

CONTROL SYSTEMS LABORATORY

ECE356

LAB 3: Familiarization with Equipment and Basic Cruise Control Design

1 Purpose

The purpose of this experiment is to introduce you to the lab setup and the associated control problem. Namely, the design of a cruise control system for a car.

2 Introduction

A car moves on a straight road with an unknown slope. It is assumed that the inertia of the wheels is negligible, that the friction force is proportional to the speed of the car, and that the engine imparts a force u . The schematic representation of the system is depicted below.

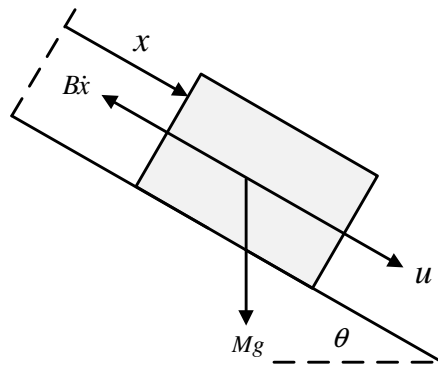


Figure 1: Schematic representation of the system

Applying Newton's law we obtain the following mathematical model of the system

$$M\ddot{x} = -B\dot{x} + u + Mg \sin \theta. \quad (1)$$

Since we are interested in controlling the speed \dot{x} of the car, we rewrite the model using $v = \dot{x}$,

$$M\dot{v} = -Bv + u + Mg \sin \theta. \quad (2)$$

The force u , imparted by a DC motor, is approximately proportional to the voltage v_m applied to the motor,

$$u = K_m v_m. \quad (3)$$

The voltage v_m is the control input to the plant. The mathematical model of the cart system becomes

$$\dot{v} = -\frac{B}{M}v + \frac{K_m}{M} \left(v_m + \frac{M}{K_m} g \sin \theta \right). \quad (4)$$

Since the slope θ is unknown, the constant $\frac{M}{K_m} g \sin \theta$ is a disturbance acting on the plant. Letting

$$a = \frac{K_m}{M}, \quad b = \frac{B}{M}, \quad d(t) = \bar{d} \cdot \mathbf{1}(t) := \frac{M}{K_m} g \sin \theta \cdot \mathbf{1}(t), \quad (5)$$

the plant block diagram is depicted in Figure 2.

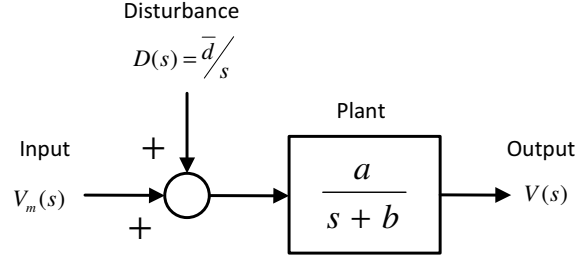


Figure 2: Block diagram of the plant

In this lab, you will familiarize yourself with the experimental setup, you will experimentally determine the constants a and b , and you'll implement basic proportional and proportional-integral controllers to regulate the car speed.

The experimental setup used in the lab is shown in Figure 3 .

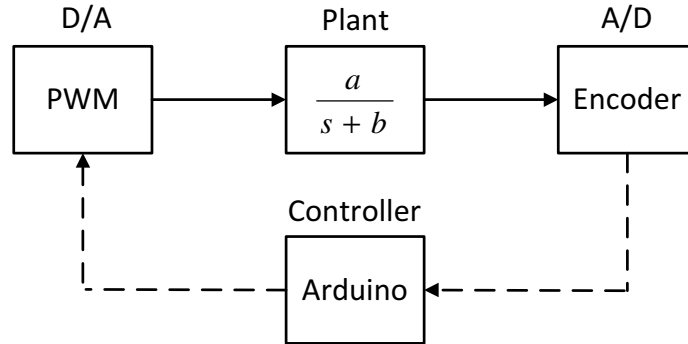


Figure 3: Block diagram of the experimental setup

The designed controller will be implemented on the Arduino board. As shown in the figure, the micro controller samples the position of the cart from the encoder (A/D) and computes the control input to be applied to the cart. Given the desired control input, the motor shield generates a PWM

signal that approximates the desired control value (D/A). (see the Appendix section for further information about the experimental setup)

