Optimization in Robotics (Acrobot) Control

Harvard University

AM205 Advanced Scientific Computing: Numerical Methods

Final Project Report

Authors:
Hengte Lin
Taosha Wang
Yifan Wang

Professor:
Chris H. Rycroft

December 12, 2017

Abstract

Your abstract.

1 Introduction

An **acrobot**[1] (in Figure 1) is a planar two-link robotic arm in the vertical plane with an actuator at the elbow (the red point), but no actuator at the shoulder. An acrobot is a typical underactuated robots, which have less actuators than degrees of freedom. The control of underactuated systems has been an interesting topic in robotics industry for it "gives some reductions of numbers of necessary actuators, of the cost and of the weight of systems" [1].

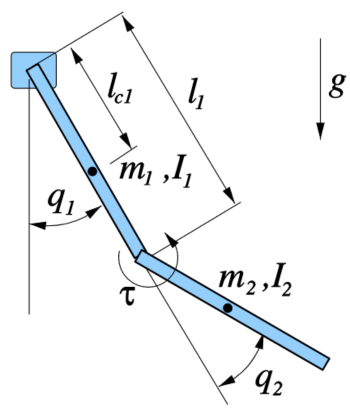

Figure 1: Acrobot (Modified on the basis of picture from [2])

An acrobot closely resembles the movement of a man, and is an important part of a robot. One of the most popular control task studied for the acrobot is the *swing-up* [3] task, in which the system must use the elbow torque to move the system into a vertical configuration and then balance. Previous studies have proposed many controllers for the Acrobot, such as energy based controllers [4][5], controllers based on partiallinearization [4][6], tracking controller [7], back stepping controller [8], a controller based on the

[1]A link to an animation of Acrobot: `http://www.princeton.edu/~rvdb/WebGL/Acrobot.html`

motion of the real gymnast [9] etc. Though optimal control is a powerful framework for specifying complex behaviors with simple objective functions, the computational tools cannot scale well to systems with state dimension more than four or five [2]. In this project, we attempt to find an optimal control solution that is valid from only a single initial condition, instead of solving for the optimal feedback controller for the entire state space. Thus, we represent the optimal control solution as a *trajectory*, $\mathbf{x}(\cdot)$, $\mathbf{u}(\cdot)$ rather than a feedback control function.

The rest of this report is organized as follows: In Section 2, we derive the motion equation of the acrobot using standard, manipulator equation form. In Section 3, we set up the Lagrangian equations and derive necessary conditions for the trajectory optimization problem. In Section 4, we present the results and runtime analysis of the trajectory optimization solutions. In Section 5, we apply a cutting-edge neural network model - the reinforcement learning algorithm to solve the trajectory optimization problem and compare the performance with traditional methods.

2 Dynamics of the Acrobot

In Figure 1, q_1 is the shoulder joint angle, q_2 is the relative joint angle at the elbow. With the notations in Figure 1 and let u represent the torque applied to the elbow, $\mathbf{q} = [q_1, q_2]^T$, $\dot{\mathbf{q}} = [\dot{q_1}, \dot{q_2}]^T$ and $\ddot{\mathbf{q}} = [\ddot{q_1}, \ddot{q_2}]^T$, we can derive the equations of motion in standard, manipulator equation form:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}u \tag{1}$$

where

$$\begin{aligned}
\mathbf{H}(\mathbf{q}) &= \begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2m_2 l_1 l_{c2} c_2 & I_2 + m_2 l_1 l_{c2} c_2 \\ I_2 + m_2 l_1 l_{c2} c_2 & I_2 \end{bmatrix}, \\
\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) &= \begin{bmatrix} -2m_2 l_1 l_{c2} s_2 \dot{q}_2 & -m_2 l_1 l_{c2} s_2 \dot{q}_2 \\ m_2 l_1 l_{c2} s_2 \dot{q}_1 & 0 \end{bmatrix}, \\
\mathbf{G}(\mathbf{q}) &= \begin{bmatrix} m_1 g l_{c1} s_1 + m_2 g (l_1 s_1 + l_{c2} s_{1+2}) \\ m_2 g l_{c2} s_{1+2} \end{bmatrix}, \\
\mathbf{B} &= \begin{bmatrix} 0 \\ 1 \end{bmatrix},
\end{aligned}$$

and u is the scalar control signal (torque acting on the second joint).

