Georgia Tech

Course and Learning Material

Inverted Cart-Pendulum

The inverted cart-pendulum is a classic example of an underactuated, naturally unstable, nonlinear system. It is somewhat simple as the project is single input. However, it does involve unmatched nonlinearities which does involve more advanced concepts.

Equations of Motion

There are many instances or versions of the equations of motion for this classical problem, depending on the nature of the pendulum. Here, we try to include the dynamics of the more general configuration. The equations of motion are usually derived from the energy of the cart-pendulum pairing,

$$\begin{aligned}\mathcal{E} &= \frac{1}{2}m_c\dot{x}^2 + \frac{1}{2}m_p\left(\dot{x} + l\cos(\theta)\dot{\theta}\right)^2 + \frac{1}{2}m_p l^2\sin^2(\theta)\dot{\theta}^2 + \frac{1}{2}J_p\dot{\theta}^2 + m_p g l(1-\cos(\theta)) \\ &= \frac{1}{2}(m_c+m_p)\dot{x}^2 + m_p l\cos(\theta)\dot{x}\dot{\theta} + \frac{1}{2}(J_p + m_p l^2)\dot{\theta}^2 - m_p g l(1-\cos(\theta))\end{aligned}$$

or the Lagrangian

$$\mathcal{L} = \frac{1}{2}(m_c+m_p)\dot{x}^2 + m_p l\cos(\theta)\dot{x}\dot{\theta} + \frac{1}{2}(J_p + m_p l^2)\dot{\theta}^2 + m_p g l(1-\cos(\theta)),$$

where x describes the cart position and θ describes the pendulum angle. Working out the Euler-Lagrange equations and adding in dissipation (which is more of a Lagrange-d'Alembert derivation if I am not mistaken) leads to

$$\begin{bmatrix} m_c+m_p & m_p l\cos(\theta) \\ m_p l\cos(\theta) & J_p + m_p l^2 \end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{\theta}\end{bmatrix} - \begin{bmatrix} m_p l\dot{\theta}^2\sin(\theta) \\ m_p g l\sin(\theta)\end{bmatrix} = \begin{bmatrix} u - \delta_\theta\dot{x} \\ -\delta_\theta\dot{\theta}\end{bmatrix}$$

If the pendulum is a ball at the end of a thin stick, then the inertia is negligible and J_p can be ignored. If the pendulum is a dense rod with no ball at the end, then the inertia J_p should not be ignored. The coefficients δ_x and δ_θ are coefficients of friction.

Given the prevalence of $m_p l$ almost everywhere, it is useful to factor the product out

$$\begin{bmatrix} \frac{m_c+m_p}{m_p l} & \cos(\theta) \\ \cos(\theta) & \frac{J_p}{m_p l} + l \end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{\theta}\end{bmatrix} - \begin{bmatrix} \dot{\theta}^2\sin(\theta) \\ g\sin(\theta)\end{bmatrix} = \frac{1}{m_p l}\begin{bmatrix} u - \delta_x\dot{x} \\ -\delta_\theta\dot{\theta}\end{bmatrix}$$

Simplifying things with a variable substitution,

$$\begin{bmatrix} M & \cos(\theta) \\ \cos(\theta) & L \end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix} \dot{\theta}^2\sin(\theta) - D_x\dot{x} \\ g\sin(\theta) - D_\theta\dot{\theta}\end{bmatrix} + \begin{bmatrix} bu \\ 0\end{bmatrix}$$

where

$$M = \frac{m_c+m_p}{m_p l},\quad L = \frac{J_p}{m_p l} + l,\quad b = \frac{1}{m_p l},\quad D_x = \frac{\delta_x}{m_p l},\quad D_\theta = \frac{\delta_\theta}{m_p l}.$$

The inverse of the matrix on the left is

$$\begin{bmatrix} M & \cos(\theta) \\ \cos(\theta) & L \end{bmatrix}^{-1} = \frac{1}{ML - \cos^2(\theta)}\begin{bmatrix} L & -\cos(\theta) \\ -\cos(\theta) & M\end{bmatrix}$$

Applying to the derived equations thus far,

$$\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \frac{1}{ML - \cos^2\theta} \begin{bmatrix} L\sin(\theta)\dot{\theta}^2 - g\cos(\theta)\sin(\theta) - LD_x\dot{x} + D_\theta\cos(\theta)\dot{\theta} + Lbu \\ -\cos(\theta)\sin(\theta)\dot{\theta}^2 + Mg\sin(\theta) + D_x\cos(\theta)\dot{x} - MD_\theta\dot{\theta} - b\cos(\theta)u \end{bmatrix}$$

$$= \frac{1}{ML - \cos^2\theta} \begin{bmatrix} L\sin(\theta)\dot{\theta}^2 - g\cos(\theta)\sin(\theta) - LD_x\dot{x} + D\cos(\theta)\dot{\theta} \\ -\cos(\theta)\sin(\theta)\dot{\theta}^2 + Mg\sin(\theta) + D_x\cos(\theta)\dot{x} - MD\dot{\theta} \end{bmatrix} + \frac{1}{ML - \cos^2\theta} \begin{bmatrix} bLu \\ -b\cos(\theta)u \end{bmatrix}$$

