Experiment C7
Inverted Pendulum
Procedure

Deliverables: Checked lab notebook, Technical Memo

SAFETY FIRST: You must wear safety glasses for this entire lab!

Overview

In this lab, you will build a "servo" using a DC motor with an encoder on the shaft. Feedback from the encoder will allow you to hold a pendulum at a desired angle θ_S. Balancing the angular acceleration with the torque from gravity, the viscous drag force of ar, and the motor yields the equation of motion for the single pendulum

$$mR^2\ddot{\theta} = -mgR\sin\theta - \gamma R^2\dot{\theta} + \tau\,, \qquad (1)$$

where m is the pendulum mass, R is the distance from the pivot to the center of mass, γ is the viscous drag force coefficient, and τ is the applied motor torque. The solution to Eq. (1) for an "underdamped" pendulum is

$$\theta(t) = e^{-\lambda t}\sin\left(\omega_d t\right), \qquad (2)$$

where the decay constant $\lambda = \gamma/2m$ and the ringing frequency $\omega_d = \sqrt{g/R - \lambda^2}$. In the pre-lab assignment, you will use Eqs. (1) to derive an LQR controller for the pendulum.

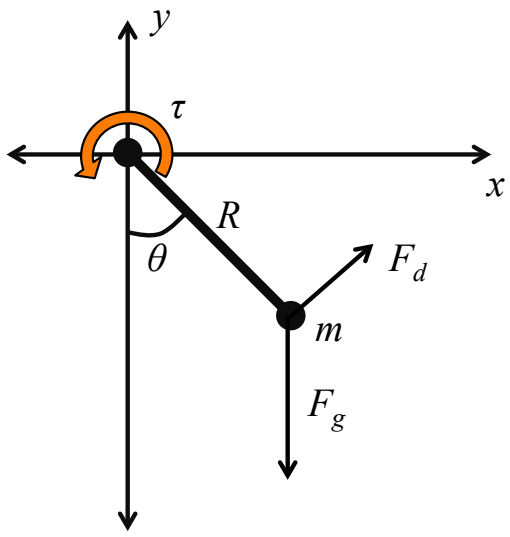


Figure 1 – A free body diagram shows the forces acting on a pendulum. The gravitational force F_g and viscous drag force F_d occur naturally, while a DC motor is used to apply a torque τ to the pendulum.

Part I: Sensor Calibration and Data Acquisition

Background

The pendulum is coupled to the shaft of a DC motor with a 500-count encoder, which will be connected to a special 3-axis encoder shield. This device contains a counter-timer IC chip that reads the encoder pulses, determines the angular displacement of the shaft, and sends the values to the Arduino through its SPI serial port.

Experimental Procedure

1. Assemble the 80-20 structure shown in Appendix D. Use several 3M Command Strips™ to securely fasten the H-bridge and Arduino boards to the top of the base.
2. Mount the motor to the 80-20 motor mount that you made for the C5 deliverable. Use the M2.5 metric screws.
3. Mount the anodized blue aluminum shaft coupling to the motor shaft using the set screws.
4. Make sure the Arduino is turned off and disconnected from any power source including the USB cable. Carefully mount the 3-axis encoder shield to the Arduino. Make sure the pins are well aligned.
5. Refer to the pin-out for the motor encoder in Appendix B. Use the colored ribbon cables to connect CHAN A, CHAN B, Vcc +5V DC, and ENC GND to the Arduino encoder shield.
6. Download the "AxisEncoderShield3.ino" code from the C7 website, and save it to your C7 folder with the name "C7_encoder_calibration.ino".
7. Open the code and look at the "setup". Note that it uses pins 8, 9, and 10, so you will not be able to use these in the programs you are about to write. All other pins are available.
8. Connect the Arduino to the lab computer, select the appropriate COM port, and load the encoder shield program to the Arduino.
9. Open the "Serial Monitor" to see the output displayed as text. Rotate the motor shaft. You should see the encoder count change for one of the three channels. Which encoder channel is the motor connected to? X, Y, or Z?
10. In the main loop of the sketch, delete the encoder measurements and subsequent "serial.print()" functions for the two encoder channels that are not connected. Also, delete any "delay()" functions in the main loop.
11. Open the "Serial Plotter" to see the output displayed as a graph. Rotate the motor shaft. You should see the line on the graph change.
12. Screw the pendulum (threaded rod) back into the shaft coupling. Tighten the nut to hold it in place.
13. Calibrate the encoder. With the code running and the Serial Monitor open, rotate the pendulum one full revolution (360°). How many encoder counts does this yield? Record the value in your lab notebook.

