Inertia Wheel Inverted Pendulum

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

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

***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. We build a physical system from scratch. In simulation, we derive the equations of motion and apply LQR and region of attraction analyses for our system. We additionally perform controllability analysis to the case where the full state may not be measurable, as happened in our physical system. In hardware, we successfully implement downward stabilization and swingup controls using both a PD controller an a bang-bang controller. We discuss the design decisions, design iterations, and present work toward implementing a controller for the inverted state.**

I. Introduction

Pendulum is inherently unstable around upright fixed point.

mds

May 22, 2019

II. Hardware Methods

First, motor with encoder, and small flywheel. Insert picture of the three iterations.

(Final motor was a drill motor, available for $25 at harbor freight)

A. Theory to Reality

Below, we will explain the LQR theory. However, in reality we relied on PD control. The reasons for this are:

- Limited torque output
- Lack of motor encoder
- Lack of current control – our software controller outputs torque, but we do not command torque directly, and needed to write an inner PID loop (I can fill this in)

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.

(Non constant time frame)

B. Swingup with Bang-Bang Control

C. Inversion with PD Control

We used a simple control with two k values: one for theta, and one for thetadot. This is equivalent to a PD controller.

D. Results

Insert a table here! Our excellent results.

And link to the videos.

III. 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)

$$PE = m_1g(-l_1\cos\theta_1) + m_2g(-l_2\cos\theta_1) \tag{11}$$

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

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

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 $\theta = \pi$

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

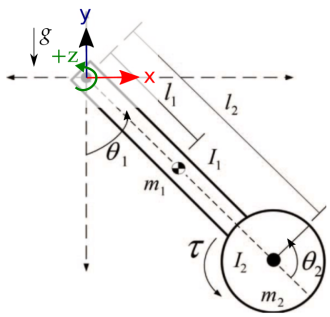

Fig. 1. Free body diagram

TABLE I
SYSTEM CONSTANTS

Property	Measurement
$m_{stick} = m_1$	115 g
$m_{flywheel}$	546 g
m_{motor}	450 g
$l_{stick} = l_2$	21 cm
$r_{flywheel}$	8.5 cm

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

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 312 & 0 & 0 & 0 \\ -312 & 0 & 0 & 0 \end{bmatrix} \tag{24}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ -1395 \\ 1901 \end{bmatrix} \tag{25}$$

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

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.

IV. 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, although for sanity we run the calculations through sympy instead of by hand. [1]

1) Equations of Motion:

2) Linearization:

V. Discussion

A. Discussion

1) estimate motor velocity: if we wanted to, we could do it from current

It does seem from online that it's possible to stabilize with this estimate (link to youtube video)

2) rebuild with less torque limited (maybe try this analysis again? I didn't get it to work above – skip if running out of time)

B. Mechanical Lessons Learned

Pressfits are great! They're used for… everything. Decisiveness is good – we weren't certain about going for the better motor, which instinctually thought it'd be needed but went for it and glad we did. Glad we gave some consideration to clearance – bigger motor and flwheel barely fit (scrapes the staples). Maybe would go back to 3d printed version Current control

VI. Future Work and Conclusion

We got it to invert! In the future, LQR ... :'(But for simple systems like this inverted pendulum (and not humanoids) PD is plenty, don't need LQR really. And then... stick it on wheels! – however this version will likely be 3d printed...

Acknowledgment

The authors would like to thank Elizabeth Mitten for theory help Shane Colton, Mason Massie, eom Shane Colton, PD control and current control Bayley Wang, current control and late-night company Ben Katz, PD tuning

References

[1] J. Mellado. (2018) js-aruco: Javascript library for augmented reality applications. [Online]. Available: https://github.com/jcmellado/js-aruco

Appendix

VII. 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 )
```

.

VIII. Bill of Materials

A list of the components of the sensor is found in II.

TABLE II
LIST OF COMPONENTS AND APPROXIMATE COSTS

Part	Details	Cost
Camera	Mini Camera module, AmazonSIN: B07CHVYTGD	\$20
LED	Golden DRAGON Plus White, 6000K. 124 lumens	\$2
4 springs	Assorted small springs set	\$5
3D printed pieces	PLA filament	\$5
Heat-set Threaded Inserts	Package of 50 from McMaster-Carr (use 2)	\$1
Misc. Bolts	Hex socket head	\$1
Epoxy	5 minute	\$5
3.3 V source (Arduino)	*Optional*	*\$15*