3 Preparation

Consider the block diagram in Figure 2 and assume that the road is flat, i.e., $\theta = 0$ and hence $D(s) = 0$. Suppose that a step voltage $v_m(t) = V_0 \cdot \mathbf{1}(t)$ ($V_0 > 0$) is applied to the DC motor and that at time $t = 0$ the cart is still (i.e., $v(0) = 0$). Using the final value theorem determine $v(+\infty) = \lim_{t \rightarrow \infty} v(t)$ in terms of V_0 , a , and b .

Submit the preparation report to your lab TA at the beginning of the lab. Read thoroughly the next section before attending the lab.

4 Experiment

4.1 Introduction to the Arduino

4.1.1 Apply sinusoidal voltage to the DC motor

In this section you will learn how to configure and setup your system. The objective is to have the cart on the rails swing back and forth following a sinusoidal pattern. Note that the various files on the computer that you will use have the label “**Lab2**” attached. This is because this experiment used to be Lab 2. Rather than renaming the files and directories, we choose to alert you to this label mismatch.

First, open the Arduino code file named “*Lab2_4_1.ino*” located in the *Lab2 > Lab2_4_1* directory and then press the **upload** button in the Arduino software to upload the code on the micro-controller. As the code is uploaded the cart will start moving for 15 seconds. Pay attention to the amplitude and frequency you will impose on the sinusoid, as it will affect the swing of the cart on the rails.

Have your TA sign here before proceeding to the next step.

4.1.2 Collecting data from the cart

To plot the position of the cart on the rails, open the Matlab m-file located in the same directory as the Arduino code named as “*real_time_data_plot_Lab2_4_1.m*” . Then launch the m-file as it will run the uploaded code on Arduino. The data received from Arduino is recorded and plotted automatically.

4.2 Identification of model parameters a and b

Recall that in the introduction we modelled the car as a transfer function $V(s)/V_m(s) = a/(s + b)$, where $V_m(s)$ is the Laplace transform of the input voltage signal and $V(s)$ is the Laplace transform of the cart speed signal. Before controlling the system, we need to experimentally determine the

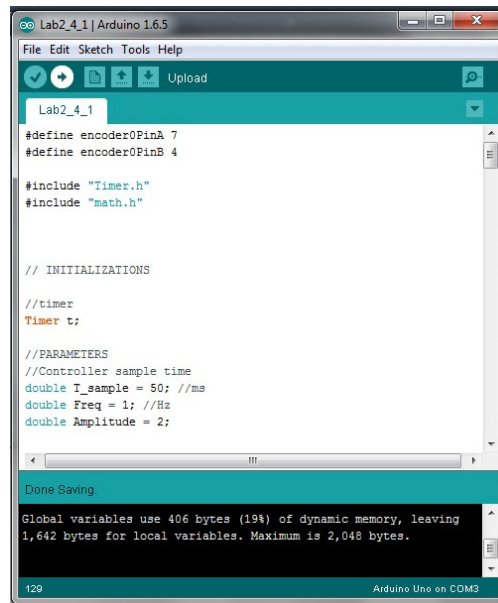


Figure 4: Arduino software user interface

parameters a and b . This is the objective of this section.



Figure 5: Schematic representation of the Matlab Simulink model

1. Create a Simulink model as in Fig. 5. The detailed steps to construct it are listed below.

- Open a blank Simulink model and the Library browser.
- Using the **Continuous/Transfer Function** block, create a system with transfer function $\frac{a}{s+b}$. This block represents the mathematical model of the plant assuming that the disturbance is zero, that is, assuming that the cart track is horizontal. The objective here is to experimentally determine the values a and b .
- Using the **Sources/Signal Generator** block in Simulink, create a block that generates a square wave for $v_m(t)$ with amplitude 1.5 V and frequency 0.5Hz.
- Connect the output of the transfer function block to a Scope (found in the **Sinks** library).

