

Proportional Control

Suppose you have an angular position reading θ from an encoder on your wheel/motor, and a desired angle θ_d that you want to position the wheel/motor at, then you can use a simple proportional feedback control law to command the motor voltage V_m to be proportional to the error in angular position:

$$V_m = K_P(\theta_d - \theta)$$

where K_p is a proportional gain factor that you can experimentally tune. Note that in order to get a stable controller, you need to be careful that 1. a positive motor voltage V_m results in your wheels turning forward, and 2. when your wheels turn forward, your encoders read a positive rotation θ. If a positive motor voltage results in your wheel turning backward, you need to reverse the leads on your motor, or add a negative sign to your motor command in your code. If your encoder reads a negative angle θ when the wheel turns forward, you need to reverse the quadrature leads from your encoder, or add a negative sign in your code when reading your encoder angle θ. When you implement your control law, start with a small K_P value and work your way up. The position tracking will improve as you increase the gain, but if you increase it too much, the control will exhibit unstable behavior.

To get smooth trajectory following, you want to read your current position and update your desired wheel position and motor voltage as frequently as possible. On the Arduino, you can keep track of your sample time ΔT (the time it takes for your sampling loop to repeat) using the *millis()* function.

Trajectory Design

Rather than commanding your wheel to instantly go to a certain desired position, we want to design a ramp trajectory such that the desired position of the wheel gradually ramps up at some desired rotational velocity ω_d. You can numerically integrate to compute the desired wheel angle θ_d at each time step, and **stop integrating when θ_d reaches your final desired wheel angle θ_f.**

$$\theta_d = \theta_d + \omega_d \Delta T$$

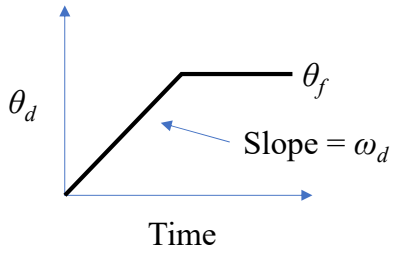


Figure 9. Desired Ramp Trajectory

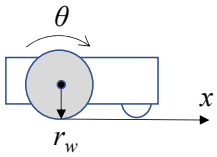
For a robot with two wheels, you need to implement your proportional control law and desired trajectory independently for each wheel:

Wheel 1: $V_{m1} = K_P(\theta_{d1} - \theta_1)$ $\theta_{d1} = \theta_{d1} + \omega_{d1}\Delta T$

Wheel 2: $V_{m2} = K_P(\theta_{d2} - \theta_2)$ $\theta_{d2} = \theta_{d2} + \omega_{d2}\Delta T$

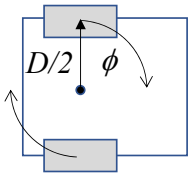
By coordinating the desired wheel angles for both wheels, you can make the robot execute a variety of maneuvers such as driving in a straight line, turning in place, or driving in a circle.

1. **To drive in a straight line**, use the same desired velocity ω_d for both wheels. You can compute the final desired wheel angle θ_f (in radians) for both wheels according to the final distance you want to travel x_f and the wheel radius r_w.



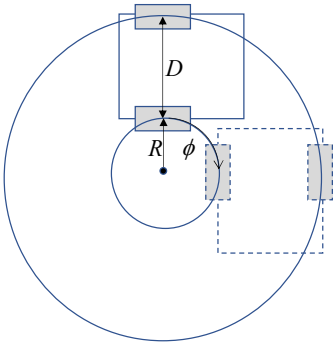
$$\theta_f = \frac{x_f}{r_w}$$

2. **To turn in place** (about the center of your drive wheels), use the same magnitude but opposite signs for the ω_d of each wheel. In this case, the distance (arc length) each wheel needs to travel on the floor is equal to the yaw angle ϕ (in radians) you want your robot to rotate multiplied by D/2 (half the distance between your wheels), so the final desired angles for each wheel are given by:



$$wheel\ 1:\ \theta_f = \frac{\emptyset\, D/2}{r_w} \qquad wheel\ 2:\ \theta_f = -\frac{\emptyset\, D/2}{r_w}$$

