Inertia Wheel Inverted Pendulum

Ashwin Krishna
Harvard University
Email: ashwin_krishna@college.harvard.edu

Nao Ouyang
Harvard University
Email: nouyang@g.harvard.edu

Written: May 22, 2019

***Abstract*—We explore the classic nonlinear controls problem, inverting a pendulum, using analyses learned in this class. Specifically, we look at using a flywheel to stabilize the pendulum. In simulation, we derive the equations of motion and apply LQR and region of attraction analyses for our system. We also build a hardware system from scratch. In hardware, we successfully implement downward stabilization, swingup (using a bang-bang four state controller), and inverted stabilization (using both a PD controller) and a bang-bang controller. Future work includes adding either current control or motor velocity estimate to allow for use of LQR control and not just PD control. A demo video can be found at https://youtu.be/bWbEt6hoUvY.**

I. Introduction

The inverted pendulum problem has been widely explored in robotics and control theory. In this project, we control a single-link pendulum using a torque-controlled inertial wheel. We tackle the problems of stabilizing at the bottom (0 deg) and swinging up to stabilize at the top (180 deg).

A. Related Work

The theory of the reaction-wheel inverted pendulum problem has been widely explored. For related work, we turned more to a few actual hardware implementations of similar problems that are documented online.

In 2010, a prior 6.832 student, Hunter McClelland, built a reaction wheel inverted pendulum. [1]. In 2013, Spanlang, Mayr, and Gattringer of the Institute of Robotics at Johannes Kepler University Linz (Austria) built a reaction-wheel inverted pendulum contained within a 18x18x18 cm cube, using the floor as the base of the pendulum. [2].

In 2012, Shane Colton built the Seg Stick at MITERS (the machine shope we built our project in). The Seg Stick is a self-balancing broomstick powered by two drill-driven wheels at the base of the stick. It has the functionality of the cart-pole system, but less constrained, and with the ability to move the system while balancing. In section IV, we further discuss the possibilities of adding a similar feature to our reaction-wheel inverted pendulum system. See [3]

In 2016, Ben Katz posted a blog about the process of building a Furuta Pendulum from scratch and using LQR controls. For Furuta-type inverted pendulums, instead of controlling the x coordinate of the base of the pendulum, the control is in the form of another rotational axis on which the pendulum is attached orthogonally. Thus, by rotating the system rapidly in the x, y plane, stability in the z plane can be achieved. [4]

A few further resources were documented online on a blog post [5].

II. Hardware Methods

To constrain our reaction wheel inverted pendulum, we constructed a wooden jig that holds a spinning shaft in place with two ball bearings. The shaft (functioning as the base of the pendulum) is fixed to the plastic lever at the end of which the flywheel and motor are mounted. An encoder is press-fit into the shaft at the base of the pendulum, which gives us our θ_1 value. To further minimize the escaping of the system's energy through means other than rotation, the system is clamped down. We used an Arduino Uno for electronics and code.

As with any hardware project, it took multiple iterations to arrive at a sufficiently-controllable prototype. The first iteration used a printer motor, and a solid aluminum flywheel with about 4 inches in diameter. After experimenting with trying to control the pendulum with this setup, it became apparent that the flywheel needed more inertia.

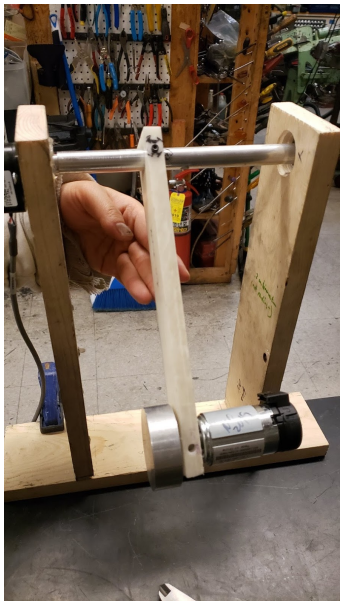

Fig. 1. Original motor (with encoder) and flywheel

After adding more weight and a slight increase in diameter to the flywheel, we experimented further. A higher swingup

was able to be commanded by this improvement, but we needed a motor with more torque, because the motor required an unnecessarily long ramp down in speed before switching directions.

Fig. 2. Three iterations of flywheel

Our third and final iteration featured a flywheel with a much larger diameter and weight distributed more on the perimeter, and a large upgrade to a drill motor with significantly more torque (available from harbor freight). These vast increases in both torque and inertia finally allowed the system to command the acceleration necessary for a full swingup.

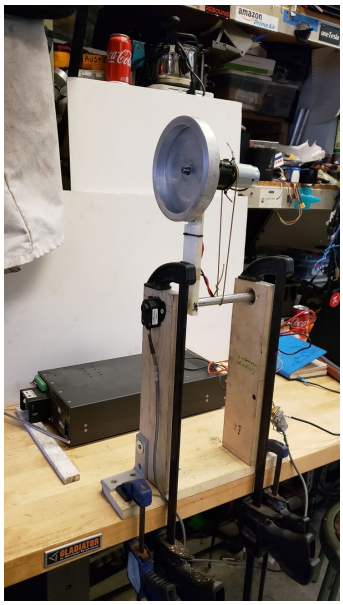

Fig. 3. Static upright pendulum

III. Hardware Methods

To constrain our reaction wheel inverted pendulum, we constructed a wooden jig that holds a spinning shaft in place with two ball bearings. The shaft (functioning as the base of the pendulum) is fixed to the plastic lever at the end of which the flywheel and motor are mounted. An encoder is press-fit into the shaft at the base of the pendulum, which gives us our θ_1 value. To further minimize the escaping of the system's energy through means other than rotation, the system is clamped down. We used an Arduino Uno for electronics and code.