14. Modify the main loop of the code to print the time in ms and angle in degrees instead of the number of encoder counts. Save a copy of the code to your code library, and give it an intelligent file name.
    a. It should print in the format "time, angle". This will make it easier to import your data into Matlab.
    b. It should start at 0° when hanging vertically downward, increase when rotated counter-clockwise, and decrease (give negative angles) when rotated clockwise.
    c. Use the millis() function to get the time and store it as a float point variable.
    d. You may need to put a nested loop inside the main loop.
15. Thoroughly test it to make sure it works.

Part II: System Identification and Characterization

Background

You will now determine the ringing frequency and damping ratio for the pendulum. You will use a procedure similar to the ringing of the Baseball bat in Lab I. Then, you will connect the DC motor to an L298N H-bridge circuit, and use the PWM output from the Arudino to control the applied motor torque *M*. Importantly, you will need to measure how motor torque *M* is related to the PWM signal, similar to how you related flow rate to PWM output for the pump in C5 and C6.

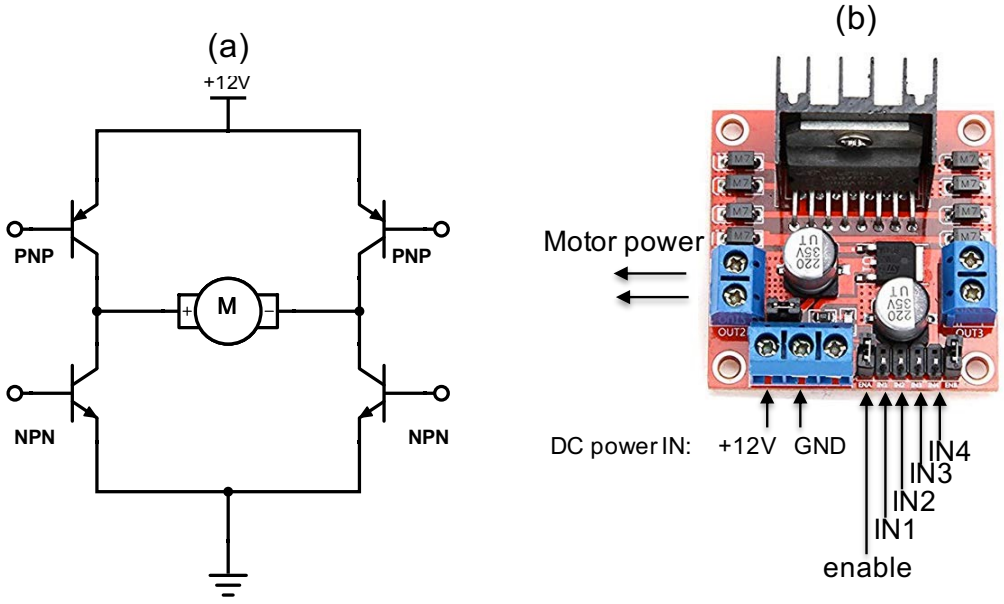


Figure 2 – (a) An H-bridge circuit uses four transistor to control the direction of the motor torque. (b) An L298N IC chip mounted to breakout board contains a more sophisticated, user-friendly version of the H-bridge circuit.

CAUTION: Things might get wacky here. Make sure nothing is in the full 360° path of the pendulum. (i.e. Don't get slapped in the face with threaded rod!)