3. **To drive in a circle of radius *R***, we need to compute the ratio of velocities for the right and left wheels. If D is the distance between the wheels, and the inner wheel is following an arc with radius R, then the outer wheel is following an arc with radius $R+D$. Therefore the ratio of velocities is given by:



$$\frac{\omega_{outer}}{\omega_{inner}} = \frac{R + D}{R}$$

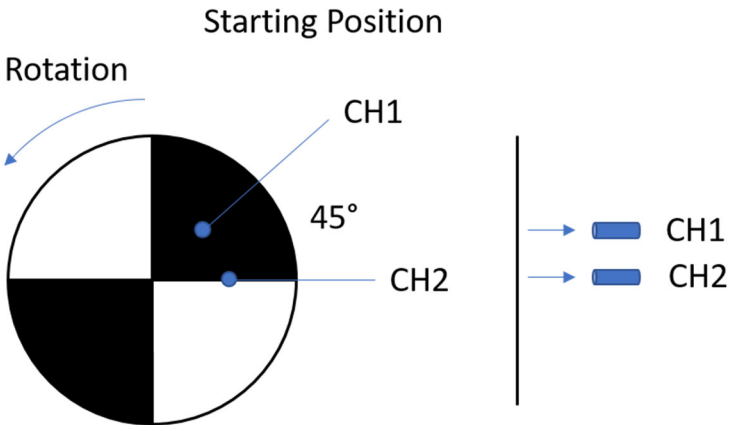
and the final desired wheel angles will be:

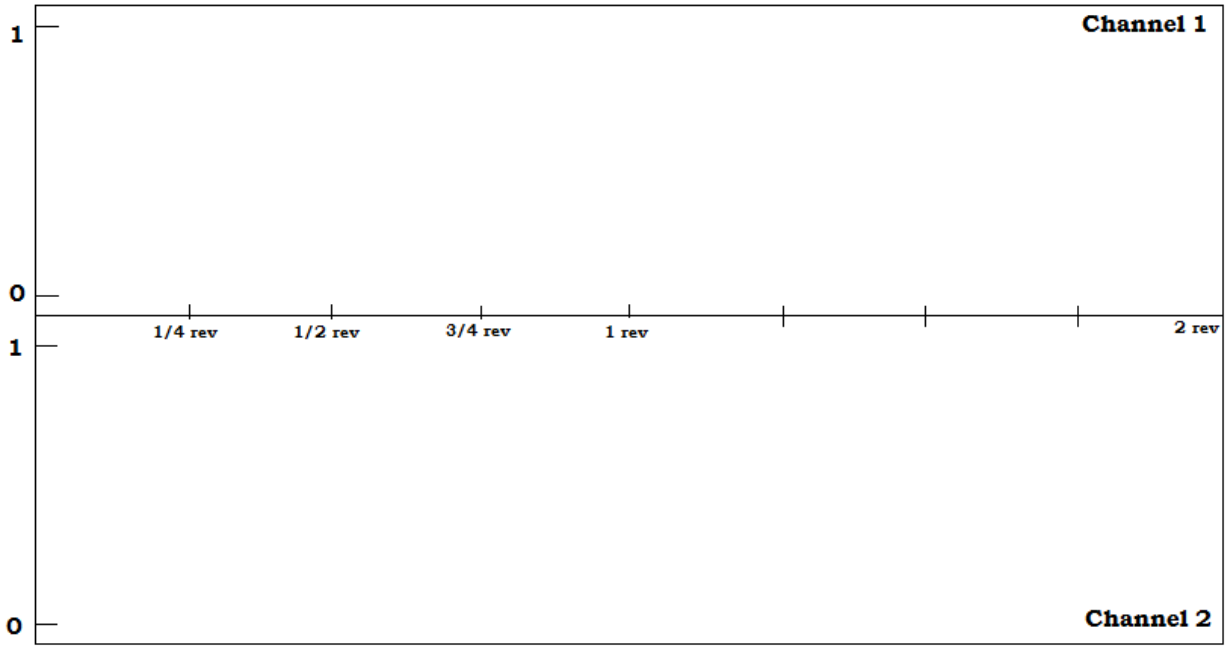
$$inner\ wheel:\ \theta_f = \frac{\emptyset R}{r_w} \qquad outer\ wheel:\ \theta_f = \frac{\emptyset(R + D)}{r_w}$$

3. Pre-Lab Exercises

1. A optical encoder with 1024 stripes is directly attached to the shaft of a DC motor. A gearbox with a 1:20 gear ratio is attached on the other end of the motor, and used to drive a wheel. What is the resolution, i.e. the smallest increment of rotation (in degrees) of the wheel that can be measured by the encoder using X4 quadrature decoding/counting?