Have the TA check your model and sign here before proceeding to the next step.

2. We choose the initial guesses for a and b in the system model as 0.5 and 5 respectively.
3. Run the Simulink model to obtain the corresponding results for the considered a and b . The scope data is stored in the Matlab workspace to be compared with the experimental results.
4. Then open the Arduino code file named as "*Lab2_4_2.ino*", following the same procedure described in the previous section, upload the code and then launch the provided Matlab m-file "*real_time_data_plot_Lab2_4_2.m*" to observe the obtained experimental results. The uploaded code will impose the same square wave form considered in the Matlab Simulink to the DC motor. Both the displacement and the velocity of the cart are recorded and illustrated in the obtained figure. The experimentally obtained velocity can be used to modify and correct the initial guesses for the a and b parameters.

Have your TA sign here before proceeding to the next step.

5. Look at the experimental velocity and determine as accurately as you can its total variation over the half period. We'll denote such variation by Δv .
6. Notice that over the half period under consideration, the signal $v_m(t)$ performs a step of amplitude $V_0 = 3V$. We'll make the approximation to consider the signal v_t to be in steady state at the end of the half period. So we deduce that, in response to an input step of amplitude $3V$, the plant output has a total variation of Δv . Using the formula you found in your lab preparation and the value Δv you just found, find a relationship between a and b . Specifically, find an expression of the type $a = f(b)$.

Have your TA sign here before proceeding to the next step.

7. It should be now clear that for any choice of b , setting $a = f(b)$ guarantees that the steady state value of the model output approximately coincides with that of the actual plant output. Now you'll tune b to make sure that the transients coincide as much as possible.

Keep the value of b you were using earlier, and set $a = f(b)$ in Simulink. Run the simulation and verify that the steady-state values of the model and actual plant outputs coincide.

8. Now try to increase b . Don't forget, every time you modify b , you must also update $a = f(b)$ in Simulink. Run the simulation to see if the new value of b yields better results. Keep tuning b until you minimize the discrepancy between the actual plant and model outputs.

You'll use the values of a and b you just found in Lab 4.

Ask the TA to check that your estimates of a and b are reasonable, and have the TA sign here before proceeding to the next step.

4.3 Proportional Control

Now you'll implement a proportional controller to regulate the speed of the cart. A proportional controller is a controller of the form

$$v_m(t) = Ke(t), \quad (6)$$

where $e(t) := r(t) - v(t)$ is called the tracking error. This is the difference between the reference signal $r(t)$ and the actual plant output $v(t)$. In the cart experiment, $r(t)$ represents a desired velocity profile for the car, while $v(t)$ represents the actual cart speed. Do the following steps for this section.

- Open the Arduino code file named as "Lab2_4.3.ino", The velocity reference is set to a square wave form with the amplitude of 0.2 m/s and frequency of 0.5 Hz.
- Set $K = 5$, then compile and upload the code.
- Following the same procedure described in the previous sections, launch the provided Matlab m-file "real_time_data_plot_Lab2_4.3.m" to observe the obtained experimental results. Both the displacement and the velocity of the cart are recorded and represented in the obtained plot.

Have your TA sign here before proceeding to the next step.

- You'll notice that the controller does not succeed in regulating the speed to the desired value. Increase the controller gain, and run the system again. Repeat this operation a few times. Do not increase K beyond 20.

Record your observations: does the P controller successfully regulate the speed to the desired value? What's the effect of increasing the gain K on the output response?

- Save the plot of the output response obtained when $K = 20$.

Have your TA sign here before proceeding to the next step.