Experimental Procedure

1. Make sure you are wearing safety glasses. You must keep them on for the remainder of the lab.
2. Remove the stainless steel M4 threaded rod. Measure its length L and mass m. (There should be meter sticks and a digital balance over on the counter top.) Record the values in your lab notebook.
3. Screw the pendulum (threaded rod) back into the shaft coupling. Tighten the nut to hold it in place. Make sure the apparatus is oriented in a way that the rod will not slam in anything if it is accidentally driven with too much torque.
4. Use the measured length to determine R. Note that R is the distance from the pivot to the center of mass of the rod.
5. Lift the pendulum to an initial angle of about 90°, and let it go.
    a. Record the oscillation of θ vs. time. You can copy and paste the data from the serial monitor into a .txt file, or you can try to directly save it using PuTTY.
    b. Import the data into Matlab and try to plot it. If it looks good, save the workspace as a .mat file in your C7 folder.
    c. Use the data to determine the decay constant λ (similar to what you do for the ringing of the baseball bat in Lab I).
    d. Use the decay constant λ and measured mass m to calculate the viscous drag force coefficient γ.
6. Open the Arduino code you wrote for the DC motor H-bridge in C5, and save it to your C7 folder with the filename "C7_motor_TEST.ino".
7. Connect the 24V DC and GND power input pins on L298N H-bridge board to the + and – bus lines on the small breadboard. Include the toggle switch as an emergency **kill switch**.
8. Make the following connections with the Arduino and small breadboard:
    a. Arduino GND → – bus line on the small breadboard
    b. Arduino Digital Pin 5 → IN1 on L298N
    c. Arduino Digital Pin 4 → IN2 on L298N
    d. Arduino Digital Pin 3 → Enable on L298N
9. Modify the main loop of the Arduino code to just turn the motor in one direction. No delay(), no TurnOFFA(), nothing but TurnForward(40) should be in the main loop.
10. Make sure the apparatus is oriented in a way that the rod will not slam in anything if it is accidentally driven with too much torque.
11. Try running the code with different PWM strengths. Note that the PWM strength essentially controls the stall torque of the motor. **Keep your finger on the kill switch.**

12. Copy and paste the setup and functions from the H-bridge code into the encoder shield code. Save it to your C7 folder with the filename "C7_motor_calibration.ino".
13. Delete any delay() that might be in the main loop, and add a "TurnForward()" function to the main loop.
14. Similar to the pumps in the previous lab, you will need to determine how the stall torque is related to the PWM %duty cycle. Measure the equilibrium angle as a function of the PWM value. Start with TurnForward(20) and gradually work your way up.
15. Note that there is a threshold % duty cycle for the pendulum to move from 0°. This is due to static friction or "stiction", as some mechies like to call it. Record the threshold duty cycle in your lab notebook.
16. Repeat the calibration for the other direction. Start with TurnReverse(20) and gradually work your way up.

Pro-tip: You might consider renaming the functions "TurnForward()" and "TurnReverse()" to something more intuitive.

17. Use the formula from the pre-lab assignment to convert the measured angles θ_S to torque τ_S.
18. Assuming the PWM signal is linearly related to the torque, such that PWM = $a\tau + b$, calibrate the motor and controller. There is an inherent asymmetry in the H-bridge controller circuit, so you will need to do separate calibration curves for both forward and reverse. Write down the two calibration formulas in your lab notebook.

Part III: Open-loop Controller

Background

You will now use your work from the pre-lab assignment and the information from Part II to control the angle of the pendulum.

CAUTION: Things might get wacky here. Make sure nothing is in the full 360° path of the pendulum. (i.e. Don't get slapped in the face with threaded rod!)

Procedure

1. Create a copy of the previous Arduino sketch, save it to your C7 folder with the filename "C7_open_loop_controller.ino".
2. Consider the formula relating the steady-state torque τ_S to the equilibrium angle θ_s you derived in pre-lab assignment. Use this formula, along with the motor calibration from Part II, to determine the relationship between the PWM value and equilibrium angle θ_s.
    a. Use this formula to write a program that drives the motor at the correct torque to achieve any desired angular set-point θ_s.
    b. You will need a conditional statement to handle positive vs. negative angles and torques.
    c. You will need another nested conditional statement to handle PWM values >255.
    d. Have the code print the time, calculated torque, PWM value, and measured angle θ. It should be formatted as "time, torque, PWM, angle", which will make it easier to import the data into Matlab.

3. Use your hand to hold the pendulum near the programed set-point, run the program, and let go when the motor starts supplying torque. Test the program for different set-points between -90° and 90° to make sure it has been implemented correctly.
4. With the pendulum happily sitting at a set-point near 40°, gently tap it with your hand. What happens? Save the angle vs. time data for this test, as well.
5. Test it some more, but this time start with the pendulum hanging vertically downward. (This is called "swing-up" control.) Record the angle vs. time for several different set-points between -90° and 90°. Where does your controller achieve its best result, near 0° or 90°? Does anything crazy ever happen?
6. Try it for a set-point > 90°. Does it work? Try is again, but use your hand to gently guide it into position and let go.