2. If you have a 2-channel optical encoder with the following color pattern wheel, sketch what the signals look like for each channel. White is high (1) and black is low (0). If you think a sensor is half on and half off a color, keep with the previous reading measured. **Note:** The channel locations are stationary and the wheel is moving.





3. This problem will investigate the effects of differentiating an encoder signal to estimate velocity. In MATLAB, simulate the quantization noise of an encoder signal with a resolution of 0.1 degrees by rounding a continuous signal to the nearest 0.1:

```
t = 0:0.01:1; % time array with 0.01 sample time
pos = 10*cos(2*pi*t); % continuous angular position (deg) vs. time
enc_pos = round(pos,1); % simulated encoder reading with resolution of 0.1
```

   a. Plot **pos vs. time** and **enc_pos vs. time** on the same subplot for comparison
   b. Compute the numerical derivative of both signals to obtain **vel vs. time** and **enc_vel vs. time**. Plot these on a 2nd subplot for comparison. Comment on the effect of the quantization noise when taking the derivative.
   c. Experiment with using your IIR digital filtering function from Lab 7 to reduce the quantization noise. Find a good filter weight that reduces the velocity noise without inducing too much phase lag. Plot your **filtered enc_vel vs. time** on the same axes as your results from part b. Does it matter if you filter before or after taking the derivative?

4. Suppose you have a robot with two wheels as diagrammed in Section 2.4 of the handout, with wheel radius of 2cm, and distance between wheels of 20cm. Use MATLAB to design a desired trajectory for both wheels, such that the robot will:
   a. drive in a straight line for a distance of 50 cm and stop
   b. turn in place by 180º and stop
   c. drive in a circle of radius 50cm and stop after completing a 90º arc

   In each case, design appropriate ramp trajectories for both wheels and plot the desired angle of both wheels vs. time. Pick a reasonable speed so that the robot completes each maneuver in approximately 5 seconds.

4. Laboratory Exercises

In this lab, you will experiment with the encoders on the wheels of the Lab Robots to measure rotational position and velocity, and program to robots to follow a desired trajectory using encoder feedback.

4.1 Raw Encoder Signal

4.1.1 Using the signal breakout pins on top of the Lab Robot, connect one of the encoders to your USB DAQ. DO NOT power on the Lab Robot until you have a TA check your connections.

Robot Breakout Pin	DAQ Connection
GND	GND
M1 ENC A	AI 0
M1 ENC B	AI 1

4.1.2 Open your **Lab3_GUI**. Set your sampling rate to 80000.

4.1.3 Download the Arduino Sketch *Lab9Template.ino* from Canvas and modify the motor command to drive motor 1 at a constant speed (e.g. *m1c* = 400). Put the robot up on blocks to prevent the robot from driving on the benchtop.

4.1.4 You should see two square wave signals after you run the scope. Use the GUI to estimate the phase shift and record it in Student Work Section.

4.1.5 Now flip the direction of the motor speed (e.g. *m1c* = -400). Use the GUI to estimate the phase shift again and record it in Student Work Section.

4.1.6 Think about how you can use the frequency and phase between the two channels to estimate the velocity of the motor. You will answer this question in the post lab.

Have your TA check your progress

4.2 Encoder Counting using Arduino

4.2.1 Disconnect the lab robot from the USB DAQ. In this section, we will use the Arduino to read the encoder signals on the lab robot, and compare the counting performance with and without using interrupt pins.

4.2.2 Set the motor commands back to zero (e.g. *m1c* = 0).

4.2.3 You will notice that the Lab9 Arduino template creates an encoder object using pins 21 and 20, which are already connected to the outputs of the Motor 1 Encoder. Pins 21 and 20 are interrupt capable, so the encoder library will automatically use software interrupts when

counting the quadrature signal on these pins. Upload the sketch to the Arduino Mega on your lab robot and test that it works. Rotate motor 1 back and forth by hand and verify that the count increases and decreases on the Serial monitor.

