Uneven Bars Robot

1st Mohammed Ehab
EECS Department
MIT
Cambridge, USA
ehab02@mit.edu

***Abstract*—The aim of this project is to simulate a robot that can perform in the uneven bars event in gymnastics. The robot should be able to start from rest on the higher bar and swing itself onto the lower one. Through hybrid trajectory optimization, a controller was synthesized that swings the robot on the high bar, launches it, and corrects its posture in the air, all in the span of less than 3 seconds.**

***Index Terms*—Hybrid Trajectory Optimization, Robotics**

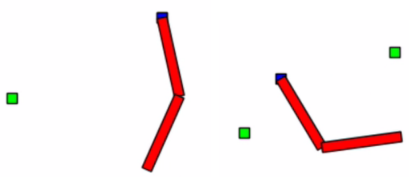
I. Introduction

Uneven bars is a popular gymnastics event where gymnasts have 2 horizontal bars with differing heights, and they can swing from one bar to the other while performing acrobatic moves. The horizontal distance between the bars is roughly twice as big as the vertical one, so they have to use a very precise technique to swing themselves up, launch with their back hollowed to get the greatest horizontal velocity, and arch their back in the opposite direction on the fly to get closer to the second bar and catch it. Synthesizing such a complex controller without any priors, especially with a general-purpose solver like SNOPT, is a really interesting problem.

II. Model

Gymnasts don't tend to use their wrists for swinging because it's very difficult. Instead, they rely on arching and hollowing their core to generate swing, making them underactuated.

To that end, the robot is modeled as 2 pendula with massive rods, connected at their ends with a continuous joint. The joint has a transmission corresponding to a human's core. Furthermore, the robot has a hook on top that's connected to the first pendulum with another continuous joint, representing the wrists. The hook is connected to the world with a floating joint, and when the robot is on one of the bars, that joint should be locked.



III. Method

The problem can be formulated as a hybrid trajectory optimization problem where the following constraints are required: the robot's initial state should be at rest, its final position should be on the second bar, and it should follow the dynamics when it's hooked for a while then follow the free-floating dynamics. It's not necessary to have a cost, but in order to make the trajectory more realistic, we will penalize using too much control. Using the direct transcription method with implicit Euler integration, the problem could be formulated as follows:-

$$\begin{aligned}
\min_{u,q,\dot{q},\ddot{q},h} \quad & \sum_{t=0}^{T-1} u_i^2 \\
\text{s.t.} \quad & q_0 \text{ is at rest} \\
& q_T \text{ is on the lower bar} \\
& h_{min} \leq h_t \leq h_{max} \; \forall t \\
& q_{t+1} = q_t + h_t \dot{q}_t \; \forall t \\
& \dot{q}_{t+1} = \dot{q}_t + h_t \ddot{q}_t \; \forall t \\
& ManipulatorEqns(q_t, \dot{q}_t, \ddot{q}_t, u_t, mode) = 0 \; \forall t
\end{aligned} \tag{1}$$

Where $ManipulatorEqns$ is a function that returns the error in the manipulator equations. To ensure the robot swings for some time then unhooks, the function is passed the mode, which is on the bar when $t < \frac{T}{2}$ and in the air when $t \geq \frac{T}{2}$. This is also why h was chosen to be a decision variable: the amount of time spent on the bar versus in the air is unknown prior to optimization.

IV. Challenges

The main implementation challenge is how to enforce the dynamics in both modes which are fundamentally quite different. If working in minimal coordinates, the robot only has two degrees of freedom on the bar, but once it unhooks and launches into the air, it gains six more.

The initial solution was to create two URDFs, one with the robot hooked to the bar and one with the robot free. Then, use the dynamics coming from each URDF on the decision variables of the corresponding mode, and manually write a constraint that insures the final conditions of the first mode correspond to the initial conditions of the second one. The constraint would essentially be a map of how $(\theta, \dot{\theta})$ translate into velocities when the robot unhooks from the bar.

Since writing constraints manually is laborious and doesn't scale well, a better solution was to be found. Indeed, working in maximal coordinates the whole time would fix the issue of the states being different, but it raises the following question: how do we obtain the manipulator equations when the robot is on the bar?

Essentially, we want some way to lock the floating joint between the hook and the world. The next attempt was to use the Drake Lock() function, hoping it would automatically alter the manipulator equations and give the correct dynamics. However, a deeper look shows this is not the case. Indeed, the manipulator equations take the form $M\ddot{q} + C\dot{q} = \tau_g(q) + Bu$. None of these terms depend on the joint being locked or not. Instead, locking a joint adds a new term corresponding to the constraint force needed to keep that joint locked, and this term needs to be independently modeled and added to the manipulator equations.

V. Modeling the Joint Locking Constraint

The joint being locked could be modeled as a kinematic constraint of the form $h(q) = 0$. The manipulator equations then become $M\ddot{q} = \tau(q, \dot{q}, u) + H^T\lambda$, where $\tau(q, \dot{q}, u)$ abstracts the Coriolis, gravity, and actuation terms, $H = \frac{\partial h}{\partial q}$, and λ is the constraint force [1]. In the case of locking a joint, $h(q) = Hq$, and H is a diagonal matrix with ones at the indices corresponding to states that joint controls.

The trivial way to model locking the joint is to compute the constraint force λ using equation (10) in [1] and add that term to the manipulator equations. However, a more careful look reveals an amazing insight that makes the solution easier. Since the matrix H has zeros in rows corresponding to unlocked joints, the constraint forces have no effect whatsoever on these coordinates. And for the coordinates of the locked joint, since its velocities are zero, the manipulator equations simplify to $\ddot{q} = 0$. The constraint forces are not this annoying term we have to calculate; they are whatever it takes to cancel the other terms and make the accelerations zero, without having any influence on the other coordinates! With this way to view the equations in mind, locking a joint, in general, can be implemented by discarding the coordinates corresponding to it from the error in the manipulator equations and setting $\ddot{q} = 0$ instead. It's both a sufficient and necessary condition, it's very easy to implement, and it's the kind of constraint MathematicalPrograms just love.

VI. Results

With that implementation of joint locking, the optimization ran smoothly, and SNOPT was indeed able to find the right technique. The code and animation could be found here.

VII. Future Directions

Hybrid trajectory optimization worked great for swinging the robot from one bar to another, but there are better solutions for more complex problems. For example, if we want the robot to perform an entire routine, we'd need to stitch trajectories together, forcing us to use around an hour of computation to create one memorized routine. Furthermore, if we ever build a physical robot to do this, its state when it lands will be very uncertain because of the hooking mechanism.

A controller that's able to swing the robot from any initial state would solve both problems. If we have that, we can create a library of trajectories that goes from one bar to the other, randomly pick one on the fly, then use that controller to swing the robot to the starting state of that trajectory and let go. The robot would be able to perform diverse and complex routines, even a physical one.

Formally, an approach stronger than hybrid trajectory optimization is splitting the problem into two sub-problems: a trajectory optimization problem for only the part where the robot is in the air, and a controller that can swing it from any initial to some desired final state while it's on the bar. The trajectory will be optimized to find the most feasible initial conditions (e.g. the ones with smallest angular velocities,) and then the controller would swing the robot from any state it finds itself in to the start of the trajectory, and that's when the robot unhooks.

The trajectory optimization component of this idea was already implemented in this notebook, and it found a quite acrobatic trajectory. The controller could perhaps then be synthesized via neural fitted value iteration or alternations - to be further investigated.

References

[1] "Ch. 22 –MultiBody Dynamics." http://underactuated.mit.edu/multibody.html (accessed May 16, 2023).