In the next steps we derive the deterministic dynamics from the above motion equations. First, we add frictions f to the system:

$$\mathbf{B}u = \mathbf{B}\hat{u} - f\dot{\mathbf{q}} \tag{2}$$

Substituting equation (2) into the right hand side of equation (1), we get:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}\hat{u} - f\dot{\mathbf{q}} \tag{3}$$

Afterwards, $\ddot{\mathbf{q}}$ can be calculated as:

$$\begin{aligned} \ddot{\mathbf{q}} &= \mathbf{H}(\mathbf{q})^{-1}[-\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) + \mathbf{B}\hat{u} - f\dot{\mathbf{q}}] \\ &= \mathbf{H}(\mathbf{q})^{-1}[-\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) - f\dot{\mathbf{q}}] + \mathbf{H}(\mathbf{q})^{\mathbf{-1}}\mathbf{B}\hat{u} \end{aligned} \tag{4}$$

Let $\mathbf{x} = [q_1, q_2, \dot{q_1}, \dot{q_2}]^T$, we can calculate its derivative with respect to time:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \ddot{q_1} \\ \ddot{q_2} \end{bmatrix} = \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \mathbf{H}(\mathbf{q})^{-1}[-\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) - f\dot{\mathbf{q}}] \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \mathbf{H}(\mathbf{q})^{\mathbf{-1}}\mathbf{B} \end{bmatrix} \hat{u}, \tag{5}$$

in the format of $\dot{\mathbf{x}} = \mathbf{f_d}(\mathbf{x}, u) = \mathbf{a} + \mathbf{b}u$.

For the simplicity of computation, we set all constant values (i.e., I_1, I_2, l_1, l_2, m_1, m_2, l_{c1}, l_{c2}, c_1, c_2) equal to 1, hence the matrix $\mathbf{H}(\mathbf{q})$, $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ and $\mathbf{G}(\mathbf{q})$ in equation 5 can be rewritten as:

$$\begin{aligned} \mathbf{H}(\mathbf{q}) &= \begin{bmatrix} 3 + 2\cos(q_2) & 1 + \cos(q_2) \\ 1 + \cos(q_2) & 1 \end{bmatrix} \\ \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} -2\sin(q_2)\dot{q_2} & -\sin(q_2)\dot{q_2} \\ \sin(q_2)\dot{q_1} & 0 \end{bmatrix} \\ \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} g\sin(q_1) + g(\sin(q_1) + \sin(q_1 + q_2)) \\ g\sin(q_1 + q_2) \end{bmatrix} \end{aligned}$$

And hence

$$\begin{aligned} -\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} 2\sin(q_2)\dot{q_1}\dot{q_2} + \sin(q_2)\dot{q_2}^2 \\ -\sin(q_2)\dot{q_1}^2 \end{bmatrix} - \begin{bmatrix} g\sin(q_1) + g(\sin(q_1) + \sin(q_1 + q_2)) \\ g\sin(q_1 + q_2) \end{bmatrix} \\ &= \begin{bmatrix} \dot{q_2}(2\dot{q_1} + \dot{q_2})\sin(q_2) - 2g\sin(q_1) - g\sin(q_1 + q_2) \\ -\dot{q_1}^2\sin(q_2) - g\sin(q_1 + q_2) \end{bmatrix} \end{aligned}$$

By the end of this section we have derived the $\dot{\mathbf{x}}$ (Equation 5). By applying $\dot{\mathbf{x}}$ to the deterministic dynamic function $\mathbf{x}[n+1] = \mathbf{x}[n] + \dot{\mathbf{x}}[n]$, we can calculate the next state given previous state and the torque, and hence we have derived the dynamics of the acrobot.

3 Trajectory Optimization

3.1 Lagrangian Constrained Optimization

Define the cost function as

$$g(\mathbf{q}, u) = \frac{r}{2}u^2 + 1 - \exp(-k\cos(q_1) + k\cos(q_2) - 2k), \tag{6}$$

where r, k are constants. The reason we choose to use cosine function ($\pm\cos(\cdot)$) as the cost function at both links q_1 and q_2 is to take advantage of the monotonic and cyclical characteristics of the cosine function. As we can see from Figure 2, $x = \pi$ (the red point) is a global minima, and no matter where we start within the boundary $[0, 2\pi]$ (the purple points), it will converge to $x = \pi$ and will end up with the global minima.