4.4 Proportional-Integral Control

You'll now implement a proportional-integral controller. A PI controller is an enhancement of a P controller and has the form

$$v_m(t) = Ke(t) + \frac{K}{T_I} \int_0^t e(\tau) d\tau, \quad (7)$$

where K and T_I are two positive design constants. Your TA will explain to you the rationale for using the controller. Taking the Laplace transform, we find that a PI controller has a transfer function (from e to v_m),

$$K + \frac{K}{T_I s} = K \left(\frac{T_I s + 1}{T_I s} \right) \quad (8)$$

Do the following steps for this section.

- Open the Arduino code file named as "*Lab2_4_4.ino*", The velocity reference is set to a square wave form with the amplitude of 0.2m/s and frequency of 0.5Hz.
- Set $T_i = 0.07$ and $K = 2$. Compile and upload the model and run it.
- Following the same procedure described in the previous sections, launch the provided Matlab m-file "*real_time_data_plot_Lab2_4_4.m*" to observe the obtained experimental results. Both the displacement and the velocity of the cart are recorded and represented in the obtained plot.
- Save the plot for $T_i = 0.07$ and $K = 2$.

Have your TA sign here before proceeding to the next step.

- Next, keep T_i constant and start increasing K . **Do not increase K beyond 20.**
Record your observations: what's the effect of increasing K ? How does the performance of the P and PI controllers compare?
- Save a plot of the output response when $T_i = 0.07$ and $K = 20$.

Have your TA sign here before proceeding to the next step.

5 Report

Please submit your final report before the assigned deadline to your laboratory TA.

6 Appendix A

6.1 What is Arduino?

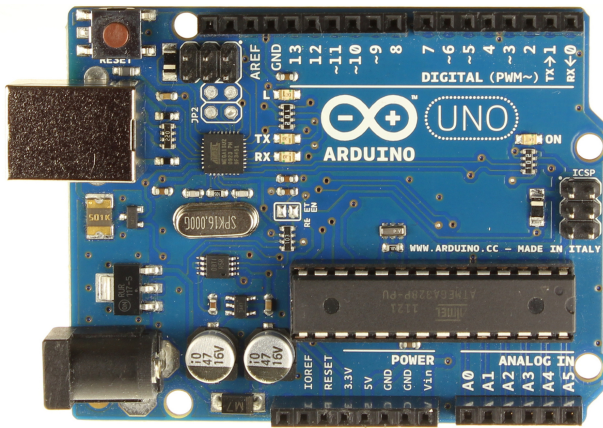
Arduino is a microcontroller that can provide extremely portable computing to circuits, enabling them to read both analog and digital inputs, perform preliminary calculations, and produce a variety of outputs. It's a physical computing platform, and a development environment for applying small-scale logic and software on circuits.

The versatility of these microcontrollers can be used to develop interactive circuits, that can receive both analog and digital inputs from a wide variety of switches and/or sensors, as well as control a variety of LEDs, motors, and other physical outputs. Arduino boards can also be used either as the primary processor in deducing logic in circuits, or as nodes that can communicate with software on other devices.

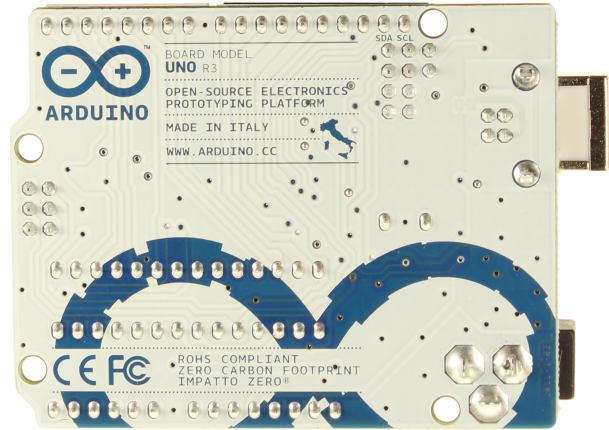
Arduino's open-source IDE (Integrated Development Environment) can be used to develop and publish software for its microcontrollers, and is available for download on Arduino's website.

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; all that is required to power the microcontroller is a connection to a computer with a USB cable or to an AC-to-DC adapter or battery.

