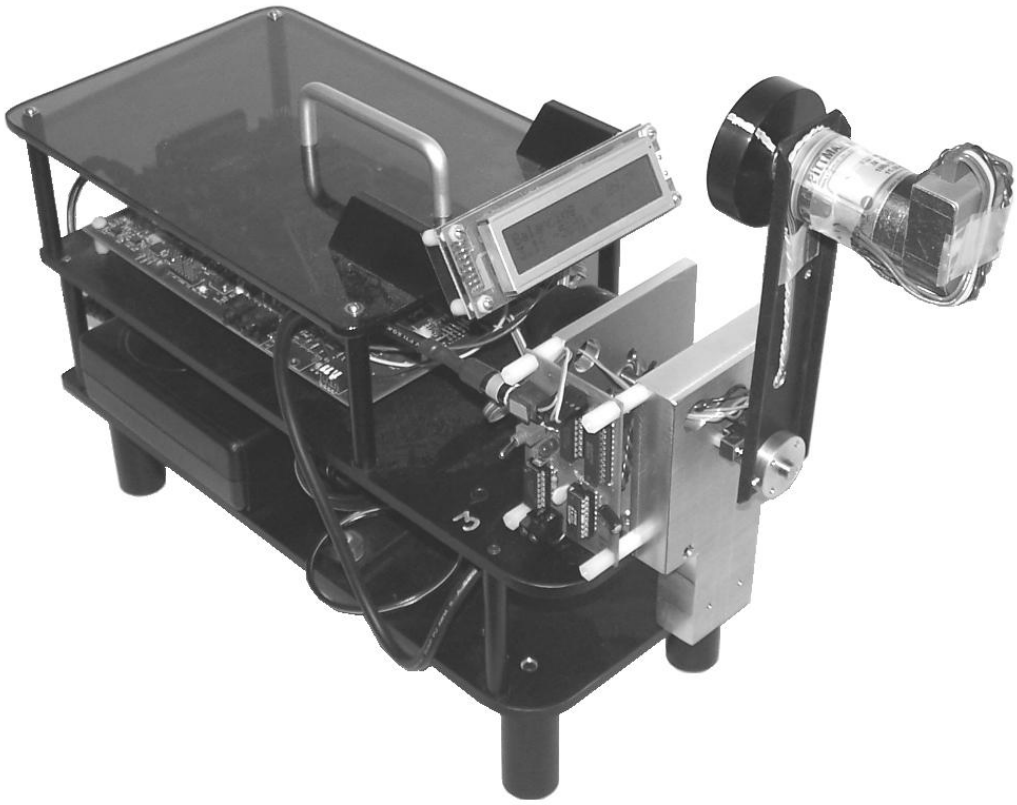
Name:
RWP #:
(painted on base of RWP)

ECE 486 Final Project:

The Reaction Wheel Pendulum



Last Edited: Spring 2026

Contents

1 Introduction 1
1.1 The Reaction Wheel Pendulum 1
1.2 Derivation of Mathematical Model 1
2 Friction Identification Using the Reaction Wheel 6
2.1 Velocity Estimation 7
2.2 PI Control for Friction Identification 10
2.3 Friction Compensation for Velocity Control 11
3 System Identification 14
3.1 Checking the Harmonic Frequency 14
3.2 Determination of Parameters ($\boldsymbol{bp}, \boldsymbol{br}$) 15
4 Stabilizing the Inverted Reaction Wheel Pendulum 17
4.1 Linearization and Controllability 17
4.2 Inverted Stabilization Using Two-State Feedback 17
4.3 Inverted Stabilization Using Three-State Feedback 19
5 Observer Design 21
5.1 Observing Four States Together 21
5.2 Decoupling and Redesigning the Observer 23
6 Up and Down Stabilizing Control 25
7 Swing-Up Control 27
Appendix A: Useful Physics Theory 28
Conversions (units of MKS) 28
Energy Equations 28
Appendix B: Implementation Notes 29
Simulink Notes 29

1 Introduction

1.1 The Reaction Wheel Pendulum

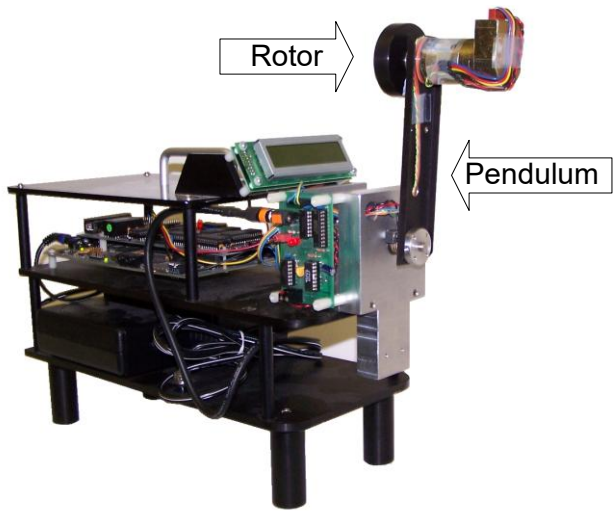


Figure 1: The Reaction Wheel Pendulum

The Reaction Wheel Pendulum (RWP), shown in Figure 1, is a simple pendulum with a rotating wheel at the end. The wheel is actuated by a 24-V, permanent magnet DC motor mounted on the pendulum. This motor can produce a torque on the wheel, causing the wheel to spin. According to Newton's *third* law, there is an equal and opposite *reaction* torque on the motor, and hence on the pendulum. This reaction torque can be used to control the motion of the pendulum. We begin by obtaining the equations of motion for the RWP. Next, control of only the reaction wheel's speed is examined. As part of this phase, we investigate counteracting the effect of friction in the motor by "friction compensation". (*Comment*: A similar identification technique was used in Lab 5 to test frictionless motor.) Finally control of the complete RWP is considered.

1.2 Derivation of Mathematical Model

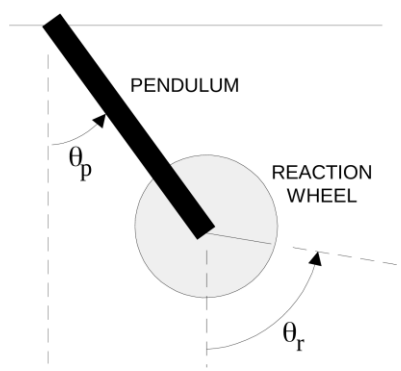


Figure 2: Schematic Diagram of the Reaction Wheel Pendulum

The first step in any control system design problem is to develop a mathematical model of the system to be controlled. Nonlinear models will first be derived using the Lagrangian approach. These models will later be linearized, and the linear models will be used to design control strategies.

A schematic diagram of the RWP is shown in Figure 2. We have chosen the angles as in Figure 2 because it is natural to use gravity to line up the pendulum hanging down. The angle θ_p is the angle of the pendulum arm measured *counterclockwise* from the vertical when facing the system, and θ_r is the wheel angle measured likewise.

The RWP is provided with two optical encoders. One encoder is attached to the fixed mounting bracket with its shaft attached to the pendulum link. It thus provides a measure of the relative angle between the pendulum and the fixed base. The other encoder is attached to the motor fixed at the end of the pendulum. Its shaft is attached to the rotating reaction wheel and thus provides the relative angle between the pendulum and wheel. Their values are initialized to zero at the start of every experiment. (*Comment*: That's why when you run your control later in Chapter 4, 6, and 7, make sure the pendulum is initialized at its resting position with the motor at the bottom.)

Looking at Figure 2, note that θ_r is the angle of the reaction wheel measured from the vertical axis. Therefore $\theta_p = \varphi_p$, $\theta_r = \varphi_p + \varphi_r$ where the encoder angles for the pendulum and rotor are ϕ_p and ϕ_r.

$$\begin{aligned} \theta_p &= \phi_p \\ \theta_r &= \phi_p + \phi_r \end{aligned} \tag{1}$$

We defined θ_r in this fashion in order to simplify the kinetic energy equations which now makes kinetic energy about θ_p and θ_r independent of each other.