As with any hardware project, it took multiple iterations to arrive at a sufficiently-controllable prototype. The first iteration used a printer motor, and a solid aluminum flywheel with about 4 inches in diameter. After experimenting with trying to control the pendulum with this setup, it became apparent that the flywheel needed more inertia.

After adding more weight and a slight increase in diameter to the flywheel, we experimented further. A higher swingup was able to be commanded by this improvement, but we needed a motor with more torque, because the motor required an unnecessarily long ramp down in speed before switching directions.

Our third and final iteration featured a flywheel with a much larger diameter and weight distributed more on the perimeter, and a large upgrade to a drill motor with significantly more torque (available from harbor freight). These vast increases in both torque and inertia finally allowed the system to command the acceleration necessary for a full swingup.

A. Theory to Reality

Later in the paper, we will explain the LQR theory. In reality, however, we relied on PD control. The reasons for this are as follows:

- Limited torque output. Even though our final motor brought a significant increase in torque, it was still far from the ideal (part of this was due to the flywheel's relatively large mass). For full controllability, we'd want a motor with enough torque to be able to handle rapid direction switching at high angular velocities.
- Lack of motor encoder. Though our first motor had an attached encoder, it did not have nearly enough torque, so we had to upgrade. Unfortunately, the newer drill motor did not have an attached encoder, so we lost the ability to track θ_2. $\dot{\theta}$, and $\ddot{\theta}_2$ accurately. Naturally, losing those state variables resulted in a decrease in controllability.
- **Lack of current control**. The software controller we wrote outputs torque, but we do not command torque directly (the motor interface only takes angular velocity as input).

To calculate the state variables, We have discrete digital values from the quadrature encoder on the shaft, which was a high quality one (1025 counts).

We furthermore use a naive method to estimate angular velocities from the encoder counts: measure time elapsed and the change in angle, divide, set as our thetadot.

B. Swingup with Bang-Bang Control

To get an initial working implementation to swing the pendulum up to 180 degrees, we used a simple bang-bang controller. The protocol is as follows. When $\theta_1 > 0$ and $\dot{\theta}_1$, run the motor at full speed clockwise to maximize the amplitude reached. When the apex of that period is reached (i.e. $\dot{\theta}_1 = 0$), run the motor at full speed in the opposite direction. Essentially, run the motor at full speed in whatever direction is consistent with the pendulum's direction of travel, w.r.t. the overall system. If the pendulum is moving clockwise, spin clockwise. If the pendulum is moving counterclockwise, spin counterclockwise. Using this bang-bang controller, we were able to achieve full swingup.

To make the swingup more efficient, we implemented a more complex controller, utilizing energy shaping (covered in section III F).

C. Inversion with PD Control

For downward convergence, we used a simple Proportional-Derivative (PD) controller with two k values: one for θ_1, and one for $\dot{\theta_1}$. Our feedback for motor output was formulated by:

$$\text{motor output} = -ceil(k(\theta_1 - \theta_{1d}) - k_d(\dot{\theta}_1 - \theta_{1dotd}))$$

where k and k_d represent the proportional and derivative constants, respectively, θ_{1d} represents the desired θ_1 value (0.0), and θ_{1dotd} the desired $\dot{\theta}_1$ value (0.0). To choose our k constants, we used a potentiometer to our arduino setup, for easy, on-the-go tuning.

D. Results

To quantify the efficacy of our controller, we timed multiple different tasks for both our first iteration (with the smaller flywheel and weaker motor), and our final iteration (with the larger flywheel and stronger motor). The tasks we timed were natural downward convergence (how long the pendulum takes to converge at 0 deg when started at 90 deg, without any control input), controlled downward convergence (how long the pendulum takes to converge at 0 deg when started at 90 deg, with control input), and controlled swingup (how long it takes for the pendulum to swing up to 180 deg). These findings are displayed in Table I. Video clips of these on the system can be found here:

TABLE I

	Natural Downward Convergence (starting at 90 deg)	Controlled Downward Convergence (starting at 90 deg)	Controlled Swingup (starting at 0 deg)
First Flywheel (smaller) and Weaker Motor	45 secs	11 secs	23 secs
Final Flywheel (larger) and Stronger Motor	46 secs	3 secs	3 sec

TABLE II
RESULTS ON THE DOWNWARD STABILIZATION AND SWINGUP TASKS (THE UPRIGHT STABILIZATION WAS NOT TIMED).

- Final Flywheel natural from 90: https://bit.ly/2JXV39B
- Final Flywheel controlled from 90: https://bit.ly/2WipsWJ
- First Flywheel (partial) Swingup: https://bit.ly/2X7tI8R
- Second Flywheel (full) Swingup: https://bit.ly/2WlHmrA
- Final Flywheel (full) Swingup: https://bit.ly/2VE6Qfn

IV. Simulation Analysis

A. Equations of Motion

To derive the equations of motion (EOM), we use the Lagrangian method. Let L equal to the kinetic energy plus the potential energy of the system.

$$L = KE - PE \tag{1}$$

By Lagrange's method,

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_i}) - \frac{\partial L}{\partial q_i} = \sum_{i=0}^{n} F_i \tag{2}$$

for $i = 1, 2, 3...n$ forces.

Thus, we need to write out the KE, the PE, the derivative of L with respective to each state q, the derivative of L with respect to the (time) derivative of each state q, and then the time derivative of that last term.

