Double Inverted Pendulum: Stable Control via Feedback Linearization with Python Simulation and Analysis

Bardia Mojra

***Abstract*—Stable control of *double inverted pendulum* (DIP) is a classic problem in control systems and is considered to be a good case for studying and control nonlinear systems. DIP is highly nonlinear and sensitive to the initial conditions which makes it particularly difficult to control. This article focuses on the derivation of *equations of motion* and *feedback linearization* method which, were used to develop a symbolic representation of the system. We show that the system characteristic matrix is positive definite and is invertible as required by *Lyapunov's direct method*. Using MATLAB Control Toolbox, we solve the continuous *Lyapunov control function* to obtain the *linear quadratic regulator* (LQR) solution which is considered to be unique, stable, and optimal. To enhance this case study, an end-to-end software suite is developed in Python to simulate, animate, and analyze the system. Moreover, a series of experiments are designed and performed to 1) study the effects of variations in model parameters (i.e. masses, pendulum lengths, and friction) on system stability, 2) investigate system dynamics and stability at different initial conditions, given tuned and untuned control parameters. 3) And to investigate and present some evidence to confirm chaotic nature of the system. The source code developed for this project will be available at https://github.com/BardiaMojra/dip.**

I. Introduction

A. Related Work

This is related

B. Objectives

The objective of this article is to present a concise, intuitive, and transferable the solution to a classical optimal control problem. Our objective is to model the plant using Lagrange's equations of motion, derive state dynamics, obtain a generalized cost function, and to optimally control the double pendulum by reaching a stable configuration in the upright position while given minim-energy control input to the system.
Also — testing
Also —

C. Article Structure

II. Estimation Formulation

There are many benefits to using the *state space* framework, notably that is well generalized and conveniently implemented on modern computers.

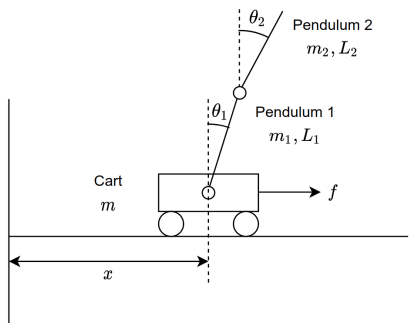


Fig. 1. Physical System

A. System Setup

Consider a resting cart on a flat rigid surface along the x-axis, on top of which are stacked two inverted pendulums, figure 01. The pendulums resemble a robotic arm but rather without a motor or external torque input. At the end of each pendulum is assumed to be a point-mass, m_1, m_2. Each pendulum moves freely and independently and is connected by lengths l_1 and l_2 from their joints, q_1 and q_2 to their point-masses, m_1, m_2. The mass of the cart is considered point-mass as well and is denoted by m_c with its position q_c along the x-axis. θ_1 and θ_2 denote angular deviations from upright position for q_1 and q_2, respectively.

B. State Definition

Suppose we have a continuous-time, nonlinear system presented in state space form.

$$\dot{x} = f(x, u), \tag{1}$$

$$y = h(x). \tag{2}$$

whats f - prediction model whats h - obs model omit obs noise, it requires EKF??

Our goal is to stabilize the described highly nonlinear system in the upright configuration

C. Noise and Friction

D. Global Coordinate Frame

We need to form a homogenous configuration space to present state variables q, θ_1, θ_2 by forming a global coordinate frame with $x(t=0)$ as the origin.

$$q_c = \begin{bmatrix} q \\ 0 \end{bmatrix}, \quad q_1 = \begin{bmatrix} q + l_1 \sin\theta_1 \\ l_1 \cos\theta_1 \end{bmatrix}, \quad q_2 = \begin{bmatrix} q + l_1 \sin\theta_1 + l_2 \sin\theta_2 \\ l_1 \cos\theta_1 + l_2 \cos\theta_2 \end{bmatrix}, \tag{3}$$

We denote position of point-masses for m_c, m_1, and m_2 by $q_c, q_1, q_2 \in R^2$ on the x-y plane. We seek to define our system in terms of these state variables. Thus, we define state variable vector $q(t)$ as

$$q = [q_c, q_1, q_2]^T \tag{4}$$

In a later section, we formally rewrite state variable definitions independent of time t via a linearization process. For simpler representation, we denote state variable vector as $q := q(t)$.