4.2.4 Now we want to see what happens if we try to count using non-interrupt pins. Since AI 13 and AI 14 are easily accessible on the breakout panel of the lab robot, we will use those as our non-interrupt pins.

- Use alligator wires to connect the M1 ENC A output on the lab robot to the AI 13 input, and the M1 ENC B output on the lab robot to the AI 14 input.
- Create two more encoder objects in your setup function as follows:

```
Encoder myEnc2(A13,20); // this one uses one interrupt
Encoder myEnc3(A13,A14); // this one uses no interrupts
```

- Now add code to your loop function to read these two encoder objects

```
counts2 = myEnc2.read();
counts3 = myEnc3.read();
```

- Re-upload your sketch and see if the counts from these two new encoder objects match the count from the original encoder object.

Have your TA check your progress

4.3 Computing Position and Velocity

4.3.1 Now, only for the original two-interrupt encoder, we will calculate the position and velocity of the wheel. You can remove the code you added in 4.2.4, but keep the original encoder object and read() command.

4.3.2 First, add code to the loop function to calculate the position of the wheel in *deg*. You will have to make use of the constants provided for you in the Lab 9 template, including the countsPerRev and the GearRatio. Store your result in the variable named *theta1*. Rotate your wheel by 360 degrees and verify that the position is computed correctly. Write your equation for position in the Student Work Section below.

4.3.3 Next add code to calculate the velocity, in *deg/s*, of the wheel and store your result in the variable named *omega1*. Write your equation for velocity in the Student Work Section below.

4.3.4 Modify the printing code in your loop function to print *theta1* and *omega1* to the Serial monitor with tabs ("\t") between. Run your sketch and use the Serial Plotter in the Arduino GUI to observe how the position and velocity estimates of your encoder behave. Note how your velocity readings fluctuate between 0 and non-zero values due to the stair-stepping of the position readings from the encoder (quantization noise). Apply a digital IIR filter to your velocity to smooth out the velocity readings. Write your filter equation in the Student Work Section below:

Have your TA check your progress

4.4 Trajectory Following

The objective of this section will be to use encoder feedback on both wheels of the lab robot to make the robot follow a desired trajectory. Make sure the robot is still up on blocks to prevent the robot from driving on the benchtop.

4.4.1 Add code to your sketch to read the encoder on the other wheel using pins 19 and 18, and compute the angular position of the wheel. Verify that when you rotate both wheels forward, the position of both wheels has the correct magnitude and sign. If not, add negative signs as necessary.

4.4.2 Based on the methods in section 2.4, add code to your loop function to compute a desired trajectory that will ramp up the desired position of each wheel until reaching a desired final position of 50 cm in about 10 seconds. Print the desired position of your wheels to the serial monitor to verify that they both ramp up to 50 cm and stop. Write the equations for your desired trajectory in the Student Work section below.

4.4.3 Based on the methods in section 2.4, add a proportional control law for each motor to compute the motor commands *m1c* and *m2c* based on the error between the actual position and desired position. Print the errors and the motor commands to the serial monitor for debugging purposes, and verify that a positive motor command drives each wheel in the forward direction. If not, add negative signs as necessary. With the robot up on blocks run the sketch, your wheels should move and you should see your desired position, actual position, error, and motor command changing in the serial monitor. Start with a low proportional gain and increase it as necessary to get the robot to drive smoothly. If the gain is too low, the robot may not move until the error builds up very large. If the gain is too high, the robot may behave too jittery/unstable. If the gain is well-tuned, the robot should closely follow the desired trajectory (the error should remain small). Once you believe your gain is well-tuned, set your robot on the ground and test your sketch to see if the robot drives in a straight line for the correct distance. Write your proportional control laws in the Student Work section below.

4.4.4 Now change the desired trajectories so that your robot turns in place by 180 degrees. Write the equation for your desired trajectory in the Student Work section below. Note: the only difference between the previous trajectory and this one is how you code *theta_final* for each wheel.

4.4.5 Finally, change the desired trajectories so that the robot drives in a circle of radius 50cm and stops after completing a 90º arc. Write the equation for your desired trajectory in the Student Work section below.