Let us first consider the unaltered case, from the problem set, where here we will derive the equations of motion by hand but otherwise simply explain the derivation in detail. Later, we will consider a system that more closely matches our real-life system. We will not be able to compare the model with reality, since we were unable to implement the full state measurement so LQR cannot apply. Instead, we show another example as applied to a modified system where the reaction wheel pendulum is put on an (unpowered) cart.

1) **Write the KE of the system**. We can decompose this into the translational and rotational components.

First, let us consider (abstractly) the translational KE of a point mass m rotating around the origin on a massless string of length l. θ is defined as angle from the downward vertical point, increasing counterclockwise (diagram not provided). The position of the point mass is $x = lcos\theta$ and $y = lsin\theta$. KE is $\frac{1}{2}m \cdot q^2$, where q is the position.

$$KE_x = 0.5m(l \cdot \frac{d}{dt}\sin\theta)^2 = \frac{1}{2}m(l\dot{\theta}\cos\theta)^2 \tag{3}$$

$$KE_y = 0.5m(l \cdot \frac{d}{dt}\cos\theta)^2 = \frac{1}{2}m(-l\dot{\theta}\sin\theta)^2 \tag{4}$$

$$KE = KE_x + KE_y = \frac{1}{2}ml^2\dot{\theta}^2(\cos^2\theta + \sin^2\theta) \tag{5}$$

$$= \frac{1}{2}ml^2\dot{\theta}^2 \tag{6}$$

$$\tag{7}$$

where on the last step we used the trig identity $\cos^2 + sin^2 = 1$.

Now applying this to the stick and flywheel components of our system, we calculate 1) the stick around the origin 2) the flywheel around the origin. Note that the KE of the stick acts at l_1, the center-of-mass of the stick, not l_2.

$$KE_{\text{translational}} = \frac{1}{2}m_1(l_1\dot{\theta}_1)^2 + \frac{1}{2}m_2(l_2\dot{\theta}_1)^2 \tag{8}$$

$$\tag{9}$$

Additionally we have the inertial component of KE since we have angular velocities here and our stick has mass and our previous point mass is instead a rotating flywheel. The general formula is $KE = \frac{1}{2}I_2\dot{\theta}^2$. Noting that angular velocities "add", and applying this to each component of our system; we calculate 1) inertial KE of the stick 2) inertial KE of the flywheel.

$$KE_{\text{inertial}} = \frac{1}{2}I_1\dot{\theta}_1^2 + \frac{1}{2}I_2(\dot{\theta}_1 + \dot{\theta}_2)^2 \tag{10}$$

The total KE of the system is the sum of the above.

2) **Write the PE of the system**. This is more straightforward. Gravitationally speaking, (and with a bit of geometry - note that our theta is defined from vertical and increasing counter-clockwise)

$$PE = m_1 g(-l_1\cos\theta_1) + m_2 g(-l_2\cos\theta_1) \tag{11}$$

3) Now we have the Lagrangian $L = KE - PE$ and must take the partial of the Lagrangian with respect to each state variable, in our case θ_1 and θ_2.

Using sympy (note: we left the sympy ordering intact, so the terms are a bit weird), we calculate

$$\frac{\partial L}{\partial q} = \begin{bmatrix} -gl_1m_1\sin(\theta_1) - gl_2m_2\sin(\theta_1) \\ 0 \end{bmatrix} \tag{12}$$

4) As an intermediate step, we calculate the

$$\frac{\partial L}{\partial \dot{q}} = \begin{bmatrix} I_1\dot{\theta}_1 + I_2(\dot{\theta}_1 + \dot{\theta}_2) + l_1^2m_1\dot{\theta}_1 + l_2^2m_2\dot{\theta}_1 \\ I_2(\dot{\theta}_1 + \dot{\theta}_2) \end{bmatrix} \tag{13}$$

5) Finally, we calculate the time derivative of the previous term

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} = \begin{bmatrix} I_2\ddot{\theta}_2 + \ddot{\theta}_1(I_1 + I_2 + m_1l_1^2 + m_2l_2^2) \\ I_2\ddot{\theta}_1 + I_2\ddot{\theta}_2 \end{bmatrix} \tag{14}$$

6) We set the equation equal, on the right hand side, to our input torque τ.

We may then directly ask sympy to solve for $\ddot{q}$

$$\ddot{\theta}_1 = -g\frac{(m_1l_1 + m_2l_2)\sin(\theta_1)}{I_1 + m_1l_1^2 + m_2l_2^2} \tag{15}$$

$$\ddot{\theta}_2 = g\frac{(m_1l_1 + m_2l_2)\sin(\theta_1)}{I_1 + m_1l_1^2 + m_2l_2^2} \tag{16}$$

More neatly, we can go directly from Eq. (14) and Eq. (12) to the "manipulator equations" as per the class textbook. Specifically, we put Eq. (14) on the left hand side, factoring out $\ddot{\theta}_1$ and $\ddot{\theta}_2$; then on the right hand side we put Eq. (12), factoring out $\dot{\theta}_1$ and $\dot{\theta}_2$ as well as adding in our input torque τ.