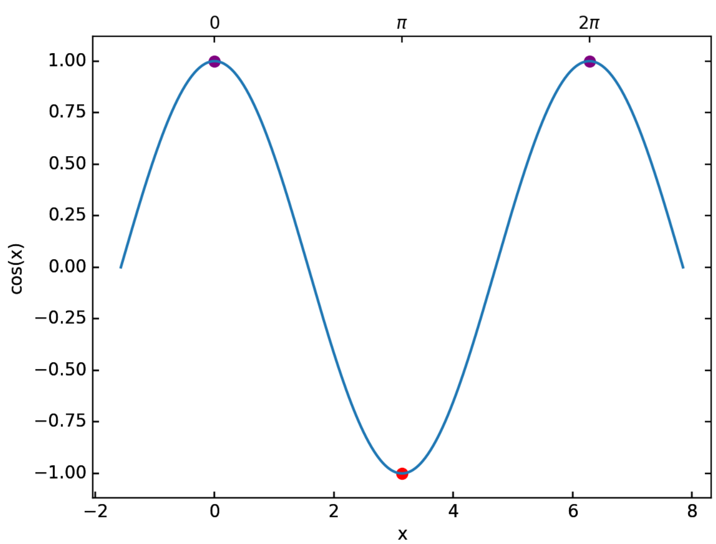

Figure 2: $\cos(x)$

In our settings (see Figure 1), q_1 is the angle between the upper arm and the vertical line going downwards and q_2 is the angle between the upper arm and the lower arm. At the beginning of the swing-up process, both arms were hanging downwards so $q_1 = 0$ and $q_2 = 0$ is the starting state. Afterwards, the torque goes into effect so that q_1 either increases towards 2π or decreases towards 0, and the global minimum can be achieved in both cases. In other words, the arms can swing up either clockwisely or counter-clockwisely, and either way will result in the desired state finally.

The constrained trajectory optimization problem in discrete time can be described as below:

$$\begin{aligned}
&\text{minimize}_{\mathbf{x}_1,\ldots,\mathbf{x}_N,u_0,\ldots,u_{N-1}} \sum_{n=0}^{N-1} g(\mathbf{x}[n], \mathbf{u}[n]),\\
&\text{subject to} \quad \mathbf{x}[n+1] = \mathbf{x}[n] + \mathbf{f_d}(\mathbf{x}[n], \mathbf{u}[n]).
\end{aligned} \tag{7}$$

We use Lagrange multipliers to derive the necessary conditions for our trajectory optimization problem:

$$L(\mathbf{x}[\cdot], \mathbf{u}[\cdot], \lambda[\cdot]) = \sum_{n=0}^{N-1} g(\mathbf{x}[n], \mathbf{u}[n]) + \sum_{n=0}^{N-1} \lambda^T[n] \left(\mathbf{f_d}(\mathbf{x}[n], \mathbf{u}[n]) - \mathbf{x}[n+1]\right) \tag{8}$$

Take first-order derivatives with respect to $\lambda[\cdot]$, $\mathbf{x}[\cdot]$ and $\mathbf{u}[\cdot]$ and set the values equal to 0's:

$$\begin{aligned}
\forall n \in [0, N-1], \frac{\partial L}{\partial \lambda[n]} &= \mathbf{f_d}(\mathbf{x}[n], u[n]) - \mathbf{x}[n+1] = 0\\
\Rightarrow \mathbf{x}[n+1] &= \mathbf{f_d}(\mathbf{x}[n], \mathbf{u}[n]) && (9)\\
\forall n \in [0, N-2], \frac{\partial L}{\partial \mathbf{x}[n]} &= \frac{\partial g(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}} + \lambda^T[n] \frac{\partial \mathbf{f_d}(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}} - \lambda^T[n-1] = 0\\
\Rightarrow \lambda[n-1] &= \frac{\partial g(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}}^T + \frac{\partial \mathbf{f_d}(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}}^T \lambda[n]. && (10)\\
\frac{\partial L}{\partial \mathbf{x}[N]} &= -\lambda[N-1] = 0\\
\Rightarrow \lambda[N-1] &= 0 && (11)\\
\forall n \in [0, N-1], \frac{\partial L}{\partial \mathbf{u}[n]} &= \frac{\partial g(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}} + \lambda^T[n] \frac{\partial \mathbf{f_d}(\mathbf{x}[n], u[n])}{\partial \mathbf{u}} = 0. && (12)
\end{aligned}$$