Part IV: LQR Feedback Control

Background

You will now implement feedback control using the LQR method. Note that we essentially have two parameters that we wish to control: the angle θ and angular speed $\omega = \dot{\theta}$. In particular, we want $\theta = \theta_s$ and $\omega = 0$.

First, we will calculate the motor torque τ_S necessary to maintain the angle θ_s. Then, we will add proportional feedback from *both* parameters, such that the torque is determined by

$$\tau = \tau_s + k_p\left(\theta_S - \theta\right) - k_d\omega, \qquad (2)$$

where k_p and k_d are the LQR feedback gains. Note that ω is the *derivative* of θ, so this is essentially a proportional-derivative (PD) controller.

Procedure

1. Create a copy of the previous Arduino sketch, and save it to your C7 folder with the filename "C7_LQR_controller.ino".
2. In the main loop, use a finite difference to calculate the angular speed ω.
3. Implement the LQR controller using the feedback given by Eq. (2).
    a. Use your calibration equations PWM = $a\tau + b$ from Part II for the forward and reverse directions.
    b. Note that the units of k_p and k_d should come out of Matlab as Nm/radian and Nms/radian, respectively. Make sure your units for angle and angular speed all work out. (i.e. Do everything in either radian and rad/s or in degrees and degrees/s.)
    c. You will need a conditional if-then statement to determine whether to use the forward or reverse function, and another nested if-then statement to keep the PWM value in the range of 0 – 255.
4. Test the controller with just proportional feedback, i.e. $k_d = 0$. Start with a small value for the proportional gain k_p and gradually increase it. When k_p becomes large, you should start to see oscillations.

5. Gradually increase the derivative gain k_d. When you think you have found a good value, try to do "swing-up" control to an angle of 180°. That is, do not use your hand to guide the pendulum into position.
6. Use the lqr() function in Matlab and the values you measured for m, R, and λ in Part II to calculate the feedback gains k_p and k_d. Note that the LQR coefficients are different for different set-points.
7. Perform the following tests using the LQR gains calculated with various different weights **Q** and **R**.
    a. Start with the pendulum hanging vertically downward ("swing-up" control). Record the angle vs. time for several different set-points.
    b. With the pendulum happily sitting at its prescribed set-point, gently tap it with your hand. What happens? Save the angle vs. time data for this test, as well.
    c. Repeat both tests above with gains obtained for various different values $\Delta\theta_{max}$, ω_{max}, and τ_{max}. Which behaves better and why?

Data Analysis and Deliverables

Using LaTeX or MS Word, make the following items and give them concise, intelligent captions. Make sure the axes are clearly labeled with units. Plots with multiple data sets on them should have a legend. **Additionally, write several paragraphs describing the plots/tables. Any relevant equations should go in these paragraphs.**

IMPORTANT NOTE: Add a horizontal line denoting the set-point whenever it is applicable.

1. A table containing the system parameters: λ, m, R, γ, the LQR feedback gains k_p and k_d, and the values of values $\Delta\theta_{max}$, ω_{max}, and τ_{max} you used to obtain the gains.
2. A plot of the angle vs. time for some of your tests of the open loop controller from Part III.
3. A plot of the angle vs. time for some of your tests of the LQR controller from Part IV.

Talking Points – Discuss these in your paragraphs.