Next, expose the structure of the control input as something like ΛBu

$$\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \frac{1}{ML - \cos^2\theta} \begin{bmatrix} L\sin(\theta)\dot{\theta}^2 - g\cos(\theta)\sin(\theta) - LD_x\dot{x} + D_\theta\cos(\theta)\dot{\theta} \\ -\cos(\theta)\sin(\theta)\dot{\theta}^2 + Mg\sin(\theta) + D_x\cos(\theta)\dot{x} - MD_\theta\dot{\theta} \end{bmatrix} + \frac{b}{ML - \cos^2\theta} \begin{bmatrix} L & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -\cos(\theta) \end{bmatrix} u$$

The Λ equivalent term is on the wrong side, but that's OK. It will have to do.

Parameters and Limits

Parameter	Value
m_c	0.5 kg
m_p	0.1 kg
J_p	0.006 kg m^2
l	0.3 m
δ_θ	0.05 N m / rad s
δ_x	0.01 N m / s
g	9.8 m / s^2

My units might be off for the friction, but the values are fine.

Activities

Step 1: Linear Controller

Linearized System: Pendulum Stabilization

You should notice that the control input affects both the cart and the pendulum, meaning that control of both simultaneously is not possible. One state coordinate will have to be uncontrolled or poorly controlled while the other will be nicely controlled. Which to do depends on the task at hand. For example, stabilizing the pendulum to the up position only will lead the cart to slowly drift away from the origin. Putting a small amount of control effort to return to the origin is possible, but it should not dominate the pendulum control. Stabilizing the location of the cart will lead to the pendulum falling.

Linearize the equations of motion of the system and provide the linearized equations. Start by specifying the control task to be stabilization of the pendulum and show that it can be stabilized from non-zero angular deflection values. Add in a less aggressive cart stabilization term to bring the cart back to the origin. Show that you can achieve both objectives for reasonable pendulum angle and cart position offsets.

Implement on the linear and nonlinear dynamics. Compare the outcomes.

Linearized System: Cart Trajectory Tracking

Derive a tracking controller for the system so that it can track a cart trajectory while maintaining the pendulum upright. You should be able to identify a reasonable trajectory to track. Implement on the linear and nonlinear dynamics. Compare the outcomes.

Adaptive System

Perturb the underlying parameters by 10-20% and compare the outcomes for the perturbed system to those from the unperturbed system. Quantify the additional control effort for the stabilization and trajectory tracking cases. Quantify the increase in tracking error for the trajectory tracking case. It is assumed you know how to compute the L_2 signal norms to quantify the control effort and state error.

Implement an adaptive system and compare the outcomes. Run the adaptive system twice. Once with the incorrect values, then once with the final adaptive coefficients from the first run as that initial adaptive coefficients for the second run. Compare the quantitative values of the incorrect run, the first adaptive run, and the second adaptive run (where compare means to provide the outcomes and to discuss them).

Step 2: Neuro-Adaptive Controller

The inverted cart-pendulum system is a bit tricky to work with from a nonlinear control perspective because the single control acts through the two states to "stabilize." Furthermore, as a nonlinear system, it is characterized by a relatively large region of attraction for the linear controller. I am not sure how large it is once the parameters vary, but for the correct parameters it can be pretty large (covering a $\pm 60^\circ$ deviation from up). The added value of a nonlinear controller might not be worth the effort depending on the domain of interest for this system.

Nevertheless, let's try to perform some kind of nonlinear cancellation. If the cancellation is to be explicitly designed, then it can only really occur for one of the two main states and the other will get a lot of extra terms in it. Instead of trying to cancel one or another, try to create a general cancellation scheme that attempts to learn the best term that will enhance tracking a reference model. Design a neuro-adaptive adaptive control structure using Gaussian radial basis functions to cancel out any weirdness in the system in an effort to result in idealized performance.

Run the system with increasing angular errors until the system loses stability. Note down a few initial conditions that will lead to instability. Run your neuro-adaptive controller on a few stable instances to learn the α parameters (include a few that go near the point of instability). With these α parameters, run the unstable initial conditions and see what happens.