Specifically,

$$
\begin{aligned}
\frac{\partial g(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}[n]} &= \begin{bmatrix} -\exp(-k\cos(q_1) + k\cos(q_2) - 2k)\sin(q_1) \\ \exp(-k\cos(q_1) + k\cos(q_2) - 2k)\sin(q_2) \\ 0 \\ 0 \end{bmatrix} \\
\frac{\partial g(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{u}[n]} &= ru \\
\frac{\partial \mathbf{f_d}(\mathbf{x}[n], \mathbf{u}[n])}{\partial \mathbf{x}} &= \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{H}^{-1}\frac{\partial \mathbf{G}}{\partial \mathbf{q}} & -\mathbf{H}^{-1}(\mathbf{C} + \mathbf{I}f) \end{bmatrix} \\
\frac{\partial \mathbf{f_d}(\mathbf{x}[n], u[n])}{\partial \mathbf{u}} &= \begin{bmatrix} 0 \\ 0 \\ \mathbf{H}(\mathbf{q})^{-\mathbf{1}}\mathbf{B} \end{bmatrix}
\end{aligned}
$$

where

$$
\frac{\partial \mathbf{G}}{\partial \mathbf{q}} = \begin{bmatrix} 2g\cos(q_1) + g\cos(q_1 + q_2) & g\cos(q_1 + q_2) \\ g\cos(q_1 + q_2) & g\cos(q_1 + q_2) \end{bmatrix}
$$

We try to solve the above equation system using different numerical methods, looking for a series of torque $\mathbf{u}[\cdot]$ that satisfied all the constraints above and therefore gives us the optimal trajectory defined by our cost function (Equation 6). To start with, suppose we have a series of torques $\mathbf{u}[\cdot]$ that is randomly generated from a normal distribution. Since we already know the initial condition $\mathbf{x}[0]$, we can infer $\mathbf{x}[1]$ by Equation 9, which is the dynamic equation. In the same manner, we can infer the next state from current state, and thus we can compute $\mathbf{x}[2], \mathbf{x}[3], ..., \mathbf{x}[N-1]$ sequentially.

Then we compute the series of $\lambda[\cdot]$ in a reverse order: since $\lambda[N-1] = 0$ (Equation 11) and we already know the sequence $\mathbf{x}[\cdot]$ and $\mathbf{u}[\cdot]$, by Equation 10 we can infer $\lambda[N-1]$. Similarly, we can compute $\lambda[N-2], ..., \lambda[0]$ sequentially.

Recall that our goal is to find a sequence of torques $\mathbf{u}[\cdot]$ that satisfies the equation system Equation 9, Equation 10, Equation 11 and Equation 12. Now we have reduced the complicated problem with a bunch of unknowns to a typical root-finding one, with the help of the sequential relationship within $\mathbf{x}[\cdot]$ and $\lambda[\cdot]$. Thus we can solve for $\mathbf{u}[\cdot]$ using Newton's method or other root finding methods.

3.2 Chebyshev Approximation

Theoretically, the torque should change smoothly with time, and hence the values of $\mathbf{u}[\cdot]$ should be very similar at adjacent time points if the time step is small enough. Therefore, it might be a waste to optimize all of $\mathbf{u}[\cdot]$ individually, considering the large number of time points we care about. To reduce the dimensions of this problem, we tried to approximate the torque values $\mathbf{u}[\cdot]$ using polynomial interpolation. The basic idea of polynomial interpolation is to fit a polynomial curve $g(t)$ to the torque values $\mathbf{u}[\cdot]$ at selected time points.

In order to minimize the approximation error, we use Chebyshev interpolation [10], which is an orthogonal collocation method. In general, Chebyshev interpolation method chooses sample points at Chebyshev points c_j that are calculated using the equation below:

$$c_j = \cos\left(\frac{(2j-1)\pi}{2n}\right), j = 1, ..., n \tag{13}$$

where n is the order of Chebyshev polynomial. By approximating the torque with a polynomial function of time, now we can describe the torque with as small as n variables instead of a large number of time steps. Moreover, we no longer need to optimize the torque values $\mathbf{u}[\cdot]$ at each time step, but only need to optimize the torque values $\mathbf{w}[\cdot]$ at selected Chebyshev points $\mathbf{c}[\cdot]$.