That is, we rewrite in form

$$\boldsymbol{M(q)\ddot{q} + C(q,\dot{q})\dot{q} = \tau_g(q) + Bu} \tag{17}$$

Doing so, we then get as given to us in the homework (yay it matches!)

$$\begin{bmatrix} m_1l_1^2 + m_2l_2^2 + I_1 + I_2 & I_2 \\ I_2 & I_2 \end{bmatrix}\begin{bmatrix}\ddot{\theta}_1 \\ \ddot{\theta}_2\end{bmatrix} + 0 = \begin{bmatrix} -(m_1l_1 + m_2l_2)g\sin\theta_1 \\ 0 \end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\tau \tag{18}$$

B. Linearization Around Fixed Point

We can further use sympy to linearize our fixed points. Focusing on the upright case, we can use the approximation

$$\sin\theta \approx \pi - \theta \text{ for } \theta \approx \pi \tag{19}$$

For the downward case, we can similarly use the approximation

$$\sin\theta \approx \theta \text{ for } \theta \approx 0 \tag{20}$$

After plugging in to sympy, we get

```
t1ddot= -g*(l1*m1 + l2*m2)*sin(t1)/(I1 + l1**2*m1 +
    l2**2*m2)
t2ddot= g*(l1*m1 + l2*m2)*sin(t1)/(I1 + l1**2*m1 +
    l2**2*m2)
```

.

Or written in Latex,

$$\ddot{\theta}_1 = -g\frac{(m_1l_1 + m_2l_2)\sin(\theta_1)}{I_1 + m_1l_1^2 + m_2l_2^2)} \tag{21}$$

$$\ddot{\theta}_2 = g\frac{(m_1l_1 + m_2l_2)\sin(\theta_1)}{I_1 + m_1l_1^2 + m_2l_2^2)} \tag{22}$$

This can be rewritten to be the same result as in the homework, where we substitute in the approximation around $\theta = \pi$

$$\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{(m_1l_1+m_2l_2)g}{(m_1l_1^2+m_2l_2^2+I_1)} & 0 & 0 & 0 \\ -\frac{(m_1l_1+m_2l_2)g}{(m_1l_1^2+m_2l_2^2+I_1)} & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix}\theta_1 - 180\\ \theta_2\\ \dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{-1}{(m_1l_1^2+m_2l_2^2+I_1)} \\ \frac{1}{I_2} + \frac{1}{(m_1l_1^2+m_2l_2^2+I_1)} \end{bmatrix}[\tau] \tag{23}$$

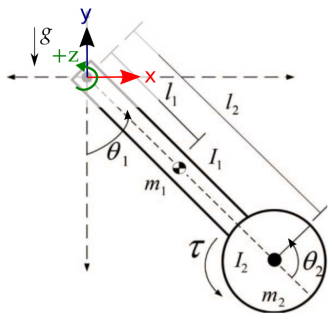

Fig. 4. Free body diagram

1) A and B: If we plug in the measurements from our physical system as in Table III, we get the following A and B matrices (rounded).

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 321 & 0 & 0 & 0 \\ -321 & 0 & 0 & 0 \end{bmatrix} \tag{24}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ -1568 \\ 2075 \end{bmatrix} \tag{25}$$

Note: We treat the stick mass as negligible compared the motor, which is modelled as a point mass at distance l_2; thus we set l_1 equal to l_2, and m_1 = m_{motor}.

C. Constants

See III

TABLE III
SYSTEM CONSTANTS

Property	Measurement
m_{stick} =	115 g
$m_{flywheel}$	546 g
m_{motor}	450 g
l_{stick}	21 cm
$r_{flywheel}$	8.5 cm

D. Applying LQR

Now that we have A and B, the matrices for our linearized dynamics of form $f(x) = Ax + Bu$, we supply our Q and R cost functions and apply lqr. We care a lot about the θ_1, some about the $\dot{\theta}_1$, a bit about $\dot{\theta}_2$, and not at all about θ_2. We also put a cost on the input using **R**(here we closely follow the assignment, since it turns out we will not be able to apply LQR to our physical system).

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{26}$$

$$R = \begin{bmatrix} 0.1 \end{bmatrix} \tag{27}$$

Using the LQR function built into Drake, we get (rounded)

$$K = \begin{bmatrix} -35 & 0 & -5 & -1 \end{bmatrix} \tag{28}$$

$$S = \begin{bmatrix} 12 & 0 & 1 & 0. \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0. & 0. \\ 0. & 0 & 3 & 0. \end{bmatrix} \tag{29}$$

where K is our control matrix, operating on each of the four states ($q = \theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2$); and S is the solution of the Ricatti equation.

E. Region of Attraction via Lyapunov

We will briefly cover the region of attraction (RoA) analysis, which is covered in the problem set already. For an LQR controller, which uses a linearization of underlying nonlinear dynamics, this analysis tells us the region for which the linearization is valid (where our LQR control can be used).

Specifically, we will use Lyapunov analysis. Lyapunov analysis is a relaxed optimization guarantee – instead of guaranteeing an controller optimal for all states will be found, we instead guarantee a controller will be able to accomplish a given state.

With the LQR controller, which operates as a constant matrix times the state error (from the desired fixed point)

$$u = -K(x_{\text{measured}} - x_{\text{fixed point}}) \tag{30}$$

We denote the error as $\bar{x} = x - x_{fp}$. For our Lyapunov function, we can use the cost-to-go of the LQR solution

$$V_{\text{cost-to-go}} = \bar{x}^T S \bar{x} \tag{31}$$

where S is as return to us by the Drake LQR solver. This S is the solution to the Ricatti equation (a randomly fancy name for a first order quadratic differential equation); in steady-state, this becomes an algebriac Ricatti equation.

Note that in performing this analysis we picked a single known reasonble Lyapunov function. Other functions are possible, which would give slightly different regions of attraction for the same LQR controller.

The value of this function, for a given state, can then be used to bound where our linear controls will work. In our controller, we simply check the value of V for the state we are in and compare it to this bound. If we are within the region of attraction, we use the LQR controller. Otherwise, we use the swing-up controller described in the following section.