A convenient way to derive equations of motion for electromechanical systems is the Lagrangian method. The Lagrangian method allows one to deal with scalar energy functions rather than vector forces and accelerations as in the Newtonian method and is, in many cases, simpler.

The RWP has two degrees of freedom. We take as generalized coordinates the angles θ_p of the pendulum and θ_r of the rotor as shown in Figure 2. We also introduce the following variables:

m_p	mass of the pendulum and motor housing/stator
m_r	mass of the rotor
m	combined mass of rotor and pendulum
J_p	moment of inertia of the pendulum about its center of mass
J_r	moment of inertia of the rotor about its center of mass
ℓ_p	distance from pivot to the center of mass of the pendulum
ℓ_r	distance from pivot to the center of mass of the rotor
ℓ	distance from pivot to the center of mass of pendulum and rotor
k	torque constant of the motor
i	input current to motor

(*Comment*: Please pay close attention to the three ℓ's above, and you can draw a picture to visualize them if necessary or label them in Figure 2.)

We also introduce the quantity

1.2.1 Lagrange's Equations

The Lagrangian method begins by defining a set of *generalized coordinates* $q_1, q_2, \ldots, q_n$, to represent an n-degree-of-freedom system. These generalized coordinates are typically position coordinates (distances or angles).

Next, compute the *kinetic energy* K, and the *potential energy* V in terms of these generalized coordinates. Typically, potential energy is only a function of the generalized coordinates, but kinetic energy is a function of the generalized coordinates and their derivatives.

In a multi-body system, the kinetic and potential energies can be computed for each body independently and then added together to form the energies of the complete system. This is an important advantage of the Lagrangian method and works because energy is a scalar-valued function, as opposed to a vector-valued function.

Once the kinetic and potential energies are determined, the *Lagrangian*, $L(q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n)$, is then defined as the difference between the kinetic and potential energies. The Lagrangian is therefore a function of the generalized coordinates and their derivatives.

$$L = K - V$$

Finally, it can be shown that the equations of motion all have the form

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} = \tau_k \qquad k = 1, \ldots, n \tag{2}$$

The variable τ_k represents the generalized force (or torque) in the q_k direction. These equations are called *Lagrange's Equations* and have the remarkable property of remaining invariant with respect to arbitrary changes of coordinates.

$$J = J_p + m_p \ell_p^2 + m_r \ell_r^2 \tag{2}$$

to represent the moment of inertia with respect to θ_p, and note the relationships

$$m = m_p + m_r \quad , \tag{3}$$

$$m\ell = m_p \ell_p + m_r \ell_r \quad . \tag{4}$$

(*Comment*: J is the total inertia when considering a non rotating rotor attached to the end of the pendulum as an object. You treat rotor plus pendulum as an overall structure.)

1-a Write down the equations for the kinetic energy K and potential energy V of the RWP. The kinetic energy of the system is the sum of the kinetic energies of each degree of freedom. How many degrees of freedom does the RWP have? *(Re-read section 1.2 above and see Appendix A for help with the physics if you're stuck. If you would like more details, check out Wikipedia on Lagrangian Mechanics.*)

Derive kinetic energy and potential energy with respect to the generalized coordinates θ_p and θ_r.

KE_{θ_p} =

PE_{θ_p} =

KE_{θ_r} =

PE_{θ_r} = 0

1-b Write Lagrange's equations (Equation (2)) for this system to show that the equations of motion for the reaction wheel pendulum are given by Equation (6) below. Here $q_1 = \theta_p$, $\dot{q_1} = \dot{\theta_p}$, $q_2 = \theta_r$, $\dot{q_2} = \dot{\theta_r}$. The torque produced by the motor results in a torque τ acting on the rotor and $-\tau$ acting on the pendulum. Use the relation $\tau = ki$ for motor torque.

Express the equations using three parameters: ${\omega_{np}}^2 = \frac{mg\ell}{J}$, $\frac{k}{J}$, and $\frac{k}{J_r}$. *(Notice that ω_{np} is the frequency of small oscillations of the system around the hanging position.)*

$K = KE_{Total}$ =

$V = PE_{Total}$ =

$L = K - V$ =

Equations of Motion =

Your final representation should be:

$$\begin{cases} \ddot{\theta}_p + \omega_{np}{}^2 \sin\theta_p = -\frac{k}{J} i \\ \ddot{\theta}_r \qquad\qquad\qquad = \frac{k}{J_r} i \end{cases}. \tag{5}$$

So far we have ignored friction. The mass on the pendulum is large enough that the friction on the pendulum link can be ignored. However, there is a significant amount of friction on the rotor link (mostly due to motor friction). Fortunately, the rotor is attached directly to the motor, allowing us to compensate for friction. The motor current i is generated by a pulse width modulation system, which is controlled from the computer. Due to current feedback, the current is proportional to the control command u from the computer. The control variable used in the computer is scaled so that 10 units correspond to maximum current. Therefore we can write

$$ki = k_u u, \quad |u| \leq 10. \tag{6}$$

We assume the friction is a function of the rotor speed $F(\omega_r)$. We will model friction in command units (units of 'u'). Applying Equation (6),

$$\begin{cases} \ddot{\theta}_p + \omega_p{}^2 \sin\theta_p = -\frac{k_u}{J}(u + F(\dot{\theta}_r)) \\ \ddot{\theta}_r \qquad\qquad\qquad = \frac{k_u}{J_r}(u + F(\dot{\theta}_r)) \end{cases}. \tag{7}$$

Finally, to clear up the clutter, we can also introduce variables $a = \omega_{np}{}^2 = \frac{mg\ell}{J}$, $b_p = \frac{k_u}{J}$, and using (7), $b_r = \frac{k_u}{J_r}$ becomes:

$$\begin{cases} \ddot{\theta}_p + a \sin\theta_p = -b_p(u + F(\dot{\theta}_r)) \\ \ddot{\theta}_r \qquad\qquad\qquad = b_r(u + F(\dot{\theta}_r)) \end{cases}. \tag{8}$$

This is a satisfactory representation of the RWP. Before we begin its control, however, let us take a detour and consider speed control of a DC motor. This will allow us to model and compensate for friction.

2 Friction Identification Using the Reaction Wheel

In this section we will identify the friction of the RWP's DC motor. Recall that we made the assumption that the pendulum link has negligible friction. To help collect data for finding the friction coeficients, we will design a velocity control for the motor. Since velocity information is not directly available to us, we will use an approximation of the motor's velocity.

Safety note: *When doing experiments in this part, remember a couple of things. Even though the motor controller has a safety mechanism to prevent the motor from spinning extremely fast, the combination of spinning fast and running for a long time will heat up or even burn out the motor. Be prepared to shut off the controller either when the system runs long enough for you to collect data or when it becomes unstable, either through the Simulink model or the switch on the amplifier board.* ***Consider 200 rad/s as fast****.*

In past labs we have analyzed the dynamics of a DC motor, using the armature voltage as the input. Here however, we select the armature current i as input. Then the motor becomes merely a current-torque transducer (see Figure 3), meaning electrical

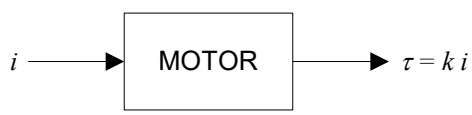


Figure 3: DC Motor Model

energy will be converted to mechanical energy.

