OCRL Final Project: Trajectory Optimization on a Two-Link Hopper

1st Brandon Hung
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, USA
GitHub Link: https://github.com/BrandonHung343/OCRL-Final

***Abstract*—This project applies trajectory optimization to a single two-link revolute-revolute joint leg to simulate hopping movements. By applying the hybrid method of dynamics simulation and assigning appropriate knot points to the stance phase and the flight phase models, a constrained optimization problem is created. Solving this problem with IPOPT allows us to track interesting movements and behaviors, including a forward hop and a stationary backflip.**

***Index Terms*—robots, hopper, hybrid, trajectory optimization**

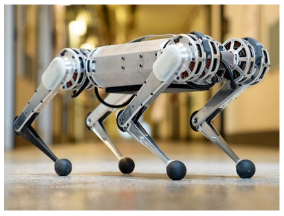
I. Introduction

With advances in computer hardware and control algorithms, legged robotics and their more complex dynamical descriptions have become an increasingly popular area of research. The increased mobility of legged robots over conventional methods such as wheels or treads grants them the potential to traverse otherwise impassible obstacles and terrain, making them particularly useful for field robotics. By combining these various motions together, legged robots can produce interesting movements useful for solving a variety of mobility-related tasks.

One of the simplest models of a leg is one composed of two links connected by a revolute joint. In practice, this model is not particularly useful because it is severely underactuated. By introducing another revolute joint at the end of one links and using it as a base for the robot, the leg gains an additional degree of freedom necessary to fully actuate its body in a 2D plane. Consequently, this type of leg is present on many popular quadrupeds, including Boston Dynamic's Spot Mini and ETH Zurich's ANYMAL; when combined with a degree of freedom in the robot's hip, the leg gains a limited range of 3D motion.

Fig. 1: Notable quadrupeds employing RR legs

(a) Boston Dynamics Spot


(b) MIT Cheetah

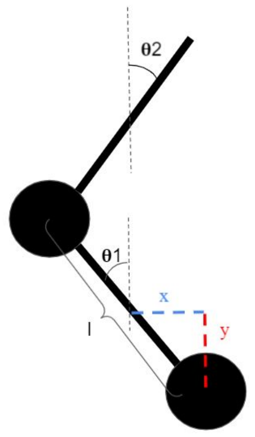
One of the most famous robots in dynamic locomotion is the Raibert hopper, a planar hopping robot which relies on dynamical stability through constant movement. A benefit over the Raibert hopper that the planar RR leg provides is a potential for dynamic stability when on the ground, provided there are sufficient contact forces. As a result, the planar RR leg is able to produce more complex behaviors, which are explored in this project.

II. Implementation

A. Model

The model used is a planar RR robot based off a two link robotic arm. The diagram below depicts the model as well as its parameterization:

Fig. 2: Planar RR Model



θ_1, θ_2 are measured with respect to the axis parallel to the y-axis passing through the center of mass of each link, l represents the length of each link, and x and y represent the distance of the center of mass of link 1 from the ground. This choice was made to simplify the dynamics of the system.

B. Dynamics

The dynamics for this model are derived from the Lagrangian of the following form:

$$L = T - V \tag{1}$$

where T is the kinetic energy and V is the potential energy of the system. The mass m of the model is 1 kg, while the length of each link is 1 meter. The potential energy V is given by the following equation:

$$V = mgy + (mgy + \frac{l}{2}cos\theta_1 + \frac{l}{2}cos\theta_2) \tag{2}$$

The kinetic energy T can be found by the following equation:

$$T = \frac{1}{2}m(v_1^T v_1) + \frac{1}{2}m(v_2^T v_2) + \frac{1}{2}J_1\omega_1 + \frac{1}{2}J_2\omega_2 \tag{3}$$

where

$$v_1 = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \tag{4}$$

$$v_2 = \begin{bmatrix} \dot{x} - \frac{l}{2}\cos(\theta_1)\dot{\theta}_1 - \frac{l}{2}\cos(\theta_2)\dot{\theta}_2 \\ \dot{y} + \frac{l}{2}\sin(\theta_1)\dot{\theta}_1 + \frac{l}{2}\sin(\theta_2)\dot{\theta}_2 \end{bmatrix} \tag{5}$$

$$J_1 = \frac{1}{12}ml^2 \tag{6}$$

$$J_2 = \frac{1}{12}ml^2 \tag{7}$$

$$\omega_1 = \dot{\theta}_1 \tag{8}$$

$$\omega_2 = \dot{\theta}_2 \tag{9}$$

The dynamics are derived from the Euler-Lagrange equation:

$$\frac{d}{dt}(\frac{dL}{d\dot{q}}) - \frac{dL}{dq} = 0 \tag{10}$$

where q is the state vector represented by

$$q = \begin{bmatrix} x \\ y \\ \theta_1 \\ \theta_2 \end{bmatrix} \tag{11}$$

Solving the Euler-Lagrange equation yields an expression of the form:

$$M(q,\dot{q})\ddot{q} + C(q,\dot{q}) = Bu - J^T\lambda \tag{12}$$

where

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{13}$$

$$J = \begin{bmatrix} 1 & 0 & \frac{l}{2}\cos(\theta_1)\dot{\theta}_1 & 0 \\ 0 & 1 & 0 & \frac{l}{2}\sin(\theta_2)\dot{\theta}_2 \end{bmatrix} \tag{14}$$

u is the control vector representing the torques on each link, and λ is a vector representing the contact forces on the foot of the robot.

