Model Predictive Control of Acrobot

Qingmu Deng

qingmu@stanford.edu

May 17, 2024

1 Midterm Report

1.1 Motivation

Model predictive control, also known as online trajectory optimization, has been a powerful method in Controls and increasing gaining popularity in real-time application with the ever faster computing capabilities in robotic systems. For the final project for AA203 Optimal and Learning Based Control at Stanford, I propose to implement model predictive control for a simulated acrobot in a simulated environment. An example of the said acrobot and control can be seen in the first minute of [1].

Through the final project, my goal is to develop solid theoretical understanding and gain practical implementation experience in Python for model predictive control. I would also love to learn to build basic simulation in Python and use some of the simulated environments for developing and evaluating control policies as sometimes it may be expensive or impractical for an individual learner such as myself to acquire all the physical hardware for a humanoid robot or a drone and repair any damages incurred during the control experiments.

The acrobot is a series of two linkages rotating around one fixed pivot on one end. The pivot joining the two links is free to rotate around and move through space depending on the position of the first link. With an actuator in the central pivot, but no actuator at the end pivot, the acrobot is a highly representitive system of underactuated robots. That is, through our control policy, we must consider the coupling between states that can be directly actuated and those that cannot. The acrobot also resembles one of the simplest models of a walking robot.[2] To that end, this system opens the gate for me to consider a lot of realistic control scenarios where the system may be underactuated.

1.2 Technical Contributions

I am proposing to take my current project in the following stages.

Firstly, I will build a simple dynamical model and simulation of the acrobot in Python. While I have mention MuJoCo in my previous proposal, being able to use Mujuco for this simulation is certainly a stretch goal as the dynamics of acrobot is relatively straightforward to specify and simulate.

With the dynamics in hand, I will solve the open loop swing-up problem through the direct method, such as sequential convex programming(SCP), and then perform a trajectory optimization

procedure with swing-up and balancing of the acrobot, such as iterative Linear Quadratic Regular(iLQR). I plan to present these results as the basic point of reference for MPC performances. The iLQR solver developed here will also be useful in solving the receding-horizon MPC problem subsequently.

As a next step, I plan to implement an MPC loop for the swing-up and balancing of the acrobot. Here, we begin with the assumptions that we do not have any disturbances, and that we do not have any model inaccuracies. Where applicable, we will compare the performance of MPC with that of open loop and/or trajectory optimized solutions.

I will then try to introduce state disturbances as a way to test and develop the robustness of the MPC loop. If I am able to demonstrate the robustness of my MPC implementation against state disturbances, I plan to also test the robustness of the MPC through model inaccuracies, such as moments of inertia slightly different from the MPC model.

I think the above details should already constitute a very thorough one-person project for learning about MPC. Nonetheless, if my time and bandwidth allows, I would love to learn to transfer the above MPC to a MuJoCo simulation as a way to learn about simulated robotic environments. This, however, shall remain a stretch goal.

2 Related Work

One of the primary references for my project is the online trajectory optimization paper by Tassa and colleagues.[3] This paper provides a detailed background on trajectory optimization procedures and related recommendation on regularization and line search to improve the solution convergence and avoid divergent results. Equally, interesting was the paper's discussion of cost function choices. We have typically specified quadratic state and control cost terms, the paper proposed using a "smooth absolute" function for the state cost and hyperbolic trig function for control costs. The equations are specified as follows, and α is a tunable parameter. The main benefita of these alternative cost function choices are that the state cost is then unit preserving of the physical system and can be better presented and that the control cost will not lead to undefined or infinite control values.

$$\ell(x) = \sqrt{x^2 + \alpha^2} - \alpha \tag{1}$$

$$\ell(u) = \alpha^2(\cosh(u/\alpha) - 1) \tag{2}$$

While the paper also spent a significant section on the development of physics simulation engine MuJoCo, I am not too concerned how to write a fast simulation engine with contact dynamics. This is more good-to-know background for MuJoCo if I do come to use it by the end of the project.

Another useful academic reference is the paper by Todorov and Li that formally introduces the iterative Linear-Quadratic-Gaussian methods for solving locally-optimal feedback control of nonlinear stochastic systems subject to control constraints.[4]

For the dynamics of the acrobot and introductions and tutorials on controller implementation in Python, I found Russ Tedrake's Underactuated Robotics comprehensive. Organized around Jupyter Notebooks, the textbook and associated exercises provided hands-on introduction to the dynamics of acrobot and the implementation of trajectory optimizers.[2]

3 Problem Statement

3.1 System Dynamics

The acrobot is a series of two linkages of uniform mass rotating around one fixed pivot on one end. We can this joint the shoulder. The pivot joining the two links is free to rotate around and move through space depending on the position of the first link. We address this join as the elbow. With an actuator in the central pivot, but no actuator at the end pivot, the acrobot is a highly representative system of underactuated robots. That is, through our control policy, we must consider the coupling between states that can be directly actuated and those that cannot. The acrobot also resembles one of the simplest models of a walking robot.[2]