The torque τ is applied to the reaction wheel (rotor) having moment of inertia J_r and speed (relative to the motor housing) of ω_r. There is also friction, due mainly to the motor brushes and represented as a torque τ_F. So the motion of the wheel is given by

$$J_r\dot{\omega}_r = ki - \tau_F \tag{9}$$

using the motor (rotor) speed $\omega_r = \dot{\theta}_r$ as the output. Putting it in terms we are familiar with, we get the following: (*Comment*: The $\tau_F$in Equation (10) is "positive" while the $F(\dot{\theta}_r)$ in Equation (11) is "negative".)

$$\dot{\omega}_r = \ddot{\theta}_r = b_r(u + F(\dot{\theta}_r)) \tag{10}$$

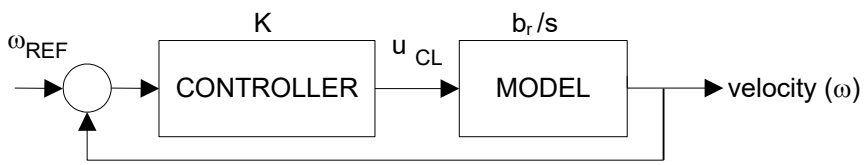


Figure 4: General Block Diagram of Velocity Controller. We assume friction is 0 here.

2-a	Using Figure 4 as a guide, design a proportional controller with a rise time of 0.2s and no steady state error. Use an input step of 100 rad/s. Assume b_r = 198 (rad/s). Simulate your controller using Simulink.

2.1 Velocity Estimation

Consider Equation $\dot{\omega}_r = \ddot{\theta}_r = b_r(u + F(\dot{\theta}_r))$ (10) and again ignore friction for now. Of course, this is an ideal model, so a few real-world issues must be dealt with.

Recall from Section 1.2, page 5, that the control input is limited to 10. That is simple enough to simulate in Simulink. *(Hint: Check out the "Saturation" block.)*

Another issue is the determination of angle from the encoder output. Think of the encoder as the Wheel of Fortune wheel; counting ticks tells you that it's turning. (There's also a provision for determining direction of spin; this is analogous to the "ticker" sounding different in either direction.) The ticks add as the angle changes. There are two issues here:

First, how does the software know where "zero" is? By convention, zero is the encoder angle when you "Start" the run.

Second, how do you determine angle from ticks? Since the motor encoder has 4000 ticks/revolution, multiply by 2π/4000 to scale to radians. (The pendulum encoder has 5000 ticks/revolution.) One other detail: the reaction wheel encoder and motor use opposite sign conventions in this setup. In other words, when a positive current is applied to the motor, it spins in a direction that the encoder calls negative. **Therefore, you will need a -1 gain before the input to the motor**.

The encoder measures position. How can the velocity $\omega = d\theta/dt$be obtained? This is done either by using a transfer function that approximates a derivative or by using a discrete version of the same. We will implement both and compare them.

A simple discrete version can be found by using Euler's method (FPE pp.167, or FPE 3rd Ed pp. 138). It states that

$$\frac{df}{dt} = \lim_{\Delta t \to 0} \frac{f(t) - f(t - \Delta t)}{\Delta t} \tag{11}$$

The discrete derivative approximation is implemented by using the "Unit Delay" block in Simulink. Even though this is a discrete approximation of the velocity, we will consider this velocity continuous for simplicity. Applying Equation (11) to our system,

$$\omega(t) \cong \frac{\theta_r(t) - \theta_r(t - \Delta t)}{\Delta t} \tag{12}$$

Note: We will be using a sample period Δt of 0.002s.

The continuous derivative approximation can be understood by looking at the frequency response of the derivative function s. We want to keep the response similar at low frequencies, but refrain from amplifying the high-frequency noise. This is accomplished by placing a pole at a sufficiently high[1] frequency, giving the transfer function

$$\frac{s}{\tau s+1}, \tag{13}$$

where $1/\tau$ is the pole location.

For your continuous derivative, use Equation (13) with $\tau = 1/50$. Now that we have a velocity estimate, we can use it as feedback.

2-b Implement your velocity approximations, and the controller designed above, in Simulink with the C2000 Microcontroller Blockset toolbox. A starter Simulink file is provided for you called part2b_starter.slx.

Procedure for copying and opening the starter file:

1. Verify that the C2000 DAQBoard is powered on and it's USB cable is plugged into your bench's Windows 11 PC.
2. Create a folder named after your NetID inside C:\matlab\ece486\. For example, if your NetID is jdoe2, create C:\matlab\ece486\jdoe2
3. Copy the folder N:\labs\ECE486\final_project\part2 into C:\matlab\ece486\<yourNetID>.
4. Start Matlab
5. Change Matlab's current working directory to C:\matlab\ece486\<yourNetID>\final_project\part2 using the Matlab command window: "cd C:\matlab\ece486\<yourNetID>\part2"

[1] *Sufficiently high:* application-specific, often selected iteratively by simulation or test runs.

6. Open the Simulink model, part2b_starter.slx, by double-clicking on it in the MATLAB Current Folder browser.
7. In part2b_starter.slx you need to select the correct COM port for the Reaction Wheel's board. In Windows 11 Device Manager check the COM port of "XDS100 Class USB Serial Port". Then in part2b_starter.slx go to "Hardware Settings" and under "Target Hardware Resources" change "External Mode" to use the COM port assigned in Device Manager and save the file "Ctrl-S".

Procedure to check feedback positive direction:
1. Before you can implement your controller, you need to scale the Reaction Wheel C2000 Block's optical encoder outputs ϕ_p and ϕ_r to radians and add a -1 gain block before the motor input. Using the scale factors given above, use two gain blocks to scale ϕ_p and $\phi_r$to radians and one more gain block to add a -1 before the motor input.
2. Use a Sum block to form θ_r.
3. Use Scope blocks to plot the values of θ_p and θ_r.
4. Run this Simulink file (Monitor and Tune) with the motor off and when time starts progressing look at the scope plots θ_p and θ_r.
5. Gently move the pendulum and motor to check if their positive directions are the same as Figure 2. If not, add a negative in the gain block to correct the direction. You will not be using θ_p or θ_r in this section 2, but you now have their positive directions verified for the remaining sections.
6. Press the "Stop Controller" Simulink button to stop your real-time code.

Implement P Controller:
1. Now implement the two velocity approximations and your proportional controller, designed 2-a, using $\phi_r$as the feedback signal. Here we are assuming the pendulum angle $\theta_p$is not moving. You can gently hold the pendulum when the controller starts moving the motor. Use a Simulink "Manual Switch" to allow your Simulink implementation to switch between using, as feedback, the two velocity approximations. Remember to click the "Enable Motor / Disable Motor" Simulink button to allow the motor to move. (The blinking blue LED indicates that the motor is enabled. Make sure to plot velocity so you can compare the two approximations.
2. Compare the simulated response in 2-a with the C2000 Microcontroller Blockset response in 2-b. What is the source of this discrepancy?

The answer to item 2 leads us to the ultimate goal of this section: friction identification and compensation.

2.2 PI Control for Friction Identification

As you know, we can counteract a constant disturbance by adding an integrator.

2-c Add an integral term to your 2-b P control implementation to create a PI controller that regulates the motor to 100 rad/s. Open part2c_starter.slx and select the correct COM port for the Reaction Wheel's board. In Windows 11 Device Manager check the COM port of "XDS100 Class USB Serial Port". Then in part2c_starter.slx go to "Hardware Settings" and under "Target Hardware Resources" change "External Mode" to use the COM port assigned in Device Manager and save the file "Ctrl-S". part2c_starter.slx shows you how to reset the integrator when the "Enable Motor / Disable Motor" button is pressed, or the manual switch is flipped. So you will be copying your P control implementation from 2-b into this part2c starter file to create your PI controller. Start with Kp equal to your 2-b value and Ki equals to this same Kp value. Run, Monitor & Tune, your model and when time starts progressing, and tune Kp and Ki until you have a response with 0.2 second rise time, less than 5% overshoot and zero steady state error. In addition to scoping the speed of the motor, use a Display block to display the control effort applied to the motor.