One of the features that make the Arduino a suitable tool to control a DC motor is its ability to generate PWM signals. PWM signals, as detailed in the following sections, can be interpreted as a simple DC voltage under specific conditions.



(a) Uno R3 Front.



(b) Uno R3 Back.

Figure 6: Arduino Uno

Arduino Uno Summary	
Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

6.2 Pulse Width Modulation (PWM)

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate DC signals with a voltage between the full on voltage (5 Volts) and off voltage (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the "pulse width". To get varying analog values, you change, or modulate, that pulse width.

One thing to keep in mind is that, without a high enough PWM frequency, this effect deteriorates. Also, the Arduino's default PWM frequency ($\approx 500\text{Hz}$) is much lower than the desired frequency ($\approx \text{a few KHz}$). As a result, the frequency of the Arduino's timers had to be explicitly increased using the timers' registers, and the PWM duty cycle had to be altered using more complex methods. This is discussed in more detail in the following sections.

In the image below, the green lines represent a regular time period. The period is given as: $\text{Period} = 1/(\text{PWM Frequency})$

In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` induces a 100% duty cycle (always on), while `analogWrite(127)` induces a 50% duty cycle (pulse width = 1ms) for example.

6.2.1 The Atmega 168/328 Timers and using the PWM Registers

The ATmega328P has three timers known as Timer 0, Timer 1, and Timer 2. Each timer has two outputs, each with a corresponding compare register. When the timer reaches the compare register value, the corresponding output is toggled. As a result, the two outputs for each timer will operate at the same frequency, but can have different duty cycles, (depending on what values are set for the

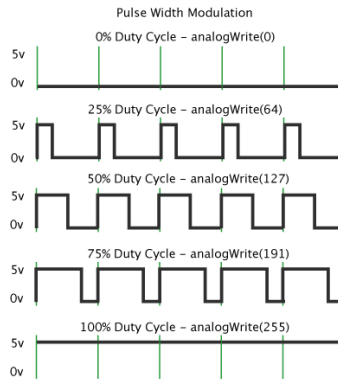


Figure 7: PWM at different duty cycles.

respective output compare register).

Each of the timers has a register that controls the frequency, and the frequency is set by dividing the system clock frequency of 16MHz by a prescale factor, determined by the register's value. The prescale factor may include values such as 1, 8, 64, 256, or 1024. Note that Timer 2 has a different set of prescale values from the other timers. The timers are complicated by several different modes. The main two PWM modes are "Fast PWM" and "Phase- correct PWM". In this lab, only "Fast PWM" will be discussed.

Timer Registers are used to control each timer. The Timer/Counter Control Registers TCCRnA and TCCRnB hold the main control bits for the timer. (Note that TCCRnA and TCCRnB do not correspond to the outputs A and B.)

6.2.2 Fast PWM

In the simplest PWM mode, the timer repeatedly counts from 0 to 255. The output turns on when the timer is at 0, and turns off when the timer matches The Output Compare Registers (OCRnA and OCRnB). The higher the value in the output compare register, the higher the duty cycle.



Figure 8: The use of OCRnA registers to generate PWM signals

6.2.3 Fast PWM Mode

The following code fragment sets up fast PWM on pins 3 and 11 (Timer 2). To give an example of the use of register settings:

```
pinMode(3 , OUTPUT);
pinMode(11 , OUTPUT);
```

```

TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 180;
OCR2B = 50;

```

Bits Set	Result
Setting the waveform generation mode bits WGM to 011	selects fast PWM.
Setting the COM2A bits and COM2B bits to 10	provides non-inverted PWM for outputs A and B.
Setting the CS bits to 100	sets the prescaler to divide the clock by 64. (Since the bits are different for the different timers, consult the datasheet for the right values.)

The output compare registers are arbitrarily set to 180 and 50 to control the PWM duty cycle of outputs A and B.