E. Other Considerations

Noise and Friction modeling, process noise versus observation noise.

III. Model Derivation

A. Equations of Motion

As it is laid out in *Lagrangian mechanics*; first we must obtain the Lagrangian of the physical model, $\mathcal{L} = K - P$. K and P represent the *kinetic* and *potential* energies of the system. K is the total kinetic energy from all three masses and in later sections, we will discuss transforming *Lagrangian mechanics* to *Hamiltonian mechanics* by replacing velocities with *momenta*. Here, we define K as

$$K = \frac{1}{2}mv^2 = \frac{1}{2}m_c\|\dot{q_c}\|^2 + m_1\|\dot{q_1}\|^2 + m_2\|\dot{q_2}\|^2. \tag{5}$$

P denotes the overall potential energy of the system; but since the cart rolls flat along the x-axis, it is inherently incapable of storing energy so we discard it. P is obtained by

$$P = g\left[m_1 l_1 \cos\theta_1 + m_2(l_1\cos\theta_1 + l_2 cos\theta_2)\right] \tag{6}$$

Where $g \in R$ denotes gravitational acceleration. Next, we envoke the *Euler-Lagrange* equation to write the following ODE system whose solution is the position and velocity configuration vector of the cart and masses on the x-y plane, given the final time, previous state, system dynamics, control input and disturbances.

$$y(t) = f(t, q, \dot{q}, u, \omega) \tag{7}$$

The Lagrange Equations of Motions are defined as

$$Q = \frac{\partial \mathcal{L}}{\partial f_i} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial f_i}\right). \tag{8}$$

Where Q represents the vector of sum of external forces acting on the plan.

$$\mathcal{L} = \mathcal{L}(t, q, \dot{q}) \tag{9}$$

$$\begin{cases} u + \omega_1 - d_1\dot{q} &= \frac{d}{dt}\{\frac{\partial\mathcal{L}}{\partial\dot{q}}\} - \{\frac{\partial\mathcal{L}}{\partial q}\} \\ \omega_2 - d_2\dot{\theta_1} &= \frac{d}{dt}\{\frac{\partial\mathcal{L}}{\partial\dot{\theta_1}}\} - \{\frac{\partial\mathcal{L}}{\partial\theta_1}\} \\ \omega_3 - d_3\dot{\theta_2} &= \frac{d}{dt}\{\frac{\partial\mathcal{L}}{\partial\dot{\theta_2}}\} - \{\frac{\partial\mathcal{L}}{\partial\theta_2}\} \end{cases} \tag{10}$$

So far our derivation has made our expressions more complicated but these steps are necessary for capturing the full dynamical characteristics of the system. In the following section, we derive the continuous-time nonlinear model of the described system. Moreover in *Optimal Control* section, we briefly explain how one could derive *the general optimal control solution* for a *positive definite and semidefinite energy system* with known characteristics. Moreover, we intuitively explain and reference why we could consider the solution space to such an ODE as *smooth manifold* that maps current and future state through recursion and its energy state decays towards a local minima.