Because the motor has damping and static friction, the control effort to hold the motor at a desired speed is nonzero. Using this fact we can command the motor to one velocity at a time and record the control effort needed to regulate the motor at that speed. This control effort is the amount of effort the motor needs to overcome friction.

Saying it another way, consider Equation $\dot{\omega}_r = \ddot{\theta}_r = b_r(u + F(\dot{\theta}_r))$ (10) at steady-state, with friction included. Remember $F(\dot{\theta}_r)$ is negative.

$$\dot{\omega}_r = \ddot{\theta}_r = b_r(u + F(\dot{\theta}_r)) = 0 \tag{14}$$

We see that for non-zero friction, the control effort will be non-zero as well. In fact, the value of friction for any velocity is merely the steady-state control effort for a setpoint of that velocity. In other words,

$$u = -F(\dot{\theta}_r) \tag{15}$$

2-d Run the motor at various speeds (i.e. vary the setpoint of the PI controller to 150, 100, 75, 50 and 25 radians/second) and record the steady-state control effort for each speed. Do this for both positive (counter-clockwise) and negative (clockwise) velocities. Fit this to two lines *(hint: see MATLAB command "polyfit")*, and note the static, c, and dynamic, b, frictions **in both directions** (they will probably differ). Write down the expression for $F(\dot{\theta}_r)$.

Now that we have characterized friction, we can explore a way of negating its effects on our system.

2.3 Friction Compensation for Velocity Control

Friction affects all systems, and can add to or modify system dynamics, bring in noise, decrease resolution, and introduce offsets. We have implemented one method of dealing with offsets introduced by Coulomb friction – integral control. However, the dynamics of the system are still affected by friction. By considering friction as a linear function (a velocity gain and offset for each direction), we see another way of dealing with it. Since we now know the value of friction (in control units; see Equation (6) and the associated discussion on page 5) for any speed, we can let Simulink "adjust" for friction by counteracting its value as a function of speed (see Figure 5).

2-e Implement friction compensation.

Start with the PI control you developed in 2-c.

Typing “fricblocks” at the Matlab prompt opens the block that you used in Lab 5 to calculate asymmetrical friction. Enter the c+, b+, c-, b- values you found in 2-d.

Design your Simulink model using two more manual switches so that you can switch off your PI control and just apply a constant zero, and also a switch that switches on or applies zero for the friction compensation part. (Looking back at the Lab 5 friction compensation Simulink implementation may help you with the placement of these manual switches.)

Start out with your PI control switched off and simply implement friction compensation. What do you expect will happen? Reason out what you expect to see, then gently hold the motor and manually spin the motor in either direction and see what happens.

Tune your friction b and c values in both directions in order that when you manually spin the motor it keeps on spinning in the same direction for at least 20 seconds. Most of the time you will only need to adjust b+ and b-. If your motor does not move when first enabled your c+ and c- are more than likely good values. Use these tuned b+, c+, b-, c- values as your friction values.

Now switch on just your P controller by zeroing the Ki gain. A proportional control should now regulate the system to (or very close to) the desired velocity. Is integral control still needed, or is proportional control sufficient? Answer this question by setting Ki back to the value you tuned in 2-c. Observe the effects of PI control with friction compensation as you use your finger to add a very small additional damping by letting the flywheel rub against your finger as it spins.

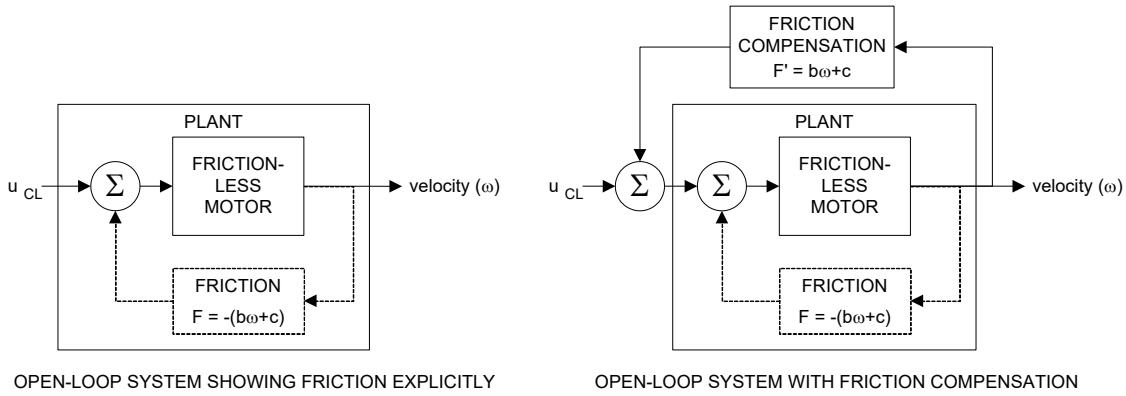


Figure 5: Functionality of Friction Compensation

Now that friction is well modeled, we can return to the overall Reaction Wheel Pendulum. First, we will do some System Identification and Model Verification, then finally delve into control.

3 System Identification

We can now determine the parameters of the Reaction Wheel Pendulum (RWP). Here we set up the RWP in the standard configuration. The parameters can be determined from physical construction data and by direct experiments on the system. It is useful to combine both methods to find all the parameters, and to make cross-checks. It also verifies that our mathematical model is reasonable.

By measuring the dimensions of the components, weighing them, and computing moments of inertia using simplified formulas we find:

m_p = 0.2164 kg

m_r = 0.0850 kg

J_p = 2.23310^{-4} kgm^2

J_r = 2.49510^{-5} kgm^2

ℓ_p = 0.1173 m

ℓ_r = 0.1270 m

For you to find:

J	=	kgm^2
m	=	kg
ℓ	=	m
ω_{np}	=	rad/s
ω_{np}'	=	rad/s

measured frequency of oscillation:

$\omega_{np}'_{meas}$= rad/s

3-a Use the relations and definitions given in Section 1.2 to get values for J, m, ℓ, and ω_{np}. Also find ω_{np}'. (Defined by Equation (16) below)

In order to verify the natural frequency, we can do a free swing test (below).

3.1 Checking the Harmonic Frequency

3-b Use the provided part3.slx file (procedure below). Do not flip on the motor amplifier for this run, and start the Simulink file with the Pendulum hanging in the down position. Once the Scope plot for θ_p and the Scope for ϕ_r start plotting, move the pendulum to approximately 90° and let it swing freely. When the wheel stays stuck to the pendulum, i.e. the encoder reading ϕ_r is constant, determine the frequency of oscillation ($\omega_{np}'_{meas}$).

Procedure:

1. Create a folder named after your NetID inside C:\matlab\ece486\ if you haven't done so already. For example, if your NetID is jdoe2, create C:\matlab\ece486\jdoe2
2. Copy the folder N:\labs\ECE486\final_project\part3 into C:\matlab\ece486\<yourNetID>.
3. Start Matlab

4. Change the current working directory to C:\matlab\ece486\<yourNetID>\final_project\part3 using the Matlab command window: "cd C:\matlab\ece486\<yourNetID>\part3"
5. Open the Simulink model by double-clicking on it in the MATLAB Current Folder browser.
6. In part3.slx you need to select the correct COM port for the Reaction Wheel's board. In Windows 11 Device Manager check the COM port of "XDS100 Class USB Serial Port". Then in part.slx go to "Hardware Settings" and under "Target Hardware Resources" change "External Mode" to use the COM port assigned in Device Manager and save the file "Ctrl-S".
7. Run, Monitor & Tune, this model and when time starts progressing you can record data using the Simulink scopes.

This measured frequency is different from ω_p because the rotor is contributing to the moment of inertia. The quantity you measured is actually

$$\omega_{np}{}' = \sqrt{\frac{mg\ell}{J+J_r}} \tag{16}$$

Compare the experimental value with the theoretical value, computed from the parameters (all known).

