Trajectory Optimization for a 2D Hopper

Nicolas Arons

***Abstract*—Trajectory optimization allows for control of high-dimensional, nonlinear systems by trying to solve for the optimal control from only one initial condition. A two dimensional hopper was deemed a good candidate for trajectory optimization due to its nonlinearity. Due to the stiff, nonlinear dynamics of the hopper, a predetermined mode sequence sequence for the hopper's contact with the ground was utilized to help the solver find a solution. Optimal solutions were found for a hopper jumping up stairs and for jumping long distances.**

I. Introduction

In Marc Raibert's 1984 paper "Hopping in Legged Systems - Modeling and Simulation for the Two-Dimensional One-Legged Case," he described how the control of a two dimensional hopper can be broken up into the smaller tasks of controlling the vertical hopping height, controlling the horizontal velocity, and controlling the attitude of the body. Decomposing the control in this way proved useful for the hoppers as Raibert was bale to successfully simulate and stabilize a hopper using this method [1]. This version of control does not come without drawbacks, however; by simplifying control and separating it, it is possible that some trajectories and parts of the state space are ignored, leading to solutions that are not necessarily optimal. In order to solve problems that Raibert's original controller couldn't, I tried implementing contact trajectory optimization on the 2d hopper system. Due to the high nonlinearity of the dynamics, it was deemed infeasible to linearize the dynamics or to try to make the optimization problem convex. Because of this, MIQP and LQR solvers could not be used, so SNOPT was chosen for its ability to solve non-convex problems.

II. System Dynamics

The model for the hopper is given in Figure 1. The position vector q is made up of x, y, θ_1, θ_2 and z. The dynamics equations were copied from `https://github.com/RobotLocomotion/drake/blob/last_sha_with_original_matlab/drake/examples/PlanarMonopodHopper/HopperPlant.m`.

An interesting note about these equations is that there seems to be a problem when the body is at the center of mass. When $z = -0.5$, the equations seem to cancel to 0, as the solver returned some impossible solutions, with entire robot moving a meter and rotating half a radian in the span of 200 milliseconds. To help fix this issue, I tried to implement a constraint such that $z \neq -0.5$. Due to the solver requiring $\geq$ and not accepting $>$, I approximated the constraint by saying $(z+0.5)**2 >= \varepsilon$, where ε ended up being 0.00001. This did not appear to affect solutions, as the values of z are discretized and so they can pass by -0.5, but it stops the solver from finding trivial solutions to the dynamics with $z = -0.5$.

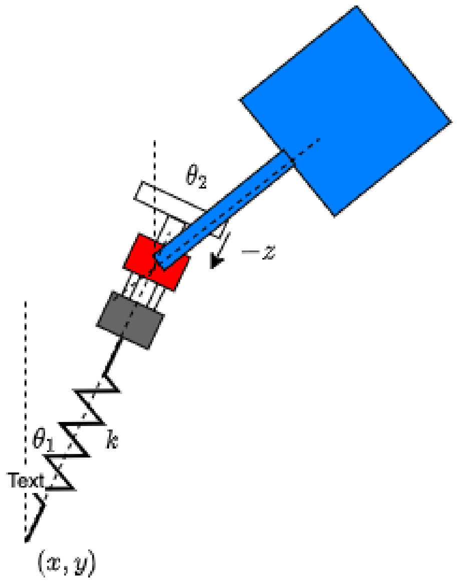

Fig. 1. The 2D Hopper model. x and y represent the position of the foot with θ_1 representing the angle. The body's (blue block) angle is represented by θ_2 and distance from the top of the leg is indicated by z, with down being negative. The torque actuation happens where the body meets the leg at the red hip joint, and the preload distance is between the red joing and the gray piece.

A. Modeling Contact

The contact between the ground and toe of the hopper and between the body and leg were modeled as springs following Hooke's law. This helped the nonlinear solver as it helped "soften" the very stiff dynamics of the hopper. Additionally, it was easy to implement. Although one might expect a system with two masses and one spring to resonate forever, the Backward Euler discrete time approximation of continuous time added effective damping to both of the springs.

