Exercise Solution: Pole Balancing

Advanced Control lecture
at Ecole Centrale Paris

Anne Auger and Dimo Brockhoff

firstname.lastname@inria.fr

Jan 10, 2014

Abstract

Balancing a pole on a moving cart is a standard benchmark problem of control engineering. A related control problem has to be solved within the Segway personal transporter. In this exercise, we implement the most basic pole balancing problem (one single pole mounted on a cart that is only able to move in one dimension where we abstract from friction).

Solutions are written in blue.

1 Simulating the Pole Balancing Problem

Choose your favorite language (recommended: MATLAB/SciLab[1]) and implement the pole balancing problem from the lecture. To this end, use the simple Euler method to approximate the ODEs for angle and position accel-

[1]Useful commands to look at: `sign`, `fprintf`, `disp`

erations, given by

$$\begin{aligned}
\ddot{\theta}_t &= \frac{g\sin\theta_t + \cos\theta_t\left[\frac{-F_t - m_p l\dot{\theta}_t^2\sin\theta_t}{m_c+m_p}\right]}{l\left[\frac{4}{3} - \frac{m_p\cos^2\theta_t}{m_c+m_p}\right]} \\
\ddot{x}_t &= \frac{F_t + m_p l\left[\dot{\theta}_t^2\sin\theta_t - \ddot{\theta}_t\cos\theta_t\right]}{m_c+m_p},
\end{aligned}$$

the linear controller mentioned in the lecture, and the variables and parameters as described in Table 1.

Recommended Procedure:

a) Start with functions/methods for computing $\ddot{\theta}$ and $\ddot{x}$.
The MATLAB code can be found in `polebalancing/getAngleAccelaration.m` and `polebalancing/getPosAccelaration.m`.

b) Continue with a function for the linear controller.
The response F_t of the controller is based on the four constants k_1, k_2, k_3, and k_4 (choose them from $[0,1]$) as well as on $F_m = 100N$:

$$F_t = F_m\text{sgn}(k_1 x_t + k_2\dot{x}_t + k_3\theta_t + k_4\dot{\theta}_t).$$

The function $\text{sgn}(x)$ is the signum function, giving 1 if $x > 1$, 0 if $x = 0$, and -1 otherwise.
The MATLAB code can be found in `polebalancing/linearController.m`

c) Combine all parts to a single script/function that simulates the system for 120s and a fixed parameter setting using the Euler method:

$$\begin{aligned}
x_{t+1} &= x_t + \tau\dot{x}_t \\
\dot{x}_{t+1} &= x_t + \tau\ddot{x}_t \\
\theta_{t+1} &= \theta_t + \tau\dot{\theta}_t \\
\dot{\theta}_{t+1} &= \dot{\theta}_t + \tau\ddot{\theta}_t
\end{aligned}$$

The MATLAB code can be found in `polebalancing/simulation.m`.

Table 1: System parameters and variable names for the pole balancing problem.

Symbol	Name	Description
θ	Pole Angle	measured (in radians) relatively to the upright position, initial value in $[-0.1, +0.1]$
$\dot{\theta}$	Pole Velocity	angular velocity of the pole in rad/s
$\ddot{\theta}$	Pole Accelaration	acceleration of the pole in rad/s^2
x	Card Position	measured relatively to the middle of the track (in m), initial value in $[-1, +1]$
$\dot{x}$	Card Velocity	velocity of the cart (in m/s)
$\ddot{x}$	Card Acceleration	acceleration of the cart (in m/s)
g	Gravitational Acceleration	acceleration due to gravity (g =9.81 m/s^2)
m_c	Cart Mass	1.0 kg
m_p	Pole Mass	0.1 kg
l	Pole Length	distance from pivot to the pole's center of mass (l=0.5m)
t	Time	measured in s
F_t	Force	force applied to the cart at time t (in N, always $F_t \neq 0$ for a bang-bang controller)
h	Track Limit	$\pm$2.4m from track center
r	Pole Failure Angle	$\pm 12^\circ$ from vertical ($12^\circ \approx 0.209$rad)
τ	Time Step	discrete integration time step for the simulation ($\tau = 0.02$s)
F_m	Controller Constant	constant of linear controller (set to $F_m =$ 100N)
k_1, k_2, k_3, k_4	Controller Constants	further constants of controller (in $[0, 1]$, to be optimized)

d) Output the number of iterations until your simulation results in an unstable pole (θ not in $[-12°, +12°]$) to see whether a given parameter setting is producing a good controller. The MATLAB code can be found in `polebalancingRandom/`. The script to start the simulations is `polebalancingRandom/simulate_random.m` which calls the function `polebalancingRandom/simulate.m`. This function itself contains the previous script `polebalancing/simulation.m` written as a function.

e) Test your controller by choosing 1000 different (randomly chosen[2]) settings for k_1, k_2, k_3, and k_4.

2 Questions

a) Is it easy to find parameter values that produce a stable controller?
In a sense, yes, it is easy to find stable controllers, because already a random search finds in about 10% of the cases a stable controller. This is typically not the case for more complicated optimization problems as we will see later on in the class.

b) Are different starting conditions x and θ of the system simulation equally difficult for the linear controller?
No, the starting conditions have a large impact on whether it is easy to find good parameter values for the linear controller randomly. The closer the angle θ and the position x are to zero, the easier it is to find parameter settings that result in a stable linear controller.

c) What is the influence of the simulation frequency τ?
The simulation accuracy is a crucial parameter. The larger τ is chosen, the less probable it is to find good parameter values for the linear controller. The smaller τ, the better. It is recommended to choose the simulation frequency τ as low as possible (i.e. with the simulation accuracy $1/\tau$ as high as possible) such that you can still run the simulations in reasonable time (matter of seconds here).

[2]Look out for the command `rand(n,1)`.

3 Non-Mandatory Questions

If you have more time, are you able to answer the following two questions? With the provided MATLAB code, the answers are easily obtainable from additional simulations with different parameter settings.

a) Are the found good controllers also robust to changes in the cart and the pole mass?
As long as cart and pole mass are not drastically changed, the simulation results in terms of the stability of the linear controller stay nearly the same.

b) How would you find robust parameter values of a controller for (more or less) arbitrary starting conditions and masses?
One possibility would be to do the simulation for each parameter setting of the linear controller several times with different (random) starting conditions. The average/median/sum/minimum number of stable iterations over all simulations could be used as a quality of each parameter setting. What would be a better choice for this quality criterion, the sum or the minimum?

c) Is it easier to build a bang-bang controller or one that allows arbitrary (continuous) forces to be applied in each step?
The provided MATLAB code also allows to easily change the bang-bang controller (with a force of either $-F_m$ or $+F_m$) into a controller where the force can be any continuous number. In principle, this change reduces the probability that a randomly chosen parameter setting results in a stable linear controller.

d) Is this also true if the starting position of the pole is exactly in the middle?
Interestingly, this behavior of the continuous force controller changes when the angle and position are chosen optimally in the beginning (i.e. $\theta_0 = 0$ and $x_0 = 0$). But the explanation is trivial: the continuous force controller chooses a force of $F_t = 0$ for all time steps and the pole stays in its initial (but instable) position for all parameter settings while the bang-bang controller is forced to give a non-zero force all the time.