Notice how ω_{np} and ω_{np}' are very close in value. For that reason, we can use ω_{np}' as our identitied value for ω_{np}.

Notice also that the decay in the swing amplitude is slow. On the other hand, if the rotor is excited with the maximum current, and then the current is removed, it takes only a few seconds to come to complete rest. In both cases, friction is the only deceleration force (for the pendulum, consider conservation of energy and for the rotor, apply Newton's first law of motion). This helps to validate our assumption that the friction in the pendulum link is negligible, but friction in the motor is not.

3.2 Determination of Parameters (b_p, b_r)

There are two parameters that we still don't know. These are b_p and b_r. By examining Equation (17), we see a way to find b_p and b_r.

$$\begin{cases} \ddot{\theta}_p + a\sin\theta_p = -b_p(u + F(\dot{\theta}_r)) \\ \ddot{\theta}_r \qquad\qquad\quad = \quad b_r(u + F(\dot{\theta}_r)) \end{cases} \tag{17}$$

Our sensors directly measure θ_p,θ_r. We developed an approximation for $\dot{\varphi}_r$ in section 2.1 and you can use the same velocity approximation for $\dot{\theta}_p$and $\dot{\theta}_r$. If we estimate $\ddot{\theta}_p$and $\ddot{\theta}_r$by differentiating again and use our friction model from section 2.3 to replace $F(\dot{\theta}_r)$, the only unknowns in the top equation of (17) are b_p and b_r. Solving for them is trivial, assuming we can find the second derivative. Alas, Figure 6 shows that the first derivative is noisy, and the second derivative is worthless. A better approach would find a polynomial fit for θ_p,θ_r, and differentiate this fit to get clean values for $\dot{\theta}_p$,$\dot{\theta}_r$,$\ddot{\theta}_p$and $\ddot{\theta}_r$.

These clean values can then be put into Equation (17) to solve for the torque constants. In practice, a cubic fit of the RWP response to a step input from rest gives good results.

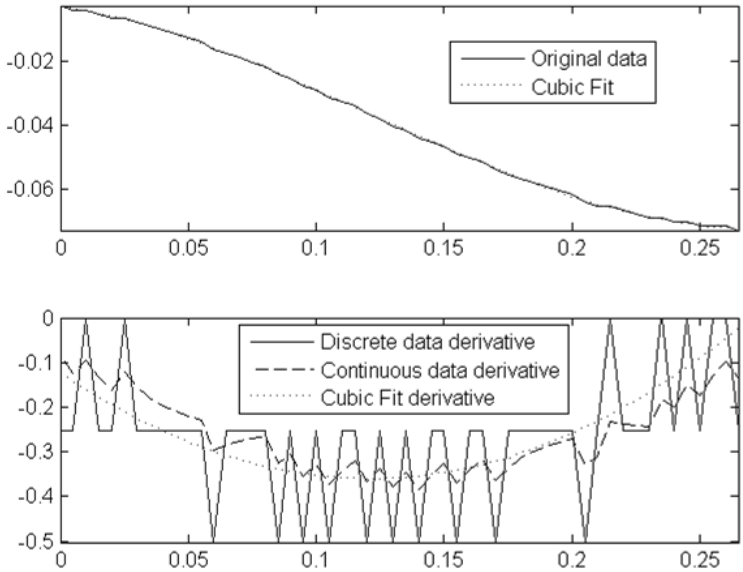


Figure 6: Derivative approximations add significant noise

3-c Use the method described above to find b_p and b_r. The given Simuink file, part3.slx, collects the needed data in the scope labeled "Theta_p, Theta_r, & control effort". Each time before running the Simulink file, bring the pendulum up to about 90° and let it swing freely until it stops. This lets the pendulum find its own zero position and usually results in a better place to start the system for the identification run. Now run (Monitor and Tune) the Simulink file. When the realtime code has been loaded and running, time "t=" will start increasing in the bottom right corner of your Simulink model. At this point flip the motor amp switch on and then click the "Enable Motor / Disable Motor" button in Simulink model. A step input will be given to the motor when this Simulink button is clicked. A partial m-file is provided for you on the lab website that will help you use this collected data to find b_p and b_r The given Simulink file uses a step of magnitude 5 for u.

Another way to determine b_p and b_r (and thus k_u), uses equation (6). The i is determined for the maximum input u of 10. Properties of the motor and controller tell us the value of $i_{\max}$ and k, giving

$b_p = 1.08$

$b_r = 198$

Your results should approximately agree (within 30%). For the remainder of this lab, use your identified values for ω_{np}, b_p and b_r.

4 Stabilizing the Inverted Reaction Wheel Pendulum

4.1 Linearization and Controllability

The Reaction Wheel Pendulum (RWP) has equations of motion, ignoring friction, given by

$$\begin{cases} \ddot{\theta}_p + a\,sin\,\theta_p = -b_p u \\ \ddot{\theta}_r \qquad\qquad\quad = \quad b_r u \end{cases} \tag{18}$$

> 4-a Linearize this system about the equilibrium position of θ_p = π. Write the state-space model for this system in the blanks provided in Equation (19) and check for controllability from the single input u. What are the system's open-loop poles (or eigenvalues). *Note:* Your new state variables are delta-angles, where $\delta\theta_p$= $\theta_p - \pi$ and $\delta\theta_r$= $\theta_r - 0$.

$$\begin{array}{ccccccc} \dot{\mathbf{x}} & = & \mathbf{A} & \mathbf{x} & + & \mathbf{B}\,u \end{array}$$

$$\begin{bmatrix} \quad \\ \quad \end{bmatrix} = \begin{bmatrix} \qquad & \qquad \\ \qquad & \qquad \end{bmatrix} \begin{bmatrix} \delta\theta_p \\ \dot{\theta}_p \\ \delta\theta_r \\ \dot{\theta}_r \end{bmatrix} + \begin{bmatrix} \quad \\ \quad \end{bmatrix} u \tag{19}$$

The system should be controllable; otherwise we would need to add another actuator to be able to complete the project.

4.2 Inverted Stabilization Using Two-State Feedback

When designing a State Feedback controller ($u = -kx$) we usually talk about placing the poles of the closed loop system. These poles need to be stable and responsive enough to make the system respond in a desired fashion. But where to place the poles, especially when you are first working with a new system, is a good question. For the reaction wheel pendulum we only care about the pendulum stabilizing in its upright position. The angle of the motor does not matter. (We will find soon that the velocity of the motor is important though.) So at first let's just design a 2 state feedback controller that only uses the pendulum's feedback. This way the linearized closed loop system will be 2nd order and you can use your knowledge of the ideal second order system's characteristic equation $s^2 + 2\zeta\omega s + \omega^2$ to choose two poles. Here we are only considering the top left 2X2 portion of your linearized A matrix and the first two entries of the B matrix, therefore a 2nd order system.

4-b Design a 2 state-feedback controller for the RWP using the MATLAB command *place*. Choose poles by trial and error by finding the roots of $s^2 + 2\zeta\omega s + \omega^2$ given the constraints:
1. $\omega > \omega_{np}$ so the closed loop dynamics are faster than the open-loop dynamics.
2. $\zeta < 1/\sqrt{2}$.
3. Keep both K values less than 400.
Simulate your system starting with part4simulation_starter.slx which has the nonlinear "Reaction Wheel Block Diagram Model" (See Appendix B) and "Reaction Wheel Animation" blocks. For this simulation don't bother estimating velocity; just use the exact states from the model block. Also note that your control effort should connect to the given Saturation Block so that control effort is saturated to the maximum motor input -10 to 10.
Simulate ① Initial Condition (IC) deviation ($\delta\theta_p$ or $\delta\dot{\theta}_p$ nonzero),
② a pulse (simulating a tap) disturbance input to the pendulum arm (τ_p) with duty cycle of 5% and period of 4 seconds,
③ a constant disturbance input to the pendulum arm.
Is the response satisfactory (i.e. stable and fairly fast)? Now look at rotor velocity. Do you see any problems?