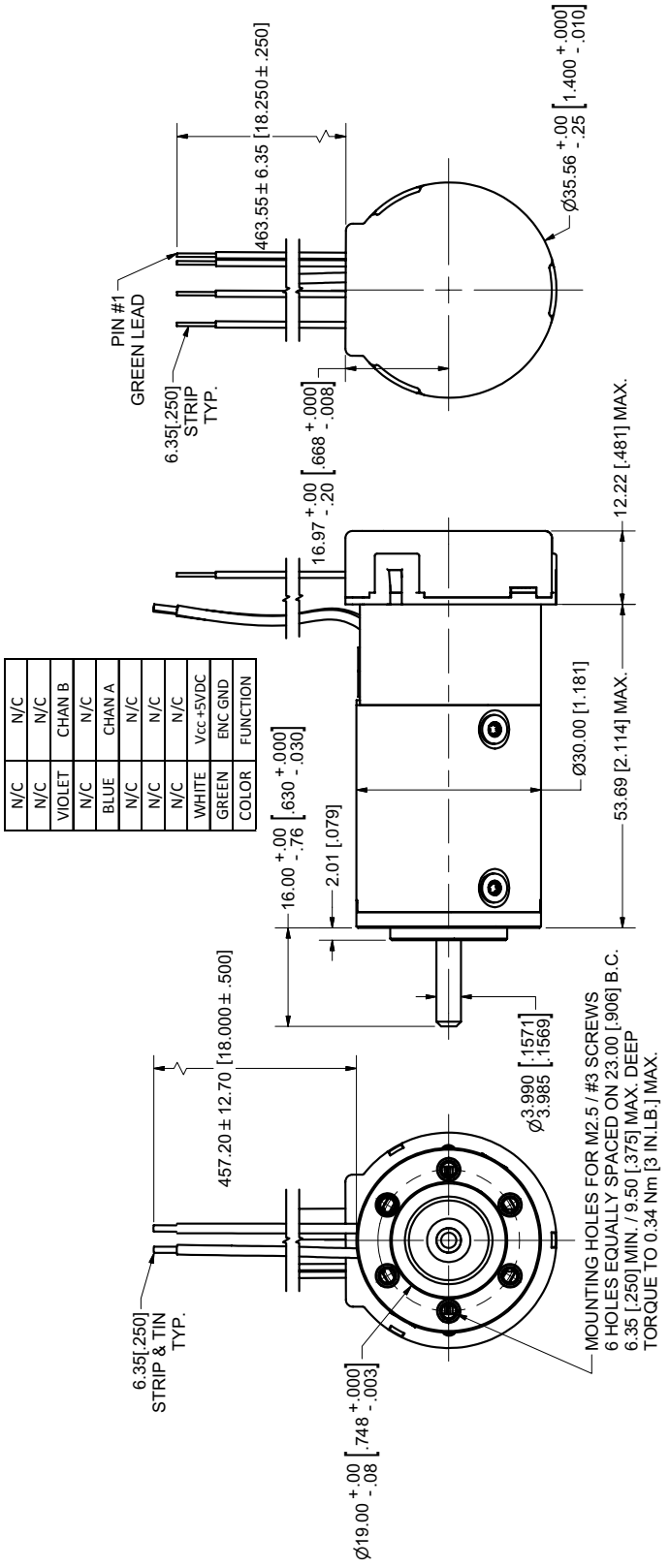
- Include any relevant equations you derived in the pre-lab assignment.
- Describe the method used to determine λ and ω_d.
- Describe some of the unexpected results you observed? What problems might arise in an industrial setting?
- What were the most challenging aspects of developing this controller?