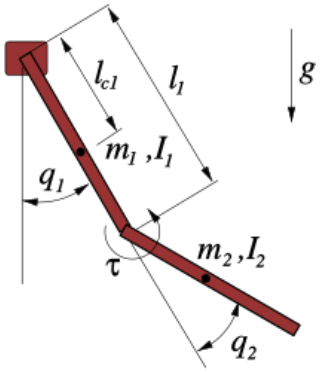

Figure 1: Acrobot system illustration. Image Source [2]

Using Lagranian formulation, we can write the acrobot dynamics in the manipulator equation form:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \boldsymbol{\tau}_g(\boldsymbol{q}) + \boldsymbol{B}\boldsymbol{u},$$

using $\boldsymbol{q} = [\theta_1, \theta_2]^T$, $\boldsymbol{u} = \tau$ we have:

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} I_1 + I_2 + m_2 l_1^2 + 2m_2 l_1 l_{c2} c_2 & I_2 + m_2 l_1 l_{c2} c_2 \\ I_2 + m_2 l_1 l_{c2} c_2 & I_2 \end{bmatrix}, \tag{3}$$

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} -2m_2 l_1 l_{c2} s_2 \dot{q}_2 & -m_2 l_1 l_{c2} s_2 \dot{q}_2 \\ m_2 l_1 l_{c2} s_2 \dot{q}_1 & 0 \end{bmatrix}, \tag{4}$$

$$\boldsymbol{\tau}_g(\boldsymbol{q}) = \begin{bmatrix} -m_1 g l_{c1} s_1 - m_2 g (l_1 s_1 + l_{c2} s_{1+2}) \\ -m_2 g l_{c2} s_{1+2} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{5}$$

where we $\theta_1 = q_1$ is the shoulder angle , and $\theta_2 = q_2$ is the elbow angle. I_1 and I_2 is the moment of inertia of each link with respective to their pivots. l_1 and l_2 denotes the length of each arm linkages, where as l_{c1} and l_{c2} denotes the length from the pivot point to each link's own center of mass, which in this case is just half l_1 and l_2 assuming a uniformly distributed rod. The abundance of cosine and sine have prompted the following notation

$$\cos\theta_1 = c_1 \tag{6}$$

$$\sin(\theta_1 + \theta_2) = s_{1+2} \tag{7}$$

At a high level, $\boldsymbol{M}$ is the inertia matrix, $\boldsymbol{C}$ describe the Coriolis forces, and $\boldsymbol{\tau}_g$ is the gravity vector.

To make a tractable control and simulation problem, we now convert the above second order differential equation into a system of first-order differential equations

$$\begin{bmatrix} I_{2\times 2} & 0_{2\times 2} \\ \boldsymbol{C}(\boldsymbol{q}) & \boldsymbol{M}(\boldsymbol{q}) \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \boldsymbol{\tau}_g(\boldsymbol{q}) + \boldsymbol{B}\boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ -m_1 g l_{c1} s_1 - m_2 g (l_1 s_1 + l_{c2} s_{1+2}) \\ -m_2 g l_{c2} s_{1+2} + \tau \end{bmatrix} \tag{8}$$

With the acrobot system, we are interested in swinging the two links into up-right position and maintain it there. To that end, we will begin by exploring both the open loop sequential convex programming and the trajectory optimization solution with iLQR before adapting elements from these solutions for the full MPC policy.

3.2 Controller Design

3.2.1 Model Predictive Control

The important learning goal of this project is the implementation of model predictive control for the swing-up and balancing of the acrobat. The algorithm can be broadly summarized as the following.

At time instance t_k, we observe our system to be in state $x(t_k)$. For a finite time horizon of length N, i.e. $\{t_{k+1}, t_{k+2}, ..., t_{k+N}\}$, we solve the trajectory optimization problem to obtain the optimal control sequence

$$U^*(x(t_k)) := \{u_k, u_{k+1}, ..., u_{k+N-1}\}$$

we then implement the first control step u_k, which will lead us to the system state $x(t_{k+1})$ at the next time step t_{k+1}. Then, we can iterate through the algorithm again for $x(t_{k+1})$ and all future states. Because of the moving time horizon of this scheme, model predict control, also known as online trajectory optimization, is also named receding horizon control in some literature. The textbook by Borrelli and et al. provides a good overview of the theoretical foundation of such receding horizon control with regards to their feasibility and stability.[5]

3.3 Trajectory Optimization

Following the work by Tassa and et al., I will be using iLQR method to perform trajectory optimization. The iLQR method is based on a linear approximation of the system dynamics and a quadratic approximation of the cost function. A single iteration of the iLQR method is summarized below

1. Backward Pass: Given a nominal state and control sequence, we calculate the derivatives of cost function and system dynamics. With these derivatives, we iterate backwards from t_N to t_0 to obtain the optimal perturbation to our nominal control sequence and update our quadratic approximation.

2. Forward Pass: Based on the optimal control perturbation, we propagate the control and state sequence forward based on system dynamics to obtain a new nominal state and control trajectory.

4 Approach

5 Experiments

6 Conclusions and Future Work

References

[1] Y. Tassa, T. Erez, and E. Todorov, "Model predictive control with ilqg and mujoco," YouTube, May 2019.

[2] R. Tedrake, *Underactuated Robotics*. 2023.

[3] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

[4] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, pp. 300–306 vol. 1, 2005.

[5] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.