F. Energy-based Controller for Swingup

The idea behind the energy-based swingup controller is straightforward: we add torque in the direction that the magnitude of θ_1 is increasing in. We care to increase θ_1. However, we cannot directly control θ_1, but instead apply torque to control θ_2. (In jargon, this is called "non-collocated input").

For the simple pendulum, we observed

$$\dot{E} = u\dot{\theta} \tag{32}$$

For our system, we rederive $\dot{E}$ accounting for the fact that our input is now non-collocated.

We actually desire $\dot{E}$ to be zero, since our E will be at steady-state, thus our energy error is directly $\dot{E}$. We then derive what u must be to drive this to zero.

In the end, in the actual sytem, a bang-bang controller was sufficient. Additionally, as the derivation is already covered in the homework, we will not repeat it here.

G. Controllability

This is again covered in the homework and will not be derived here.

The existence of multiple examples online would show that this system is generally theoretically controllable, even given torque limits.

In our case, we can also more directly consider a quick-and-dirty calcuation: what is maximum torque produced by gravity, compared to the maximum torque our motor can generate? Additionally, in reality the flywheel will also saturate at some max speed of the motor, past which back EMF will limit the speed of the flywheel. This strongly impacts the difficulty of controlling our system.

This analyses is omitted, as the results didn't quite match reality, and likely would need to be tailored further for the non-idealities of our physical system.

H. Is it Underactuated?

When talking to friends about this, one of the first questions asked was invariable "is that really an underactuated system?" especially since we do not care about θ_2.

By the definition in the textbook, A control system described by equation 1 is underactuated in state (q,q˙) at time t if it is not able to command an arbitrary instantaneous acceleration in q. With the particular caveat that the same system could be technically be fully or underactuated at different moments in time.

V. LQR for "System on Wheels"

For a detour (in order to demonstrate understanding of the problem set material) we imagine sticking the whole thing on wheels and redo the same analysis. (For sanity we run the calculations through sympy instead of by hand).

A. Equations of Motion

1) **Write the KE of the system**. We can decompose this into the translational and rotational components. First, let's write the x and y components of each part. The cart is located at x = x and y = 0.

$$\text{position\_cart} = [x, 0] \tag{33}$$
$$\text{position\_stick} = [x + l_1 \sin(\theta_1), l_1 \cos(\theta_1)] \tag{34}$$
$$\text{position\_wheel} = [x + l_2 \sin(\theta_1), l_2 \cos(\theta_1)] \tag{35}$$

Asking sympy to take derivatives since it's easy to drop terms by hand,

$$\text{velocity\_cart} = [\dot{x}, 0] \tag{36}$$
$$\text{velocity\_stick} = [l_1\dot{\theta}_1 \cos(\theta_1) + \dot{x}, -l_1\dot{\theta}_1 \sin(\theta_1)] \tag{37}$$
$$\text{velocity\_wheel} = [l_2\dot{\theta}_1 \cos(\theta_1) + \dot{x}, -l_2\dot{\theta}_1 \sin(\theta_1)] \tag{38}$$

Then we get

$$KE_{\text{translational}} = \frac{1}{2}M\text{v\_cart}^2 + \frac{1}{2}m_1\text{v\_stick}^2 + \frac{1}{2}m_2\text{v\_wheel}^2 \tag{39}$$

And as before we get the inertial kinetic energy term

$$KE_{\text{inertial}} = \frac{1}{2}I_1\dot{\theta}_1^2 + \frac{1}{2}I_2(\dot{\theta}_1 + \dot{\theta}_2)^2 \tag{40}$$

3) Now we have the Lagrangian $L = KE - PE$ and must take the partial of the Lagrangian with respect to each state variable, in our case x, θ_1 and θ_2.

$$\frac{\partial L}{\partial q} = \begin{bmatrix} 0 \\ -(gm_1l_1 + gm_2l_2 + m_1l_1\dot{\theta}_1\dot{x} + m_2l_1\dot{\theta}_1\dot{x}\sin(\theta_1)) \\ 0 \end{bmatrix}$$

4) As an intermediate step, we calculate the partial of L with respect to $\dot{q}$, $\frac{\partial L}{\partial \dot{q}}$.

Here we copy from sympy the three terms

```
([[M*xdot + m1*(2*l1*t1dot*cos(t1) + 2*xdot)/2 + m2
    *(2*l2*t1dot*cos(t1) + 2*xdot)/2,

1.0*I1*t1dot + 0.5*I2*(2*t1dot + 2*t2dot) + l1**2*m1
    *t1dot*sin(t1)**2 + l1*m1*(l1*t1dot*cos(t1) +
    xdot)*cos(t1) + l2**2*m2*t1dot*sin(t1)**2 + l2*
    m2*(l2*t1dot*cos(t1) + xdot)*cos(t1),

0.5*I2*(2*t1dot + 2*t2dot)]]))
```

.

5) Finally, we calculate the time derivative of the previous term. Here we again note directly from sympy $\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i}$ =

```
t1ddot*(l1*m1 + l2*m2)*cos(t1) - t1dot**2*(l1*m1 +
    l2*m2)*sin(t1) + xddot*(M + m1 + m2)],

1.0*I1*t1ddot + 1.0*I2*t1ddot + 1.0*I2*t2ddot + 1.0*
    l1**2*m1*t1ddot - 1.0*l1*m1*t1dot*xdot*sin(t1) +
     1.0*l1*m1*xddot*cos(t1) + 1.0*l2**2*m2*t1ddot -
     1.0*l2*m2*t1dot*xdot*sin(t1) + 1.0*l2*m2*xddot*
    cos(t1),

1.0*I2*(t1ddot + t2ddot)
```

.