Appendix A

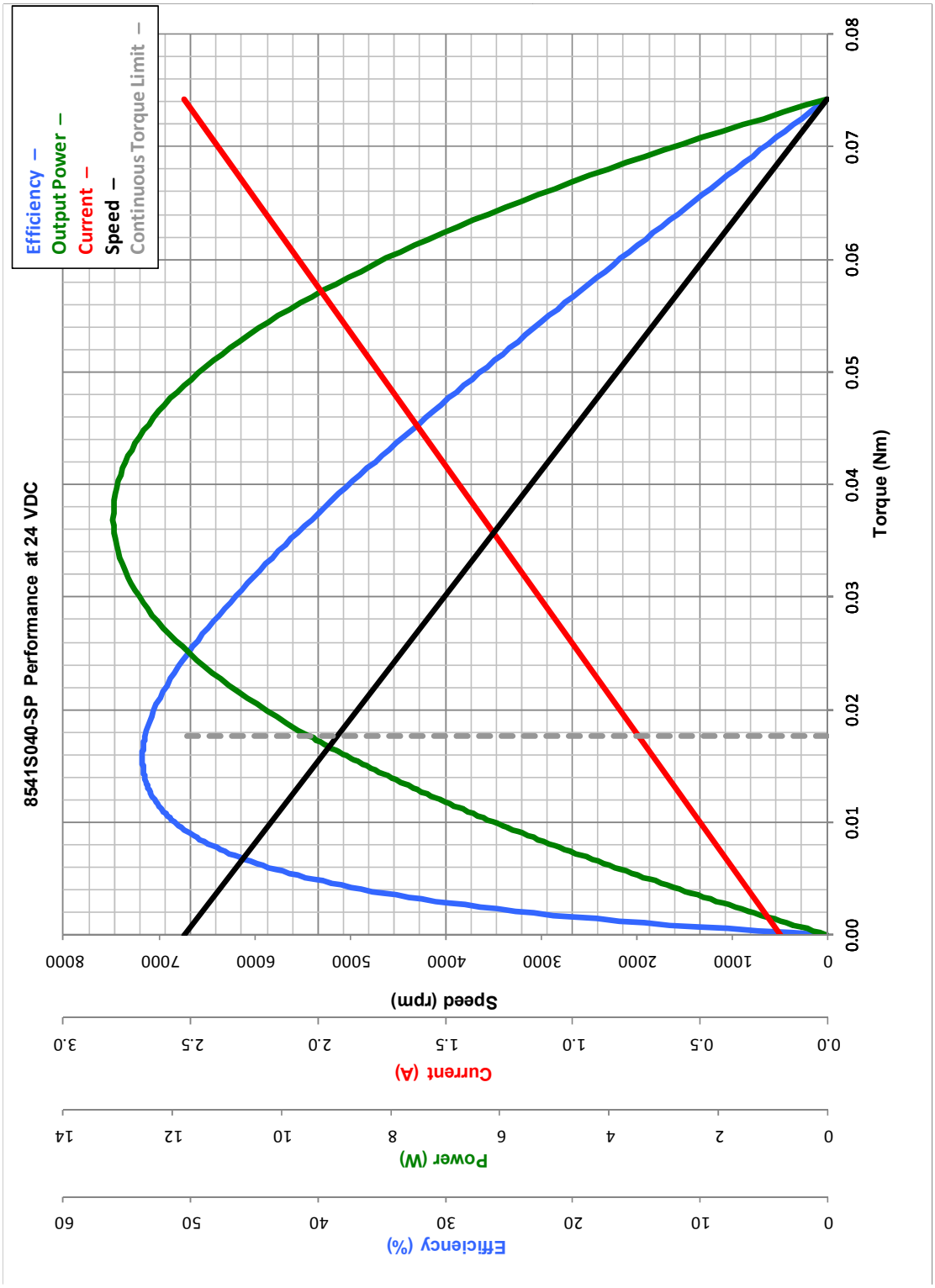
Equipment

- Standard A&C lab kit in large tote (above lab bench)
- 80-20 Rails and Motor mount from C5 lab
- Protractor
- L298N DC Motor controller breakout board
- 24V DC power Supply
- RoboGaia 3 Axis Encoder Counter Arduino Shield (SKU 00021)
- Purple, green, white, blue female-female ribbon cable
- Blue anodized motor shaft coupler
- M4 threaded rod w/ M4 nut
- 24V DC motor with encoder (Haydon-Kerk-Pittman 8541S040-SP or 8541S041-SP)
- Safety glasses