Friction between the toe and ground was assumed to be infinite, and was modeled as $F_x = -C * \dot{x}$ when $y <= 0$. This was the easiest friction force to implement, as it did not have any stiff, binary if statements that implementing coulomb friction would give.

I found it easiest to implement contact forces implicitly in the dynamics rather than use them as decision variables with additional constraints.

B. Predetermined Contact Schedule

Initially, the nonlinear solver had a very tough time figuring out where and when contacts with the ground should occur. To help facilitate this process, a predetermined contact schedule was used. A first guess estimation was made at when the leg would be in contact with the ground, and how often. In both cases shown later, there was one contact placed in the middle of the time steps. At each time step there are different constraints depending on the contact schedule. If the leg is not in contact with the ground, then the y position is constrained to be above 0, and the dynamics for the hopper in flight are used. If leg is contacting the ground, then the y position is less than 0 and the dynamics change, with added forces coming from the ground in terms of friction and the normal force. Because the time steps before and after contact were evenly spaced, but the timing and length of the contact period was unknown, three separate time step values were used. These represented the time in the air before contact, during contact, and the time in the air after contact, and were passed to the solver as decision variables.

III. Jumping Up Steps

One problem identified that seemed a good candidate for trajectory optimization was that of jumping up a set of steps. In order to simplify the problem, the steps were assumed to be uniform. This meant that the problem could be treated as a more traditional steady state gait problem. This implies that the initial and final states are the same, except for a translational difference in x and y denoted by $\vec{v}$. For this specific problem, it also meant that only one contact was required in between each step. The binary predetermined value keeping track of contact was denoted by $g(n)$. Adding in a cost based on control effort, $u^t Ru$, results in the solver finding an optimal control for the specific trajectory. The optimization problem formulation is given by:

$$\begin{aligned}
\min_{q,\dot{q},\ddot{q},u,h} \quad & \sum_{n=0}^{N} h(n) * u^T Ru \\
\text{s.t.} \quad & q(n+1) = q(n) + h(n) * \dot{q}(n+1), \\
& \dot{q}(n+1) = \dot{q}(n) + h(n) * \ddot{q}(n), \\
& \ddot{q}(n) = f(q(n+1), \dot{q}(n+1), u(n)), \\
& q(N) = q(0) + \vec{v}, \\
& \dot{q}(N) = \dot{q}(0), \\
& \begin{cases} q(n+1) <= 0 & g(n) = \text{True} \\ q(n+1) >= 0 & g(n) = \text{False} \end{cases} \\
& \begin{cases} \ddot{q}(n) = f_1(q(n+1), \dot{q}(n+1), u) & g(n) = \text{True} \\ \ddot{q}(n) = f_2(q(n+1), \dot{q}(n+1), u) & g(n) = \text{False} \end{cases} \\
& h(n) = \begin{cases} h_g & g(n) = \text{True} \\ h_1 & g(n) = \text{False}, n < N/2 \\ h_2 & g(n) = \text{False}, n >= N/2 \end{cases} \\
& h_{min} <= h_g, h_1, h_2 <= h_{max}, \\
& u_{min} <= u <= u_{max}
\end{aligned} \tag{1}$$

The solver was able to find a solution, and the results are shown in figures 2 and 3.