Procedure:
1. Create a folder named after your NetID inside C:\matlab\ece486\ if you haven't done so already. For example, if your NetID is jdoe2, create C:\matlab\ece486\jdoe2
2. Copy the folder N:\labs\ECE486\final_project\part4 into C:\matlab\ece486\<yourNetID>. This will copy all the files you need for part 4.
3. Start Matlab
4. Change the current working directory to C:\matlab\ece486\<yourNetID>\final_project\part4 using the Matlab command window: "cd C:\matlab\ece486\<yourNetID>\part4"
5. Open part4simulation_starter.slx by double-clicking on it in the MATLAB Current Folder browser.

Note: Remember that the controller uses delta states, whereas the nonlinear model outputs absolute states.

Comment: In order to fill out the first columns in the below table, you need to test IC deviations for both $\delta\theta_p$ and $\delta\dot{\theta}_p$ But you only test one at a time, i.e., for example, when you test IC deviation, add small deviation to $\delta\theta_p$ but keep zero deviation for $\delta\dot{\theta}_p$. Then you test $\delta\dot{\theta}_p$ similarly.

Table 1: Robustness Comparisons

Test condition	**Two-State Feedback (4.2)**		**Three-State Feedback (4.3)**		**Observer (5.1)**	
① Max IC deviations	$\delta\theta_p$	$\delta\dot{\theta}_p$	$\delta\theta_p$	$\delta\dot{\theta}_p$	$\delta\theta_p$	$\delta\dot{\theta}_p$
② Max pulse						
③ Max constant disturbance						

As you can see, the rotor velocity stays constant at steady-state without any disturbances. However, with a constant disturbance, however small, the motor undergoes a constant acceleration to counteract it, which causes the velocity to increase without bound. Since we have a bound on velocity (there is *always* a bound on velocity!), this is not practical for implementation. Therefore we must feedback the rotor velocity information.

This may raise a question: In simulation, if a constant u can cause velocity to become arbitrarily large, why can't that happen in our system? Because as velocity increases, friction increases, and (u-F) decreases until friction effectively "cancels out" u!

4.3 Inverted Stabilization Using Three-State Feedback

Consider the eigenvalues of the 4-state state-space model you found in Question 4-a. The zero eigenvalues represent the rotor position and velocity. Our goal is to pull the velocity eigenvalue into the LHP, but leave the position eigenvalue alone. This will **stabilize** the rotor velocity, while still **ignoring** its position.

4-c Using the same constraints as 4-b, design a 3-state feedback controller with θ_p, $\dot{\theta}_p$, and $\dot{\theta}_r = \dot{\varphi}_r + \dot{\varphi}_p$ as the feedback and simulate in Simulink. Again for this simulation do not estimate the velocity, just use the exact states from the model block.[2] Place the $\dot{\theta}_r$ eigenvalue between the other two LHP poles. *(Hint: Use* MATLAB *"place" – just keep the fourth pole at zero. This makes the fact that we're ignoring θ_r evident.)* Simulate conditions ①, ②, and ③ from 4-b again. Record in Note: Remember that the controller uses delta states, whereas the nonlinear model outputs absolute states.

Comment: In order to fill out the first columns in the below table, you need to test IC deviations for both $\delta\theta p$ and $\delta\ \theta p$ But you only test one at a time, i.e., for example, when you test IC deviation, add small deviation to $\delta\theta p$ but keep zero deviation for $\delta\ \theta p$. Then you test $\delta\ \theta p$ similarly.

[2] We're not asking you to estimate the velocity in the simulation because we want to start the pendulum around π radians. This requires you to input the correct initial value into the velocity estimate transfer functions by subtracting θ_p's initial condition from θ_p and θ_r, which would make your Simulink diagram more complicated and harder to manage.

Table 140-c Using the starter file part4_implement_starter.xls implement this controller on the actual RWP. Open part4_implement_starter.slx and select the correct COM port for the Reaction Wheel's board. In Windows 11 Device Manager check the COM port of "XDS100 Class USB Serial Port". Then in part4_implement_starter.slx go to "Hardware Settings" and under "Target Hardware Resources" change "External Mode" to use the COM port assigned in Device Manager and save the file "Ctrl-S". "Monitor and Tune" when you are ready to run and remember to pressy the "Enable Motor / Disable Motor" Simulink button to allow the motor to spin.
Here you will have to estimate velocity using the method you chose in 2-b. If the RWP is too sensitive to stay balanced, check the rotor velocity. If the RWP is spinning up to a high velocity before falling, then your $\dot{\theta}_r$ feedback gain may be too small. You can adjust the $\dot{\theta}_r$ gain until you get a satisfactory response or you can choose faster closed loop poles for your controller to achieve a good response. Show your TA the working controller. When the RWP is successfully stabilized, you should see limit cycle behavior[3].

4-d Add to your 4-c implementation friction compensation (which you designed in Question 2-e) and observe the change in behavior.

$$u = -k * x + friction_compensation$$

Demonstrate it to your TA.

We will next explore the observer's approach.

[3] Persistent (but not necessarily precisely) repeating behavior that does not die out is called *Limit Cycle Behavior*.

5 Observer Design

We will now design an observer for the Reaction Wheel Pendulum (RWP) to estimate the states of the full state feedback controller we designed previously. The observer will estimate both velocities of the system. And since we're designing a full-order observer, it will also "estimate" both positions.

Look back at your full-state feedback design; you pulled all of the open-loop poles except the θ_r pole into the left half plane. When we design an observer, however, we must place *all* of the observer poles in the left half plane; our criteria being: "significantly farther" than the desired closed-loop poles. See Figure 7 for an illustration of this (not to scale, and relative pole locations may vary by design).

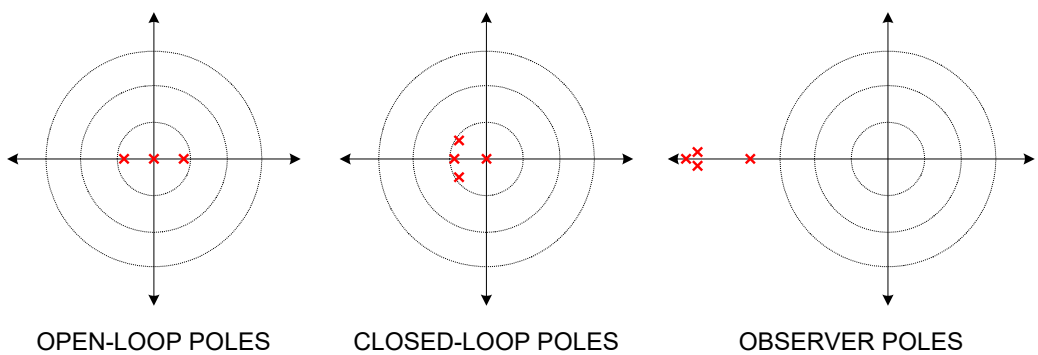


Figure 7: Illustration of Pole Locations

5.1 Observing Four States Together

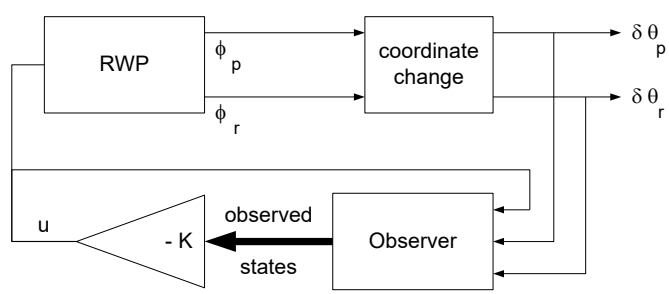


Figure 8: Block Diagram of System with Observer

Figure 8 shows the structure of our closed-loop system with observer-based control. In Simulink, the observer block can be modeled as a State-Space block. However, in order to do so, we need to define the A, B, C, and D matrices (where the state equation is defined as $\{\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u, \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}u\}$).