On the Arduino Duemilanove, these values yield:

- Output A frequency: $16 \text{ MHz} / 64 / 256 = 976.5625 \text{ Hz}$
- Output A duty cycle: $(180+1) / 256 = 70.7\%$
- Output B frequency: $16 \text{ MHz} / 64 / 256 = 976.5625 \text{ Hz}$
- Output B duty cycle: $(50+1) / 256 = 19.9\%$

The output frequency is the 16 MHz system clock frequency, divided by the prescaler value (64), divided by the 256 cycles it takes for the timer to wrap around.

6.3 The Relationship Between PWM and DC

Given that the DC motor acts like a filter, the observed DC output at the motor's side is simply the average of the PWM signal.

So, if the peak-to-peak voltage of the PWM signal = 5v, and the Duty Cycle is 40%, the corresponding DC output observed by the motor is 40% of the 5v, which equals 2v. Note that the PWM frequency should be high enough for the motor to behave in the desired manner. In order to get a certain DC output, lets say 3v, we start by determining the duty cycle of the PWM by dividing the desired output over the peak-to-peak voltage of the PWM signal, $3\text{v} / 5\text{v} = 0.6 = 60\%$. Then the value in the output compare register OCR2A = $(0.6 * 256) - 1 \approx 153$.

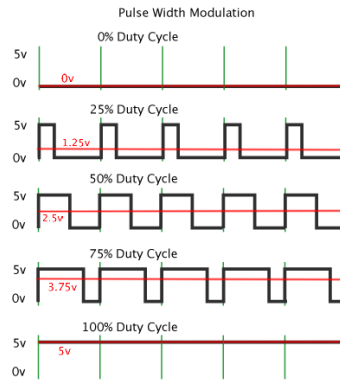


Figure 9: PWM at different duty cycles and the corresponding DC output

7 Appendix B

A rotary encoder is a device for measuring angular movement and is used to measure the rotation of motors or rotating shafts. The simplest form of encoders utilizes a light source and two photodetectors (light sensors) within close proximity of one another behind a slotted disk; those photodetectors form the channel waveforms A and B.

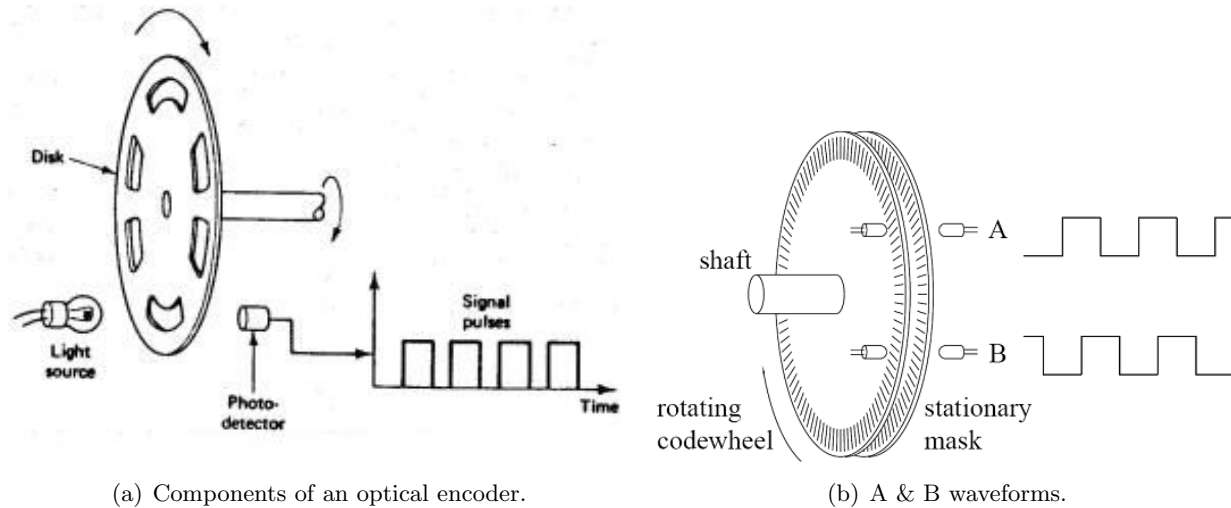


Figure 10: Illustration of optical encoders

The slotted disk is attached to the rotating shaft, and as it rotates the slots allow the light to reach the photodetectors turning their signals on and off. Reading those waveforms simultaneously can provide a very precise measurement of the angular movement. We discuss how to read the encoder waveforms in the interrupts section.