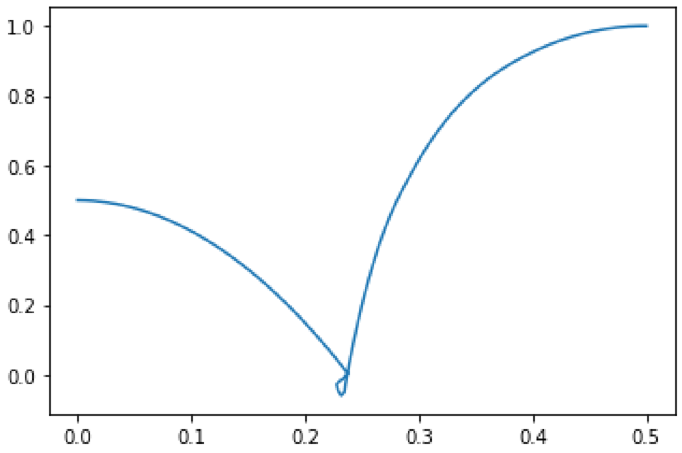

Fig. 2. The (x, y) position of the leg for the optimized stair climbing trajectory. The leg starts at $x = 0$, $y = 0.5$, hits the ground, and bounces up to the final position, $x = 0.5$, $y = 1$.

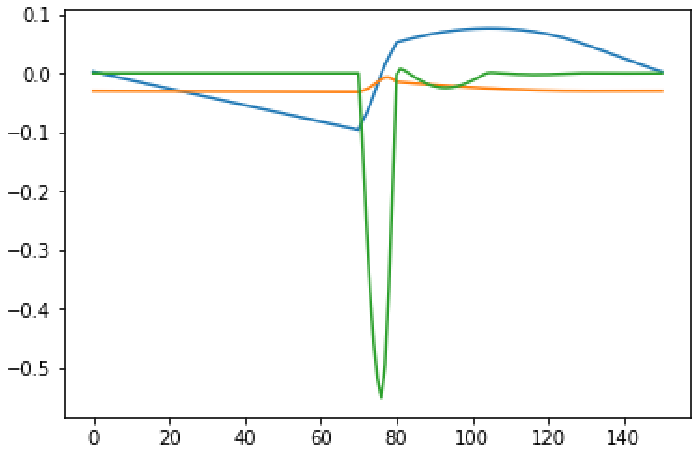

Fig. 3. θ_1 (blue), θ_2 (orange), and z (green) of the leg for the optimized stair climbing trajectory

A. Issues

A number of issues were experienced while working through this problem. Although z was never below a value of -0.8, as seen in figure 3, changing the constraint from $0.1 >= z >= -1$ to $0.1 >= z >= -0.8$ caused SNOPT to not find a solution. This issue persisted even when SNOPT was given the initial guess of the working solution.

Changing the step size, or offset vector $\vec{v}$, by small amounts resulted in failure for the solver. I found 0.5 to work for change in x and y, but value such as 0.4 and 0.6 failed.

IV. Long Jump

Another idea I had to test the fesability of trajectory optimization applied to the hopper was to try to optimize for the fastest steady state jumping speed, trying to mimic a sprint. The problem formulation for the long jump is very similar to climbing stairs, with a few key differences. There is an added cost of the form $-c * \dot{x}(0)^2$, which maximizes the speed at the initial time step. Because the system is cyclical, there is also no longer a prescribed initial y value, which let the solver instead choose the y value that minimizes the cost. Addtionally, the constraint on $q(N) = q(0) + \vec{v}$ was removed. Additionally there were some constraints added such as $y(0) >= 0.4$ and $\dot{x}(0) >= 0.1$ that were found necessary in order for the solver to come to a nontrivial solution. Figures 4 and 5 show the solution for the position variables.

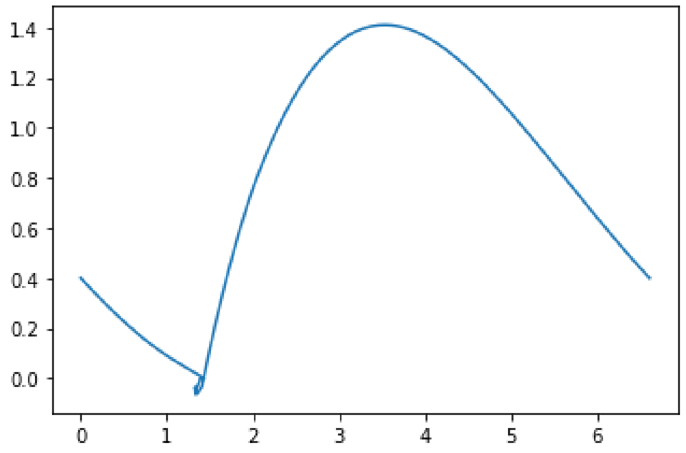

Fig. 4. The (x, y) position of the leg for the optimized long jumping trajectory.