Appendix B

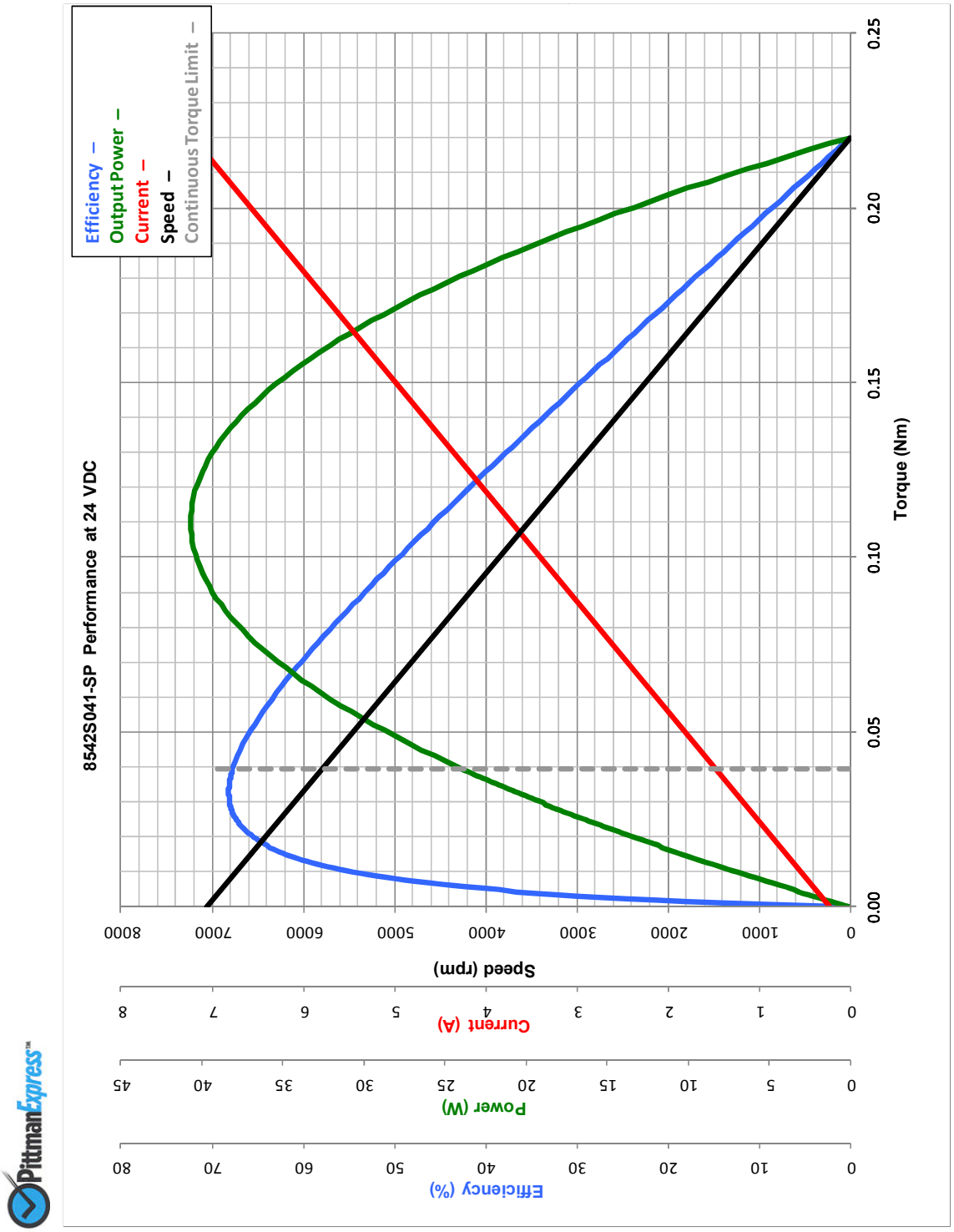

463.55 ± 6.35 [18.250 ± .250]
Ø35.56 +.00 -.25 [1.400 +.000 -.010]
PIN #1
GREEN LEAD
6.35[.250]
STRIP
TYP.
16.97 +.00 -.20 [.668 +.000 -.008]
12.22 [.481] MAX.
Ø30.00 [1.181]
53.69 [2.114] MAX.
16.00 +.00 -.76 [.630 +.000 -.030]
2.01 [.079]
457.20 ± 12.70 [18.000 ± .500]
Ø3.990 [.1571]
3.985 [.1569]
MOUNTING HOLES FOR M2.5 / #3 SCREWS
6 HOLES EQUALLY SPACED ON 23.00 [.906] B.C.
6.35 [.250] MIN. / 9.50 [.375] MAX. DEEP
TORQUE TO 0.34 Nm [3 IN.LB.] MAX.
6.35[.250]
STRIP & TIN
TYP.
Ø19.00 +.00 -.08 [.748 +.000 -.003]
N/C N/C
N/C N/C
VIOLET CHAN B
N/C N/C
BLUE CHAN A
N/C N/C
N/C N/C
N/C N/C
WHITE Vcc +5VDC
GREEN ENC GND
COLOR FUNCTION

Appendix C



All values are nominal at 25°C. Peak torque and peak current are theoretical values. Curves are shown for reference only. Visit www.pittman-motors.com.

Appendix D


8542S041-SP Performance at 24 VDC
Efficiency –
Output Power –
Current –
Speed –
Continuous Torque Limit –
Torque (Nm)
0.00
0.05
0.10
0.15
0.20
0.25
Speed (rpm)
0
1000
2000
3000
4000
5000
6000
7000
8000
Current (A)
0
1
2
3
4
5
6
7
8
Power (W)
0
5
10
15
20
25
30
35
40
45
Efficiency (%)
0
10
20
30
40
50
60
70
80
All values are nominal at 25°C. Peak torque and peak current are theoretical values. Curves are shown for reference only. Visit www.pittman-motors.com.

Appendix E