Turn in plots of the "training" runs where α was learnt and make sure to include all reasonable signals plus any comparison signals that should be plotted together with them (as needed). Turn in plots of the additional unstable runs. Did the neuro-adaptive system remain unstable or was it stable?

How well it works is a function of how many basis elements you have, their bandwidth, etc. Make sure that they cover the domain of operation (including the extended area where you'd like to have stability). Note that the nonlinearities are independent of the cart position x, so you should not include it in the function approximator. That keeps the network size smaller by not having to learn any x-dependence. The input dimension is 3 instead of 4.

Tip: Neuro-Adaptive Updates

When implementing new things, it is important to start from a position of strength. Do not try to do it all in one step given that there are many smaller steps that you most likely have never done. Break the system down into smaller pieces. From the first step, you should already have functioning adaptive controllers on only the linear system, plus running on the true nonlinear dynamics (but with a linear reference model). Naturally, this means you also have the non-adaptive versions working, as those should have been an initial testing step prior to incorporating adaptation.

That means the new part is the neuro-adaptive component. Rather than try to toss the entire thing in, it is better to implement neuro-adaptive estimation on a simpler system as the only adaptive component. In fact, it is best to first try out the neuro-adaptive controller on a first-order scalar system. The one below is a great option:

$$\dot{x} = f(x) + u(x;\alpha) = f(x) + kx - \alpha^T \Phi(x)$$

where $k < 0$. Pick some nonlinear function $f(x)$ and then see if you can estimate it using the neuro-adaptive component. Establish the domain of approximation. Figure out what a decent grid spacing for the Gaussian RBF centers is and how many would fit in the domain based on the grid spacing. Pick a bandwidth consistent with the grid spacing. Then create the adaptive system and see if it can learn the correct α. The way to test it would be to compare the function $\alpha^T \Phi(x)$ against $f(x)$. If you got it right, then they should agree up to some small approximation error. Even nicer would be to also plot the difference and the $\log$ of the difference, $\alpha^T \Phi(x) - f(x)$ and $\log(\alpha^T \Phi(x) - f(x))$, to see that it is learning the function. Usually learning is along the trajectories followed, so evaluate ths function along the trajectory of the previously integrated simulation(s). Run for different time durations. You should see the approximation get better as it runs longer, up to some limit. Heck, for completeness it would be good to try out a few functions $f(x)$ and see how well it works for them.

This is just one tip. Overall, if you are not doing it, you need to learn how to break down a problem into digestable bits. The Step 1 and Step 2 breakdown does that, but you can and should go ever further yourself when resolving this project.

Tip: Implementing the Function Approximator

One difficult thing for many newbies to Matlab is writing compact and efficient code. Efficient usually means avoiding `for` loops as much as possible. Chances are your neural network will have on the order of 100 to 10,000 *neurons*, so you really want to leverage Matlab's built-in functions for iterating over a matrices. You want to do the same for the center creation. Some Matlab function shout outs are:

- `meshgrid`: Use this to create set of centers with pre-defined spacing. Matlab will return the evaluation points as multiple matrices, one for each coordinate (up to three possible coordinates). They need to be combined into one big vector by properly vectorizing and joining.
- `bsxfun`: Use this to create the difference between the matrix of centers (given column-wise) and the point in the domain that one would like to evaluate the function at. Once this difference exists, it is pretty easy to generate the exponential function evaluations necessary in Gaussian radial

basis functions. One tip is to pre-multiply the evaluation point by the (bandwidth) Σ matrix as it too is a vector.
- `sum` and `.^`: Use Matlab's built in matrix-wide operations to compute the exponent terms of the Gaussian function evaluations.

If you do it all properly, the addition of a neuro-adaptive controller shouldn't add but about 4-5 lines of code to the setup prior to invoking the `ode45` part, about 3-4 lines of code to the actual differential equation function code, plus a 3-6 line function that computes $\Phi(x)$. For sure you should be writing a separate function for $\Phi(\cdot)$. In all, there shouldn't be more than a 20 line code difference between the traditional MRAC and the neuro-adaptive MRAC systems. I write this to help you try to get compact code that will run more efficiently. If you don't do this, then the numerical integration can take a *very, very* long time relative to doing so. In my version, the neuro-adaptive controller runs maybe 30% slower than the MRAC one. Simulation out to about 150 seconds takes about 15 seconds or less (I am just going by feel as I didn't keep track).

Step 3: What

TBD **Not Given in ECE6554 2019 Version. Do only Steps 1 and 2.**

References

No references given. This project is self-contained given the lecture notes and the tips.

Back – Main

---

ece6554/project_invcartpend.txt · Last modified: 2019/04/24 15:55 by pvela