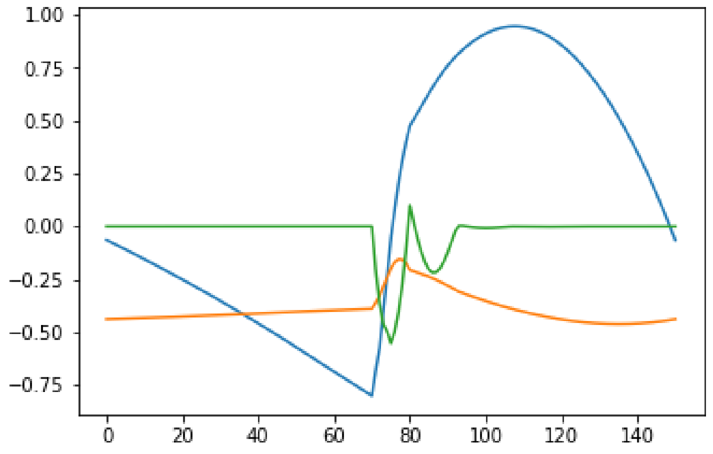

Fig. 5. θ_1 (blue), θ_2 (orange), and z (green) of the leg for the optimized long jumping trajectory.

A. Issues

Changing the coefficient c on the cost $-c * \dot{x}(0)^2$ often had wildly different results. I found when $c = 5$ the solver found a faster, farther traveling system than when $c = 10$, which seems counterintuitive. Additionally, setting $c = 9$ caused the solver to not find a solution at all. In the same vein, I was unable to get the cost $a * x(N)^2$ to find a solution, which is more direct than the cost used for the application of maximum distance in one jump.

V. Common Features

An interesting note about both trajectories is that the minimum value for z corresponds with both θ_1 and θ_2 being close to 0. When z is at its minimum value, the force on the body is highest, so any non-zero angle will result in large torques. Because the controller is attempting to minimize control effort, the optimal trajectory passes through this point to help lower the control effort needed.

While both trajectories are shown to be repetitive and consistent, no stability analysis was performed on them to ensure stability in the event of disturbances. Additionally, it is not known whether a robot could dynamically move between these trajectories or if it needs to be thrown into them. Observation indicates that the step jumping trajectory could easily be accessed by many initial states near it, as the velocities are low. It is less clear, however, for the long jump trajectory.

VI. Conclusions

Trajectory optimization shows some promise for the two dimensional hopper system. An optimization problem formulated around two common movement scenarios was shown to be effective in finding optimal solutions. There is still much unproven about this method, however. Significant simplifications were made to the contact models in order to soften the dynamics, which could mean the mathematical program doesn't represent the real system dynamics well.

Additionally, SNOPT was found to be incredibly finicky as a solver, with small adjustments to cost coefficients resulting in large changes in the results. While solutions were found for the tasks mentioned, there were a number of failed attempts, often requiring a "guess and check" method with the initial conditions or the cost coefficients to get the solver to find a solution. Due to the seemingly random , I would not recommend that any two dimensional hopper robot implement a SNOPT trajectory optimization controller online.

VII. Keyword

Underactuated :)

References

[1] Raibert, M. H., "Hopping in legged systems-modeling and simulation for the two-dimensional one-legged case," IEEE Trans. Syst., Man Cybern. 14 (3), 451–463 (May/Jun. 1984).