To include Chebyshev interpolation in our implementation, we first choose the initial torque values $\mathbf{w}[\cdot]$ at Chebyshev points $\mathbf{c}[\cdot]$, and generate the polynomial function $g(t, \mathbf{w}, \mathbf{c})$ with Lagrangian polynomial method [11]:

$$\begin{aligned} g(t, \mathbf{w}, \mathbf{c}) &= \sum_{j=1}^{n} w_j l_j(t), \\ \text{where} \quad l_j(t) &= \frac{t - c_1}{c_j - c_1} \cdots \frac{x - c_{j-1}}{c_j - c_{j-1}} \cdot \frac{x - c_{j+1}}{c_j - c_{j+1}} \cdots \frac{x - c_n}{c_j - c_n} \end{aligned} \tag{14}$$

With $g(t, \mathbf{w}, \mathbf{c})$, we can compute the torque values at each time step $\mathbf{u}[\cdot]$. Following the steps described in Section 3.1 we can derive the gradient $\frac{\partial L}{\partial \mathbf{u}[\cdot]}$.

The gradient $\frac{\partial L}{\partial \mathbf{w}}$ can be calculated with the formula below:

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{i=0}^{T} g_w(t, \mathbf{w}, \mathbf{c})^T \frac{\partial L}{\partial \mathbf{u}[i]}, \tag{15}$$

where

$$g_w(t, \mathbf{w}, \mathbf{c}) = \begin{bmatrix} l_1(t) \\ l_2(t) \\ \dots \\ l_n(t) \end{bmatrix}.$$

$l_i(t)$ is defined in Equation 14 and T is total number of discrete time steps.

4 Results

5 Reinforcement Learning

References

[1] T. Henmi, Mingcong Deng, and A. Inoue. “Swing-up Control of the Acrobot Using a New Partial Linearization Controller Based on the Lyapunov Theorem”. In: *2006 IEEE International Conference on Networking, Sensing and Control*. 2006, pp. 60–65. DOI: 10.1109/ICNSC.2006.1673118.

[2] Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*. http://underactuated.mit.edu/.

[3] M. W. Spong. “The swing up control problem for the Acrobot”. In: *IEEE Control Systems* 15.1 (1995), pp. 49–55. ISSN: 1066-033X. DOI: 10.1109/37.341864.

[4] Y. Dong et al. “Energy-Based Control for a Class of Under-Actuated Mechanical Systems”. In: *2008 Congress on Image and Signal Processing*. Vol. 3. 2008, pp. 139–143. DOI: 10.1109/CISP.2008.304.

[5] A. Inoue et al. “Swing-up and stabilizing control system design for an Acrobot”. In: *2007 IEEE International Conference on Networking, Sensing and Control*. 2007, pp. 559–561. DOI: 10.1109/ICNSC.2007.372839.

[6] X. Z. Lai et al. “Comprehensive Unified Control Strategy for Underactuated Two-Link Manipulators”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2009), pp. 389–398. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2008.2005910.

[7] Z. Dengfeng et al. “An improved departed domain control strategy for acrobot”. In: *2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE)*. 2017, pp. 5–9. DOI: 10.1109/CCSSE.2017.8087884.

[8] T. Nam et al. “Swing-up Control and Singular Problem of an Acrobot System”. In: 20(1) (2002), pp. 85–88.

[9] T. Henmi et al. “Swing-up control of an Acrobot having a limited range of joint angle of two links”. In: *2004 5th Asian Control Conference (IEEE Cat. No.04EX904)*. Vol. 2. 2004, 1071–1076 Vol.2.

[10] J. Vlassenbroeck and R. Van Dooren. "A Chebyshev technique for solving nonlinear optimal control problems". In: *IEEE Transactions on Automatic Control* 33.4 (1988), pp. 333–340. ISSN: 0018-9286. DOI: 10.1109/9.192187.

[11] Branden Archer and Eric W. Weisstein. *Lagrange Interpolating Polynomial.* http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html.