$$u = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{15}$$

$$\lambda = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{16}$$

$M(q,\dot{q})\ddot{q}$ is the mass matrix and $C(q,\dot{q})$ is a vector. We solve for $\ddot{q}$ subtracting $C(q,\dot{q})$ from both sides and left multiplying by the inverse of the mass matrix:

$$\ddot{q} = M(q,\dot{q})^{-1}(Bu - J^T\lambda - C(q,\dot{q})) \tag{17}$$

The code for implementing the dynamics was provided by Professor Manchester using the Symbolics.jl package after my several unsuccessful attempts at a numerical description.

C. Simulation

In order to simulate the dynamics, we use an implicit Euler integrator of the form

$$x_n = x_n + h(f(x_{n+1}, u_n, \lambda_n)) \tag{18}$$

x_n represents the state at time n, u_n represents the control at time n, λ_n represents the contact forces at time n, and h represents the timesteps. Solving for x_{n+1} is done using the Newton method.

In order to differentiate between the flight phase of the dynamics (when the robot is not in contact with the ground) and the stance phase (when the robot is anchored to the ground), the contact forces are considered. In flight phase, both contact forces are 0; in stance phase, the contact forces ensure the robot has a zero velocity at the foot to simulate perfect friction with the ground.

Once again, the code for the implicit Euler solver was provided by Professor Manchester, with minor modifications by me to account for flight phase dynamics as well.

III. Non-linear Programming

A. Cost

The objective provided for the non-linear programming problem to solve is a linear-quadratic cost of the form

$$min_z \frac{1}{2}(z - z_{ref})^T Q(z - z_{ref}) + p^T(z - z_{ref}) \tag{19}$$

where z is a combined vector of the state and control vectors at each timestep.

$$z = \begin{bmatrix} x_1 \\ u_1 \\ x_2 \\ u_2 \\ . \\ . \\ . \\ x_N \end{bmatrix} \tag{20}$$

The cost Q is a scaled identity matrix, while p is a vector of the form

$$p^T = \beta(z - z_{ref})^T I \tag{21}$$

B. Constraints

In order to properly constrain the solver and generate a dynamically feasible trajectory, equality constraints were placed on the dynamics, the contact forces, the initial state, and the terminal state.

C. Dynamics Constraints

The dynamics constraints at each time step are represented by the following equation:

$$x_{n+1} = x_n + h(f(x_{n+1}, u_n, \lambda_n)) \tag{22}$$

where $f(x_{n+1}, u_n, \lambda_n)$ is the implicit Euler dynamics of the system.

D. Contact Constraints

To properly represent the different dynamic modes and apply the hybrid method to this problem, contact constraints must be in place. The contact constraints are implemented as follows:

$$\lambda_n = 0 \text{ (in flight phase)} \tag{23}$$

$$\dot{x} + \frac{l}{2}\cos(\theta_1)\dot{\theta}_1 = 0 \text{ (in stance phase)} \tag{24}$$

$$y = 0 \text{ (in stance phase)} \tag{25}$$

In stance phase, this results in the foot being pinned to the ground. In flight phase, this produces a zero contact force on the system.

E. Initial and Terminal Constraints

The initial and terminal constraints are represented by the following:

$$x_0 - x_{init} = 0$$
$$x_N - x_{end} = 0$$

F. Constraint Jacobians

The Jacobians are found using the ForwardDiff.jl package and provided to the solver. Attempts at using sparse arrays were made to speed up this process, but ultimately failed to produce the correct result and led to using dense Jacobians instead.

G. Solver

The problem is constructed using Julia's MathOptInterface, which in turn calls IPOPT to solve the constrained NLP. The solver was provided a constraint tolerance of 1e-1 and a primal and dual feasibility tolerance of 1e-2 with a maximum of 100 iterations.