Have your TA check your progress

4.5 Student Work

Name_____

4.1.4 Phase shift = _____

4.1.5 Phase shift = _____

4.3.2 Equation for Position of Wheel: *theta1 =*

4.3.3 Equation for Velocity of Wheel: *omega1 =*

4.3.4 Digital Filter Equation for Velocity: *omega1f =*

4.4.2 Desired Trajectory for Driving Straight (show numerical values of parameters)**:**

theta1_des =

theta2_des =

4.4.3 Proportional Control Laws (show numerical value of *Kp*)**:**

m1c =

m2c =

4.4.4 Desired Trajectory for Turning in Place (show numerical values of parameters)**:**

theta1_des =

theta2_des =

4.4.5 Desired Trajectory for Driving in a Circle (show numerical values of parameters):

theta1_des =

theta2_des =

5. Post-Lab Exercises

1. Knowing how dual channel encoder signals work, how would you determine the velocity of the geared DC motor output shaft (rad/s) without using quadrature encoding? Assume you can measure the frequency of the square waves, f_1 and f_2 (Hz), and the phase of each signal θ_1 and θ_2 (deg). Describe in words and equations. (note: the encoders have 16 counts per revolution and the gear ratio is 70:1)

2. The Encoder Library used on the Arduinos in this lab watches two channels for the rise and fall of the digital signals. Below is a table that has all of the channel cases assuming no lost counts, signals 1 and 2 are at a 90 degree phase, and 4x counting.

   a. In the table of encoder channel cases, fill in the count column deciding if it is +1, 1, or 0. Hint: It may be helpful to sketch a pair of quadrature cycles and in each case, use the old and new pin readings to identify where you currently are on the quadrature cycle and whether you are moving forward or backward (or not moving) along the quadrature cycle.

Case	Pin1 old	Pin2 old	Pin1 New	Pin2 New	Count
1	0	0	0	0	
2	0	0	0	1	+1
3	0	0	1	0	
4	0	1	0	0	
5	0	1	1	1	
6	1	0	0	0	
7	1	0	1	1	
8	1	1	0	1	
9	1	1	1	0	
10	1	1	1	1	

   b. Assuming your count increments from the previous problem are correct. Write pseudo code that is called when either Pin1 or Pin2 changes value. The function should output the current count.

6. Project Milestone

The purpose of PM 9 is to implement encoder position control and finalize sensing/control strategy for navigating the competition field. You will demonstrate the ability to navigate your robot using encoder feedback. During Lab 10, your team will present the following in a 15 minute PowerPoint presentation. Upload a copy of your presentation on Canvas before Lab 10 begins.

Presentation (40 points)

The presentation should have a detailed description of your encoder odometry and control method as well as a finalized autonomous method. Your presentation should include:

- A summary of each team member's contributions for this week.
- An updated CAD model of your robot
  - A rendering of your model
  - Indicate changes made from previous PMs
- A description of your autonomous plan.
  - What is your sensing and control method for navigating the playing field?
  - It may be useful to reference the state transition diagram.
  - Discuss how you will use wireless communication in case of malfunction/failure.
- An updated state transition diagram
  - Make sure to label any states that make use of feedback control (i.e. line following, drive straight, wall follow, etc)
  - Incorporate any changes to your previous sensing methods
- A description your robot's control algorithms
  - Position control using encoders
  - Wall following and/or approach
  - Line following
  - Others as applicable
- Updated pin tables for each micro controller
- Updated wiring schematics for your robot and wireless transmitter.

Demonstration (50 points)

During Lab 10, you will demonstrate position control using encoder feedback. You will also show your TA the code you used to perform feedback control (at least proportional position control) for the following tests.

- Straight Line Test – drive in a straight line for a certain distance and speed. The TA will give you the distance and speed at the time of demonstration.
- Turning Test – turn in place for a specified number of degrees that will be given at the time of demonstration.
- Circle Test – drive in a circle with a radius that will be given at the time of demonstration. The radius should line up with the inner wheel of the robot.

Be able to quickly change the requested driving parameters (distance, speed, radius, etc) at the time of demonstration. The TA will give you these numbers at the time of the demonstration. If you have any questions about the PM ask your TA.