B. Continues-Time, Nonlinear Model

We used MATLAB Symbolic Math Toolbox to derive the resultant ODE. Obtaining symbolic or implicit expressions of our ODE will allow us to analytically derive to the optimal solution.

$$\begin{aligned} u + \omega_1 - d_1\dot{q} &= (m_c + m_1 + m_2)\ddot{q} + l_1(m_1+m_2)\ddot{\theta_1}\cos\theta_1 \\ &\quad + m_2 l_2\ddot{\theta_2}\cos\theta_2 - l_1(m_1+m_2)\dot{\theta_1}^2\sin\theta_1 \\ &\quad - m_2 l_2\dot{\theta_2}^2\sin\theta_2 \\ \omega_2 - d_2\dot{\theta_1} &= l_1(m_1+m_2)\ddot{(q)}\cos\theta_1 + l_1^2(m_1+m_2)\ddot{\theta_1} \\ &\quad + l_1 l_2 m_2\ddot{\theta_2}\cos(\theta_1-\theta_2) + l_1 l_2 m_2\dot{\theta_2}^2\sin(\theta_1-\theta_2) \\ &\quad - g(m_1+m_2)l_1\sin\theta_1 \\ \omega_3 - d_3\dot{\theta_2} &= l_2 m_2\ddot{q}\cos\theta_2 + l_1 l_2 m_2\ddot{\theta_1}\cos(\theta_1-\theta_2) + l_2^2 m_2\ddot{\theta_2} \\ &\quad - l_1 l_2 m_2\dot{\theta_1}^2\sin(\theta_1-\theta_2) - l_2 m_2 g\sin\theta_2 \end{aligned} \tag{11}$$

C. Optimal Control

D. Linearization

First, we rewrite the equations of motion into matrix form to batch acceleration, velocity, and position terms into separate matrices. A second order representation of the physical system is sufficient to provide an accurate enough estimate, per *Taylor Series Linearization* method. This is an important step as identifying system dynamics lies at the heart of *Optimal Control*.

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Hu \tag{12}$$

where matrix functions $D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q)$ represent systems dynamics with respect to time, current state and previous state. Time variable t is omitted to simplify notation. The system energy state (left hand side of above EOM) is presented in relation to external forces Hu acting on the system. Vector $u \in R^1$ represents external input forces vector and matrix H represents its relationship with state variable functions. Matrix functions $D(q)$, $C(q,\dot{q})$, $G(q)$ and vector H are define as

$$D(q) \Rrightarrow (\tag{13}$$

We need to rewrite the EOM in the following matrix form to obtain the final ODE. Per *Lyapunov Control Function theory* (LCF) for autonomous dynamical systems and *LaSalle's*, we can assume there exists a *Optimal Control Trajectory* or *Hamiltonian Flow* for a valid initial condition if the LCF is *positive definite* or *positive semi-definite*.

Moreover, the energy function describing the system must be *symmetric* and *positive definite* in order to form *stable* control trajectory.

$$M(q)\dot{q} = A(q)q + Bu \tag{14}$$

Where *positive definite* or *positive semi-definite* constraints are evaluated by computing the determinant of M.

$$det(M) > 0;\ \forall q \tag{15}$$

Such that, the final ODE can be written as

$$\dot{q} = M(q)^{-1}(A(q)q + Bu) \tag{16}$$

To make remove the time index and make the ODE fully *isomorphic*, we define a new variable state vector as

$$x := \begin{bmatrix} y \\ \dot{y} \end{bmatrix} \tag{17}$$

L(x, u) = x T Qx + uT Ru
we

E. Discrete, Linear Model

$$\begin{aligned} x &= Ax + Bu \\ y &= \dot{x} \end{aligned} \tag{18}$$

where A and B are defined as

$$A = \begin{bmatrix} 0 & I \\ D^{-1}\frac{\partial G}{\partial q} & 0 \end{bmatrix} \tag{19}$$

$$B = \begin{bmatrix} 0 \\ D^{-1}H \end{bmatrix} \tag{20}$$

F. Linear Quadratic Regulator Solution

So far, we have modeled, linearized, and discretized our system using state space general form. In the process, we systematically reduced the solution space complexity by replacing time t and velocity dimension as linear products of current and previous states. This allows us to analytically derive a general form optimal solution. But before we reach that step, we need to define a general *Cost Function* J that only depends on absolute minimal parameters, which we will later optimize to arrive to a *minimal surface*. Quadratic performance index J is defined as [see page 24 of Optimal control - now we can optimize similar to static cases]