6) We set the equation equal, on the right hand side, to our input torque τ. We may then directly ask sympy to solve for $\ddot{q}$

```
xddot= -(l1*m1 + l2*m2)*(0.5*g*(l1*m1 + l2*m2)*sin
    (2.0*t1) + t1dot**2*(I1 + l1**2*m1 + l2**2*m2)*
    sin(t1) + tau*cos(t1))/(I2*(M + m1 + m2) + (l1*
    m1 + l2*m2)**2*cos(t1)**2 - (M + m1 + m2)*(I1 +
    I2 + l1**2*m1 + l2**2*m2))

t1ddot= (g*(l1*m1 + l2*m2)*(M + m1 + m2)*sin(t1) +
    0.5*t1dot**2*(l1*m1 + l2*m2)**2*sin(2.0*t1) +
    tau*(M + m1 + m2))/(I2*(M + m1 + m2) + (l1*m1 +
    l2*m2)**2*cos(t1)**2 - (M + m1 + m2)*(I1 + I2 +
    l1**2*m1 + l2**2*m2))

t2ddot= (-I2*g*(l1*m1 + l2*m2)*(M + m1 + m2)*sin(t1)
     - 0.5*I2*t1dot**2*(l1*m1 + l2*m2)**2*sin(2.0*t1
    ) + tau*((l1*m1 + l2*m2)**2*cos(t1)**2 - (M + m1
     + m2)*(I1 + I2 + l1**2*m1 + l2**2*m2)))/(I2*(I2
    *(M + m1 + m2) + (l1*m1 + l2*m2)**2*cos(t1)**2 -
     (M + m1 + m2)*(I1 + I2 + l1**2*m1 + l2**2*m2)))
```

.

B. Linearization

With the above values, we can linearize as before. Our A matrix should now be a 6x6 matrix instead of a 4x4 matrix. Solving for $\dot{x} = Ax + Bu$

$$\dot{x} = \begin{bmatrix} \dot{x} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{x} \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}, Ax = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ (\text{see } \ddot{x} \text{ above}) & & & & & \\ (\text{see } \ddot{\theta}_1 \text{ above}) & & & & & \\ (\text{see } \ddot{\theta}_2 \text{ above}) & & & & & \end{bmatrix} \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \\ \dot{x} \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{41}$$

C. LQR

Now we can directly plug A, B, Q, and S into LQR to get out a linear controller. This is ommitted for time reasons. The Lyapunov analysis should follow easily as well from the problem set. The energy shaping analysis requires more modification, but is omitted for time reasons.

VI. Conclusion and Future Work

Building this system from scratch, including fabricating, wiring, and coding, came with numerous lessons. We learned, for example, the importance of being qualitative in the setup and design of the system (i.e. calculating torque needed to control the system, given the inertial properties of the flywheel, etc.), first hand. We also experienced the nuances of different control methods, and when each can and cannot be used (i.e. not having all of the measured states required for LQR, and thus switching to a PD controller with energy-shaping). We faced the many realities that come with hardware projects, but were ultimately able to control the reaction-wheel inverted pendulum with just a PD controller, achieving relative stability at both the origin and apex.

Future work should include verifying the robustness of the system and quantifying the region of attraction of this physical system, which will allow for more precise control. Eventually, we'd like to transition to a 3D printed version of this system and stick it on wheels!

ACKNOWLEDGMENTS

The authors would like to thank many people, including: on the theory side, Elizabeth Mitten and Shane Colton for theory help, the TAs Wei Gao and Yunzhu Li, the instructor Prof. Russ Tedrake. Motor control discussion, Bayley Wang (and Shane). PD tuning method, Ben Katz.

REFERENCES

[1] H. McClelland. Reaction wheel control demo — youtube. [Online]. Available: https://www.youtube.com/watch?v=j9RDpmamRRQ

[2] J. Mayr, F. Spanlang, and H. Gattringer, "Mechatronic design of a self-balancing three-dimensional inertia wheel pendulum," *Mechatronics*, vol. 30, pp. 1–10, 2015.

[3] Seg... stick — instructables. [Online]. Available: https://www.instructables.com/id/Segstick/

[4] Desktop inverted pendulum part 2: Control. [Online]. Available: http://build-its-inprogress.blogspot.com/2016/08/desktop-inverted-pendulum-part-2-control.html

[5] Final project proposal research (6.832) — orangenarwhals. [Online]. Available: http://orangenarwhals.com/hexblog/2019/04/10/Final-Project-Proposal/#Note