Below is an image showing the waveforms of the A & B channels of an encoder.

When the code finds a low-to-high/ high-to-low transition on the A channel, it checks to see if the B channel transitioned then increments/decrements the variable to account for the direction that the

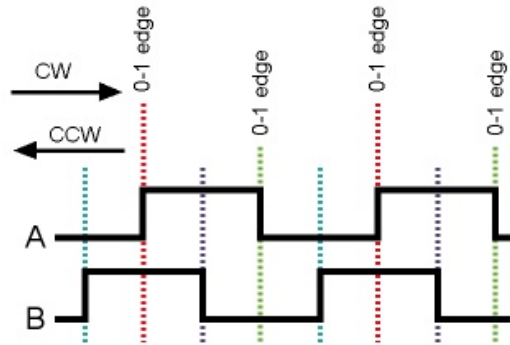


Figure 11: Waveforms of the A & B channels of an encoder

encoder must be turning in order to generate the waveform found (and visversa for the B channel).

Forward direction (CW):

- (A=Low & B=High) then (A=High & B=High),
- (A=High & B=Low) then (A=Low & B=Low)

Reverse direction (CCW):

- (A=High & B=Low) then (A=High & B=High),
- (A=Low & B=High) then (A=Low & B=Low)

7.1 Interrupts

An interrupt, as the name hints, is a signal emitted from hardware or software events to the processor, allowing the processor to pause its current activities momentarily to perform some (usually high-priority) processing based on the triggering event. The processor responds by suspending its current activities, saving its state, and executing a method or function called an interrupt handler to deal with the event. This interruption is temporary, and after the interrupt handler finishes, the processor resumes execution of the previous thread.

7.2 Using Interrupts to Read Rotary Encoders

One could attach interrupts to the A & B channels. When the Arduino sees a change on the A channel, it immediately skips to the “doEncoder” function, which parses out both the low-to-high and the high-to-low edges, and the same happens for the B channel.

Using interrupts to read a rotary encoder would be very suitable, considering the interrupt service routine (a function) can be short and quick, because it doesn’t need to do much.

8 Appendix C

8.1 Motor Drivers

Motors typically require voltages and/or currents that exceed what can be provided by the controlling logic circuits. The motor driver provides the interface between the signal processing circuitry and the motor itself. Based on the input signal, the motor controller can provide a corresponding voltage and/or current to the motor. It can be thought of as the amplifier for the motor.

In order to prevent system failures in the event of electrical or mechanical faults, motor drivers may also include robust protection schemes, including short-circuit, over current, over temperature, shoot-through and under-voltage protection.

In this experiment, the Cytron 10A Motor Driver will be used.

8.1.1 Cytron 10A Motor Driver Shield



Figure 12: SHIELD-MD10 - Cytron 10A Motor Driver Shield

SHIELD-MD10 is an Arduino shield for controlling high current brushed DC motor up to 10A continuously.

SHIELD-MD10 Shield comes with these features:

1. Bi-directional control for 1 brushed DC motor.
2. Support motor voltage ranges from 7V to 25V.
3. Maximum current up to 10A continuous and 15A peak (10 seconds).
4. 3.3V and 5V logic level input.
5. Solid state components provide faster response time and eliminate the wear and tear of mechanical relay.
6. Fully NMOS H-Bridge for better efficiency and no heat sink is required.
7. Speed control PWM frequency up to 10KHz.
8. Stackable I/O header pin.
9. Selectable digital pins for PWM and DIR.