The standard differential equation for an observer is

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}u + \boldsymbol{L}(\boldsymbol{y} - \hat{\boldsymbol{y}})$$

$$\hat{\boldsymbol{y}} = \boldsymbol{C}\hat{\boldsymbol{x}} + \boldsymbol{D}u \tag{20}$$

Keeping in mind that we want $\hat{\boldsymbol{x}}$ as the states, $\hat{\boldsymbol{x}}$ as the outputs, and both delta-angles (collectively called $\boldsymbol{y}$, where $\boldsymbol{y} = [\delta\theta_p\ ,\ \delta\theta_r]^{\mathrm{T}}$) as well as u as the input, Equation (20) can be manipulated to give

$$\begin{cases} \dot{\hat{\boldsymbol{x}}} = (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{C})\hat{\boldsymbol{x}} + [\boldsymbol{B}\ \ \boldsymbol{L}]\begin{bmatrix} u \\ \boldsymbol{y} \end{bmatrix} \\ \boldsymbol{z} = \boldsymbol{I_4}\hat{\boldsymbol{x}} \end{cases} \tag{21}$$

where **z** represents the output of the observer. **You should be comfortable going from Equation (20) to Equation (21)**. (***Hint:*** when doing matrix algebra, always check dimensions!)

In particular, what is **C**? From Equation (20), we see that **y** and $\hat{\boldsymbol{y}}$ must have the same dimension, and, in order for their difference to be meaningful, must represent the same physical phenomena (e.g., subtracting a velocity from an angle is meaningless).

since **y** = **Cx**, $\quad \mathbf{C} = \begin{bmatrix} \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad \end{bmatrix}$ *(fill in your* **C** *matrix)*

Also, why $\boldsymbol{I_4}$? ($\boldsymbol{I_4}$represents the 4 × 4 identity matrix) Because we want to output all of our states **individually**. If we used a scalar z and defined $z = \hat{x}_1 + \hat{x}_2 + \hat{x}_3 + \hat{x}_4$, then instead of having more information from the observer, we would actually have less!

5-a	Using the MATLAB *place* command to find a L matrix that places the observer poles significantly farther than your closed-loop poles (as designed in Section 4.3). Five to ten times faster is a good distance. Keep these poles near or on the real axis. Also note that the 'place' command cannot solve for repeated roots. Check the observer's eigenvalues, eig(A-L*C) using MATLAB *eig* to double check you placed the poles correctly.

5-b	Simulate your observer design in Simulink. You can start with your 3-state simulation you made in part 4. Test and record the same things you tested in Question 4-b. How does this controller compare to the three-state feedback controller? Now vary the nonlinear model's parameters slightly. Does the controller still work?

The extremely high sensitivity of this controller to variations in the plant may surprise you, but in the next section we will explore the RWP model in more depth in order to understand why this is the case, and find a way to modify your controller in order to make it less sensitive to plant variations.

5.2 Decoupling and Redesigning the Observer

Recall that the state-space model for our system has the form

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ a & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ -b_p \\ 0 \\ b_r \end{bmatrix} u \tag{22}$$

We see that **A** has a specific form – the top right four values and the bottom left four values are zero. There is a special term for that form – *block diagonal*. Let us take a small digression and explore the implications of **A** being in block diagonal form. If we rewrite the system as

$$\begin{bmatrix} \dot{\boldsymbol{x}}_{1,2} \\ \dot{\boldsymbol{x}}_{3,4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & 0 \\ 0 & \boldsymbol{N} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{1,2} \\ \boldsymbol{x}_{3,4} \end{bmatrix} + \begin{bmatrix} \boldsymbol{P} \\ \boldsymbol{Q} \end{bmatrix} u \tag{23}$$

where the "elements" of these vectors and matrices are now vectors and matrices themselves, this representation of **A** makes **A** look like a diagonal matrix. Separating the two vector equations, we get

$$\begin{cases} \dot{\boldsymbol{x}}_{1,2} = \boldsymbol{M}\boldsymbol{x}_{1,2} + \boldsymbol{P}u \\ \dot{\boldsymbol{x}}_{3,4} = \boldsymbol{N}\boldsymbol{x}_{3,4} + \boldsymbol{Q}u \end{cases} \tag{24}$$

And then writing each vector equation as a system,

$$\begin{cases} \dot{\boldsymbol{x}}_{1,2} = \begin{bmatrix} 0 & 1 \\ a & 0 \end{bmatrix} \boldsymbol{x}_{1,2} + \begin{bmatrix} 0 \\ -b_p \end{bmatrix} u, & \boldsymbol{C}_{1,2} = [\qquad] \\ \dot{\boldsymbol{x}}_{3,4} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{x}_{3,4} + \begin{bmatrix} 0 \\ b_r \end{bmatrix} u, & \boldsymbol{C}_{3,4} = [\qquad] \end{cases} \tag{25}$$

This is quite significant. It says that the dynamics of these subsystems are *decoupled*, or independent of each other. This is a direct result of **A** being in block diagonal form. (Note that this does *not* mean that you can control the dynamics of both arbitrarily – the same input u applies to both.) Another implication is that the eigenvalues of **A** are the union of the eigenvalues of **M** and the eigenvalues of **N**.

This last fact can be used to our advantage. We wish to make the observer response converge very quickly, and therefore want to set the eigenvalues of (**A-LC**) to be fast. Remember that we're now designing the internal dynamics of the observer; the input u is not applied here. Here is the breakdown of (**A-LC**) designed in 5-a and 5-b:

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \\ l_{31} & l_{32} \\ l_{41} & l_{42} \end{bmatrix}, \quad (\boldsymbol{A} - \boldsymbol{LC}) = \begin{bmatrix} -l_{11} & 1 & -l_{12} & 0 \\ a - l_{21} & 0 & -l_{22} & 0 \\ -l_{31} & 0 & -l_{32} & 1 \\ -l_{41} & 0 & -l_{42} & 0 \end{bmatrix} \tag{26}$$

So the first and third columns can be modified. Must the values that are not on the block diagonal be nonzero? For example, look at l_{12}. It determines the effect of $\hat{x}_3$ on $\dot{\hat{x}}_1$. But since the dynamics of the first two states are decoupled from the last two, that term is unnecessary. In other words, we can arbitrarily determine the dynamics of the observer

using only l_{11}, l_{21}, l_{32}, and l_{42}! In fact, the other terms only serve to make the observer more sensitive to model errors and noise.

If the dynamics are independent, then why did MATLAB give nonzero off-diagonal terms? Answer: the algorithms MATLAB uses do not check for independence.

5-c Split up the system as shown in Equation (25), and use MATLAB to find the 2 small **L** matrices. Then combine the two **L** matrices put their values in the l_{11}, l_{21}, l_{32}, and l_{42} positions. Compare this new **L** matrix with the **L** matrix found in Question 5-a. Now take the **L** matrix found in Question 5-a and zero out the terms off the block diagonal. Compare again. Is it important to split up and redesign, or do you find it sufficient just to zero out the terms off the block diagonal? *(Hint: Look at their effects on the observer eigenvalues.)* Repeat Question 5-b with the new **L** matrix. Any improvement?

Now we have a good design, and are ready to put it to work on the actual RWP.

5-d Copy and rename the Simulink file you developed in 4-c, 4-d to implement your 3-state feedback controller. Give it a name with "part5" in the name. Then use this file as a starter to implement your observer design on the RWP. (Make sure to set its COM port to the Reaction Wheel's board.) How does this controller compare to three-state feedback control? *In most systems if you have a direct measurement of a state like we do with the optical encoders, the direct velocity measurement is more robust than using the linearized approximate model in an Observer.*

6 Up and Down Stabilizing Control

We will now explore the topic of *switching control*. We will first discuss this topic, and then consider the example of the RWP, which we are quite familiar with. We can then design a switching controller for the RWP without much trouble.