$$J(x, u) = \frac{1}{2}x^T Qx + \frac{1}{2}u^T Ru \tag{21}$$

whose solution is the minimal cost or minimum-energy input required for the system to reach local minima, for any given initial state. Next, we minimize J which will yield the state feedback weight vector K and is obtained by recursively solving the dynamic *Riccati* equation. It is important to note that calculation of Quadratic losses are carried out by performing dot product of state vector with its transpose and energy magnitude.

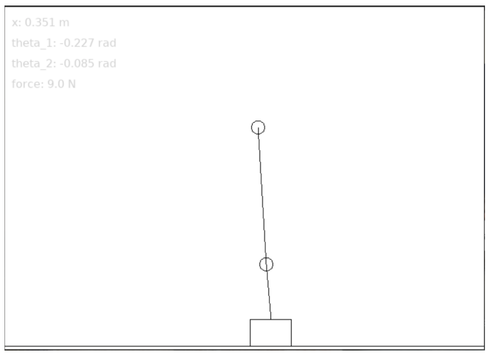
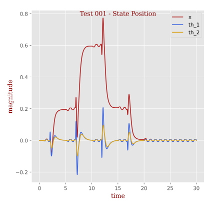
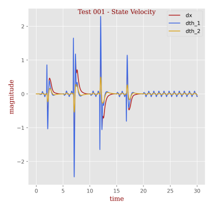


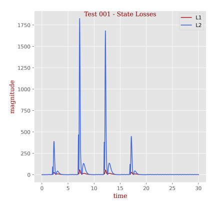
Fig. 2. Animation of Double Inverted Pendulum



(a) $q(t)$



(b) $\dot{q}(t)$



(c) Loss

Fig. 3. Test 001

IV. Simulation and Animation

For this project, both MATLAB and Python were used at different stages of the development. Besides symbolic math derivation, MATLAB Control Toolbox was used to check Python ODE simulation output and to obtain the LQR solution. MATLAB is the gold-standard of scientific numerical analysis and Python is versaltile and rich with high level feature. Various Python-based control system libraries were explored and considered but were abandoned after observing variations in the *control vector* parameters. The LQR solution is expected to be optimal for *Lyapunov energy systems* and variations in the solution raises doubts about numerical integrity of the computation process. For that reason, derivations and control solutions used in this work are consistent with MATLAB.

Various Python libraries are used to simulate, animate and analyze model dynamics. Pymunk and Pygame libraries are used create a 2D physics-based simulation environment. Pyglet library (based on OpenGL) is used to create the animation, figure 2. Source code for the animation and graphics are base on examples and documentation provided by the authors [1].

A. Experiment I

In the first experiment, we perform a set of tests with the goal of studying system stability with respect to changes in model parameters.

V. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

TABLE I
EXPERIMENT I

Test ID	l_1	m_1	l_2	m_2
001	0.6	0.2	0.6	0.2
002	0.6	0.3	0.6	0.2
003	0.6	0.4	0.6	0.2
004	0.6	0.4	0.6	0.3
005	0.6	0.4	0.6	0.4
006	0.6	0.3	0.6	0.4
007	0.6	0.2	0.6	0.4
008	0.6	0.2	0.5	0.2
009	0.6	0.2	0.7	0.2
010	0.6	0.2	0.8	0.2
011	0.5	0.2	0.6	0.2
012	0.4	0.2	0.6	0.2
013	0.3	0.2	0.6	0.2
014	0.7	0.2	0.6	0.2

REFERENCES

[1] V. Blomqvist, "Pymunk: A easy-to-use pythonic rigid body 2d physics library (version 6.2.1)," 2007.