APPENDIX

A. Code for Equations of Motion (flywheel pendulum)

```
import sympy
from sympy import sin, cos, simplify, Derivative,
    diff
from sympy import symbols as syms
from sympy.matrices import Matrix
from sympy.utilities.lambdify import lambdastr

import time


t1, t2, t1dot, t2dot, t1ddot, t2ddot, tau = syms('t1
     t2 t1dot t2dot t1ddot t2ddot tau')
m1, l1, I1, m2, l2, I2, g = syms('m1 l1 I1 m2 l2 I2
    g')

p = Matrix([m1, l1, I1, m2, l2, I2, g])       #
    parameter vector
q = Matrix([t1, t2])

qdot = Matrix([t1dot, t2dot]) # time derivative of q
qddot = Matrix([t1ddot, t2ddot]) # time derivative
    of qdot
K_translat = Matrix([0.5 * m1 * (l1 * t1dot)**2 + \
    0.5 * m2 * (l2 * t1dot)**2])
K_inertial = Matrix([0.5 * I1 * t1dot**2 + \
                     0.5 * I2 * (t1dot + t2dot)**2])

P = Matrix([-1 * m1 * g * (l1 * cos(t1)) + -1 * m2 *
     g * (l2 * cos(t1))])
L =  K_translat + K_inertial - P

# To calculate time derivatives of a function f(q),
    we use:
# df(q)/dt = df(q)/dq * dq/dt = df(q)/dq * qdot


partial_L_by_partial_q = L.jacobian(Matrix([q])).T
partial_L_by_partial_qdot = L.jacobian(Matrix([qdot
    ]))
d_inner_by_dt = partial_L_by_partial_qdot.jacobian(
    Matrix([q])) * qdot + \
    partial_L_by_partial_qdot.jacobian(Matrix([qdot
    ])) * qddot

lagrange_eq = partial_L_by_partial_q - d_inner_by_dt

r = sympy.solvers.solve(simplify(lagrange_eq),
    Matrix([qddot]))

t1ddot = simplify(r[t1ddot])
t2ddot = simplify(r[t2ddot])

print('t1ddot= {}\n'.format(t1ddot));
print('t2ddot= {}\n'.format(t2ddot));

# --- Simply substitute, for theta = pi2,  sin pi =
    1, sin theta ~= (pi - theta )
```

.

B. Code for Swingup and Upright Controller

The constants were determined by hand (in another (separate) program, two potentiometers were wired up and used to tune the gains).

Gain tuning proceeded as per Ben Katz's suggestion:

- Set K_p to zero. Increase K_d until chattering unreasonable (where the tiniest disturbance will cause motor to go forward and reverse rapidly). This shows the maximum damping the system can produce.
- Next increase the K_p until too much overshoot occurs.
- Profit.

```
// Modify for encoder-less (no motor encoder) new
    prototype
// 16 May 2019

#include <Rotary.h>
#include <MegaMotoHB.h>
#include <math.h>

//https://cdn.usdigital.com/assets/datasheets/
    H5_datasheet.pdf?k=636931248608523021

// --------Lever Encoder--------
int val;
volatile int encoder1Pos = 0;
volatile int encoder2Pos = 0;
Rotary rMotor = Rotary(2, 3); // motor (theta2)
Rotary rStick = Rotary(A5, A4);  // stick (theta1)
int n = LOW;
/*const byte CPin = 0;  // analog input channel*/
/*int CRaw;      // raw A/D value*/
/*float CVal;    // adjusted Amps value*/

// --------Motor--------
int EnablePin = 8;
int duty;
int PWMPin = 11;  // Timer2
int PWMPin2 = 10;
MegaMotoHB motor(11, 10, 8);
int motor_output = 0; // command to motor

// --------P-Controller--------
double thetadot_deadband = 0.2;
double theta_deadband = 5;

int sample_time = 5; // 15 msec

double theta1 = 0.0; // get_from_encoder()
double theta2 = 0.0; // get_from_encoder()
double prev_theta1 = 0.0; // get_from_encoder()
double prev_theta2 = 0.0; // get_from_encoder()

double theta1dot = 0.0; // get_from_encoder()
double delta_theta1 = 0.0; // get_from_encoder()
double theta2dot = 0.0; // get_from_encoder()
double delta_theta2 = 0.0; // get_from_encoder()

double theta1_desired = 0.0;
double theta1dot_desired = 0.0;

double err_theta = 0.0;
double err_thetadot = 0.0;

int delta_motor = 0;
int prev_motor = 0;

bool theta_CW;
bool motor_CW;

unsigned long now = 0;
unsigned long time_elapsed;
unsigned long prev_time = 0;

double state[4];
```

```
double k = 4; // theta constant
double kdot = -80; // thetadot

void setup() {
    Serial.begin(230400);
    /*Serial.begin(9600); // for use with plotter
    tool */

    rMotor.begin();
    rStick.begin();
    PCICR |= (1 << PCIE2);
    PCMSK2 |= (1 << PCINT18) | (1 << PCINT19);
    PCICR |= (1 << PCIE1);
    PCMSK1 |= (1 << PCINT13) | (1 << PCINT12);
    sei();

    /*motorOn();*/
    motor.Enable();
    motor.SetStepDelay(1);
    /*setPwmFrequency(PWMPin, 8);  // change Timer2
    divisor to 8 gives 3.9kHz PWM freq*/
}

void loop(){
    // -------- update time --------
    now = millis();
    time_elapsed = (now - prev_time);
    if (time_elapsed >= sample_time)
    {

    // -------- update theta --------
    // issue: prev_theta is the same as theta

    theta2 = getCurrentTheta2();
    theta1 = getCurrentTheta1();
    delta_theta2 = theta2 - prev_theta2;
    delta_theta1 = theta1 - prev_theta1;
    theta1dot = delta_theta1 / time_elapsed;
    theta2dot = delta_theta2 / time_elapsed;
    prev_theta2 = theta2;
    prev_theta1 = theta1;
    state[0] = theta1;
    state[1] = theta2;
    state[2] = theta1dot;
    state[3] = theta2dot;

    // -------- determine motor input --------
    /*Serial.println(motor_speed);*/

    err_theta = theta1 - theta1_desired;
    err_thetadot = theta1dot - theta1dot_desired;
    motor_output = - ceil( k * (theta1 -
    theta1_desired) - kdot * (theta1dot -
    theta1dot_desired));
    delta_motor = motor_output - prev_motor;
    prev_motor = motor_output;
    //    Serial.print(motor_output);
    /*aprintf("\ntheta1 %f, t2 %f, t1dot %f, t2dot %
    f, out %d, deltath %f, cw ", */
    /*theta1, theta2, theta1dot, theta2dot,
    motor_output, delta_theta1);*/
    /*aprintf("\n %d %f %d %f %f ", delta_motor,
    theta1, motor_output, err_theta, err_thetadot);
    */
    /*aprintf("\n %f %f %f %f ", theta1, err_theta,
    theta1dot, err_thetadot);*/
    /*aprintf("\n %f %f ", theta1dot, err_thetadot);
    */
    aprintf("\n t1 %f errtheta %f, errdot %f, motor
    out %d, t1dot %f", theta1, err_theta,
    err_thetadot, motor_output, theta1dot);
    /*Serial.println(theta1);*/
    /*Serial.print(delta_motor);*/
    /*// -------- write appropriate motor input
    --------*/
    motor_output  = abs(constrain(motor_output,
    -150, 150));
    motor_output = 200;
    if (theta1 > 8) {
        if (theta1dot > 0.01) {
            motor.Rev(motor_output);
            /*motorCCW(abs(motor_output));*/
        }
        else if (theta1dot < 0.01) {
            motor.Fwd(-motor_output);
            /*motorCW(abs(motor_output));*/
        }
    }
    else if (theta1 < -8) {
        if (theta1dot > 0.01) {
            motor.Rev(motor_output);
            /*motorCCW(abs(motor_output));*/
        }
        else if (theta1dot < -0.01) {
            motor.Fwd(-motor_output);
            /*motorCW(abs(motor_output));*/
        }
        else {
            // do nothing
        }
    }
    else {
        // theta angle small; do nothing or use
        //motor.Stop();
        motorWrite(1);
    }

    // SANITY CHECK
    /*
    motor.Rev(200);
    delay(500);
    motor.Fwd(200);
    delay(500);
    motor.Stop();
    delay(500);
    */

    prev_time = now;
    }
}

// --------- Helper Functions -------

// -------- Motor Funcs --------


// Implement bang bang control on theta2 dot dot
// -- This is PID loop to control actual motor speed
     to desired speed
void motorWrite(int someValue) {

    if (someValue > 0) {
        if (theta2dot > 0 ) motor.Rev(someValue); //
    motor.Stop();
        else motor.Rev(someValue);
    }
    else if (someValue < 0) { // < 0
        someValue = abs(someValue);
        if (theta2dot > 0 ) motor.Fwd(someValue);
        else motor.Fwd(someValue);
    }
    else {
        //do nothing
    }
}
```

```
// -------- Angle Conversion --------
double getCurrentTheta1() { // calibration for stick
     encoder = 1250
    double val = (double(encoder1Pos) / 1250) * 360;
    val = fmod(val, 360);
    if (val > 180) {
        val = val - 360;
    }
    return val;
}

double getCurrentTheta2() {  // 500 ticks / rev, for
     motor encoder
    double val = (double(encoder2Pos) / 500 ) * 360;
    val = fmod(val, 360);
    if (val > 180) {
        val = val - 360;
    }
    return val;
}


// ---- Set interrupt to read encoder ----


// -------- Read encoders --------

ISR(PCINT2_vect) { // motor, on D2 and D3
    unsigned char result = rMotor.process();
    if (result == DIR_NONE) {
    }

    else if (result == DIR_CW) {
        encoder2Pos--;
    }
    else if (result == DIR_CCW) {
        encoder2Pos++;
    }
}

ISR(PCINT1_vect) { // stick, on A5 and A4
    unsigned char result = rStick.process();
    if (result == DIR_NONE) {
    }

    else if (result == DIR_CW) {
        encoder1Pos--;
        //        Serial.println(getCurrentTheta());
    }
    else if (result == DIR_CCW) {
        encoder1Pos++;
    }
}



//---- print help ---------
int aprintf(char *str, ...) {
  int i, j, count = 0;

  va_list argv;
  va_start(argv, str);
  for(i = 0, j = 0; str[i] != '\0'; i++) {
    if (str[i] == '%') {
      count++;

      Serial.write(reinterpret_cast<const uint8_t*>(
    str+j), i-j);

      switch (str[++i]) {
        case 'd': Serial.print(va_arg(argv, int));
    // int
          break;
        case 'l': Serial.print(va_arg(argv, long));
    // long
          break;
        case 'f': Serial.print(va_arg(argv, double))
    ; // float
          break;
        case 'c': Serial.print((char) va_arg(argv,
    int)); // char
          break;
        case 's': Serial.print(va_arg(argv, char *))
    ; // string
          break;
        case '%': Serial.print("%");
          break;
        default:;
      };

      j = i+1;
    }
  };
  va_end(argv);

  if(i > j) {
    Serial.write(reinterpret_cast<const uint8_t*>(
    str+j), i-j);
  }

  return count;
}


void setPwmFrequency(int pin, int divisor) {
  byte mode = 0;
  if(pin == 5 || pin == 6 || pin == 9 || pin == 10)
    {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 64: mode = 0x03; break;
      case 256: mode = 0x04; break;
      case 1024: mode = 0x05; break;
      default: return;
    }
    if(pin == 5 || pin == 6) {
      TCCR0B = TCCR0B & 0b11111000 | mode;
    } else {
      TCCR1B = TCCR1B & 0b11111000 | mode;
    }
  } else if(pin == 3 || pin == 11) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 32: mode = 0x03; break;
      case 64: mode = 0x04; break;
      case 128: mode = 0x05; break;
      case 256: mode = 0x06; break;
      case 1024: mode = 0x07; break;
      default: return;
    }
    TCCR2B = TCCR2B & 0b11111000 | mode;
  }
}
```

.