Most systems that we control are nonlinear. We simply choose an operating condition and linearize about that condition. However, what if we want to control this system over a broader range of operating conditions? For example, airplanes are extremely nonlinear systems. Fighter jets are even more so, due to the enormous range of airspeeds and maneuverability requirements. A nonlinear controller would be extremely complicated and may even become unstable near the extremes, due to modeling errors. The approach used is to switch between many different linear controllers based on the states. Each controller uses a different model for the system, and applies a different type of controller, but all share the broad goal of keeping the jet in the sky[4]. The difficult aspect of switching control is handling the switching transients: When switching from one model and controller to a completely different one, how do you guarantee that the system won't go unstable?

To explore this further, let us consider again the RWP. Unlike a fighter jet, a pendulum has only two equilibrium points: up and down. We have designed a controller to balance up. If we make a switching control to balance down when the pendulum swings past the upward stabilizable region, will we be able to guarantee stability? Keeping in mind that the downward equilibrium is a stable equilibrium, it is not rocket science to determine that after the switching transients, the down controller will be able to stabilize the pendulum.

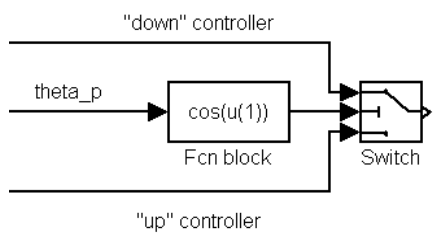


Figure 9: Implementing Switching Control

6-a Design a switching controller that will stabilize the RWP in either the up or down position based on the pendulum angle θ_p. To make your life easier, use three-state feedback controllers as in Section 4.3. *(**Hint:** use the "Switch" block controlled by a function of θ_p—see figure above.)* First simulate it, then implement it on the RWP.

[4] Controlling a nonlinear plant by switching between a family of linear controllers, each tuned for certain operating conditions, is called *gain scheduling*

If you don't see the effects of the controller in the down position, swing the pendulum freely and see how long it takes for the swinging to stop. There should be a significant decrease in that time with your new controller.

7 Swing-Up Control

After all that talk of avoiding nonlinear control by using switching control, let us now look at nonlinear control itself. It can be a very useful tool, especially when it is used along with other control algorithms. We can use switching control to switch between a nonlinear controller and a linear controller. This may sound complicated, but actually "switching control" is nothing more than an algorithm that switches between multiple controllers. We can use a nonlinear controller to get the system into the region that is stabilizable with linear control, and then switch over to the linear controller. That is the approach we will use to swing-up the pendulum and then stabilize it at the top position.

The concept of nonlinear control may sound daunting, but look at the problem in this way: how can we pump energy into this system properly, and how can we get the system to recognize that it is in the "region" of stabilizability? A clue lies inside that question: energy. We can measure kinetic and potential energy, we want a certain setting of kinetic and potential energy, and we can apply kinetic energy.

Let us look at this from a naïve point of view. Assume that we want to tap the pendulum really hard at the bottom, but just hard enough to get it to swing up and come to rest (briefly, of course) at the top. **(Do not try this! The RWP is fragile.)** You can imagine tapping it harder or softer based on how high up it swings. Is there a way to figure out just how hard you need to tap it? Yes! The energy at the bottom is purely kinetic, and you want the energy at the top to be purely potential. Therefore, you can compute how intensely you must tap.

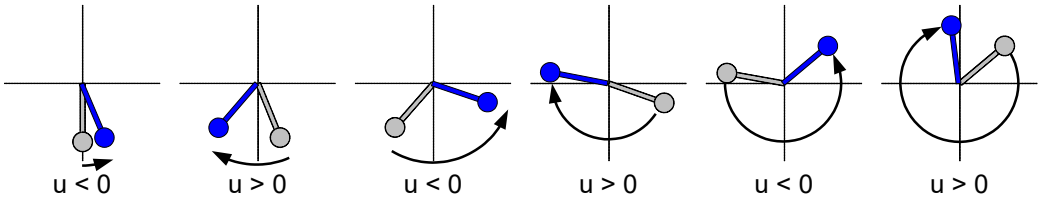


Figure 10: Swing-Up Control Energy Sequence

On the actual RWP, you have the added advantage that the motor is mounted on the pendulum, so you do not need to tap. Rather, you can apply a long-term force (see Figure 10). But there is a disadvantage to this: the force is limited by the maximum velocity of the rotor (as we have seen before). It turns out that the motor cannot provide the necessary energy input in a single swing. Therefore, the motor must dump some energy during one swing, then dump energy in the other direction during swingback, and so on until the RWP has the correct amount of energy. Figure 10 shows a plot of the intersection pendulum energy and the ideal energy when the pendulum is balanced in an inverted position

There is, of course, the added complication of friction. To account for this, we can simply dump in more energy than required without friction and hope it works. This process takes much trial and error.

This is only one of many methods of doing swing-up control. There are many implementation details involved in making a swing-up controller. Have fun!

Appendix A: Useful Physics Theory

Conversions (units of MKS)

- Length (m), Mass (kg), and Time (s) are basic units.
- Force is mass • acceleration, and has units of newtons ($N = kg \bullet m/s^2$).
- Energy is force applied over a certain distance, and has units of joules ($J = N \bullet m = kg \bullet m^2/s^2$).
- Power is an impulse of energy, with units of watts ($W = J/s = N \bullet m/s$).
- Inertia is the change in force required to make a unit change in acceleration. It has units of change in force per change in acceleration, or $N/(m/s^2)$.
- Moment of inertia is the analog of inertia for rotational objects. It is the change in torque required to make a unit change in angular acceleration. It has units of change in torque per change in angular acceleration, or $Nm/(rad/s^2) = kg \bullet m^2$.

  Note: (rad) is considered unitless

Energy Equations

Potential energy …

… for a mass $= mgh$

Kinetic energy …

… for a moment of inertia $= \frac{1}{2} J\omega^2$

Appendix B: Implementation Notes

This appendix contains many details that will be critical during simulation.

Simulink Notes

- *To start Simulink:* First open MATLAB. Then either type `simulink` in the command window, or click on the Simulink button (enlarged below) on the toolbar.

- *Setting up Simulink parameters:* When running a Simulink simulation, you need to set up a few parameters in order to keep conformity with Windows Target. From the **Simulation** menu, select **Configuration Parameters**, then in the **Solver Options** box, set Type to "Fixed-step" and "ode1: Euler".
- *Changing simulation Start/Stop time:* You can also change Start Time and Stop Time in the Simulation Parameters box.
- *Nonlinear RWP model:* There is a nonlinear RWP model available for simulation. To find it, type `pend_blks` in the MATLAB command window. You will need the Reaction Wheel Block Diagram Model (nonlinear model) and the Reaction Wheel Animation (animation block). Points to remember:
    - For the Reaction Wheel Block Diagram Model, "Tau1" corresponds to the pendulum arm. (That arm is not actuated, but that's where the disturbances are applied.)
    - "Tau2" corresponds to the rotor. (That's where the control effort should be applied.)
    - Don't forget to include a Saturation block (to saturate the control effort at ± 10) before "Tau2".
    - The nonlinear model outputs encoder states (ϕ_p, ϕ_r), but your controller uses delta states ($\delta\theta_p$ instead of θ_p).
    - You can change the initial conditions of the Reaction Wheel Block Diagram Model by double-clicking on the block. You should only have to change the first initial condition (pendulum position, where π signifies upwards).
    - To slow down animation speed, go to **Simulation** » **Configuration Parameters**, and decrease the "Max step size". (This will force Simulink to do more calculations, thus slowing down the simulation.)