8.2.2 Motion Encoder

The IP02 Linear Motion Servo Plant by Quanser also contains a high precision encoder. The encoder outputs 4096 clicks per revolution, which translates to about 0.09 degrees per click.

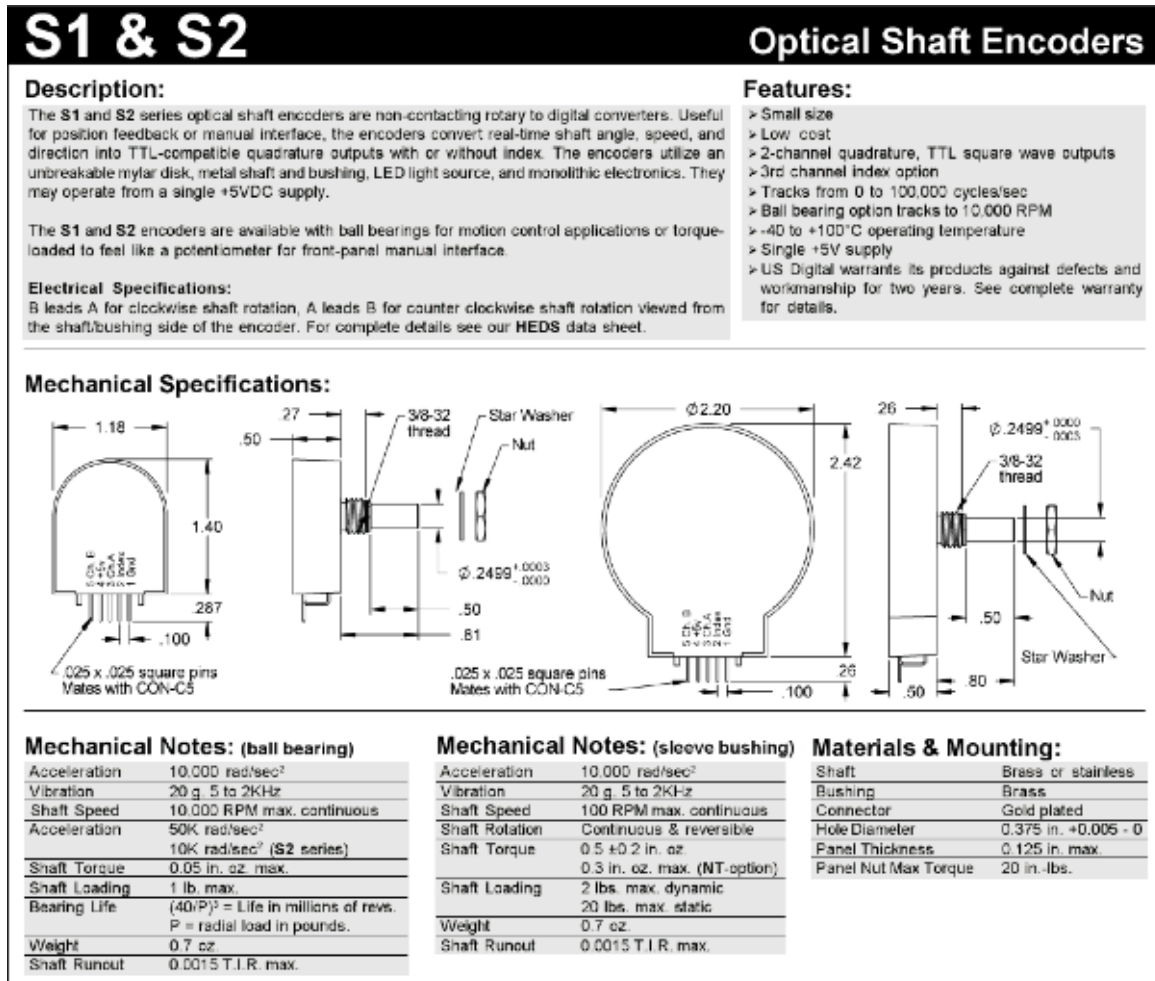


Figure 14: The specifications for the encoder.