IV. Results

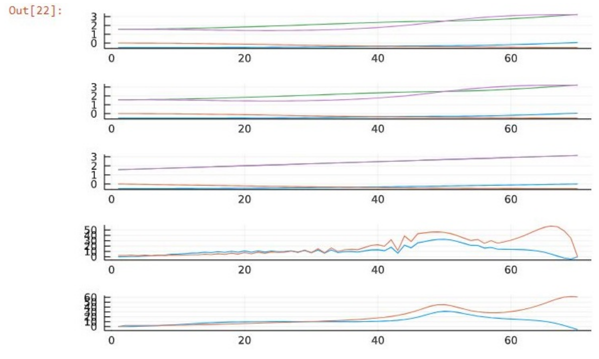
Each of the following tests were run at 100 Hz. All except for the pure dynamics test were given the initial state

$$x_{init} = \begin{bmatrix} -\frac{l}{2}\cos(\frac{\pi}{4}) \\ \frac{l}{2}\sin(\frac{\pi}{4}) \\ \frac{\pi}{4} \\ -\frac{\pi}{4} \end{bmatrix} \tag{26}$$

A. Pure Dynamics

The first test constrained the control inputs to be 0 and looked to see if the solver could mimic the integrated dynamics of the system. The solver is kept in stance phase for the entire 70 knot points. Starting from a stance phase with angles of $\theta_1 = \pi, \theta_2 = \frac{\pi}{2}$, the results are shown below:

Fig. 3: Dynamics Simulator vs Solver

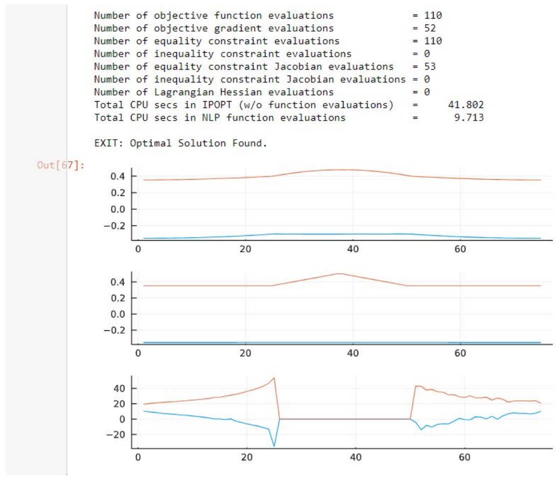


The optimized trajectory in the top plot almost perfectly matches the dynamics simulator in the second plot from the top. The solver's contact forces (second to bottom plot) don't exactly match the actual contact forces (bottom plot) but are similar enough to be reasonable. A possible explanation for this is the constraint tolerance, which is discussed in the conclusion.

B. Vertical Hop

The next test looked at a small vertical hop, causing the robot to jump into the air about 0.2 meters. This test contained 75 knot points with the first 25 assigned to a stance phase, the next 25 assigned to a flight phase, and the last 25 assigned to a stance phase. The orange line in the plots represents y height of link 1, and the blue line represents x location of link 1. The solver's trajectory is plotted on top, followed by the reference trajectory, and finally followed by the contact forces.

Fig. 4: Small Vertical Hop



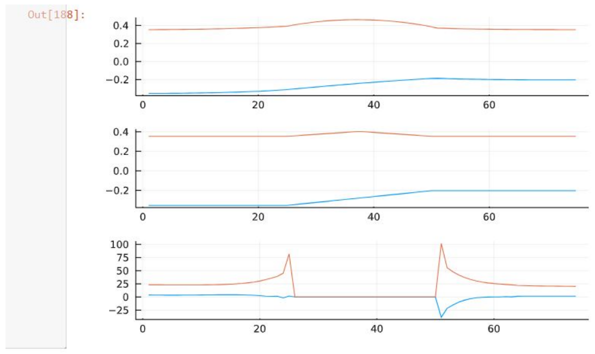
```
Number of objective function evaluations             = 110
Number of objective gradient evaluations             = 52
Number of equality constraint evaluations            = 110
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 53
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =     41.802
Total CPU secs in NLP function evaluations           =      9.713

EXIT: Optimal Solution Found.
```

The hopper takes a small vertical hop by actuating its links towards the vertical and pushing off the ground. It is not quite as high as the reference trajectory, but that is likely the physics of the system limiting the height of the hopper. The contact forces appear reasonable, with a sharp spike when taking off and landing to represent pushing off the ground.

C. Forward Hop

The next test produced a small forward hop. The reference trajectory allowed the robot to jump 0.15 meters forward and 0.1 meters upwards, with 75 knot points being distributed in sets of 25 to a stance-flight-stance pattern again. Once again, the orange line represents y and the blue line represents x. The solver's trajectory is plotted on top, followed by the reference trajectory, and finally followed by the contact forces.

Fig. 5: Forward Hop



The robot is able to stick the landing almost perfectly this run, resulting in a forward hop very close to the reference trajectory. The blue spike at the beginning of the second stance phase in the contact forces indicates that the robot is pushing down to stop its forward momentum.

D. Flips

The final tests were able to produce a front flip and a back flip. Each of these flips were directly vertical and leapt upwards an additional 0.375 meters before landing on the foot. During the flip, an angular velocity was applied to both links which allowed them to complete a single rotation while in flight. 100 knot points were assigned, with 50 assigned to the starting stance phase and the remaining 50 assigned to the flight phase. For the top plot, the middle plot, and the bottom plot in each figure, the orange line represents y and the blue line represents x. For the other two plots, the orange line represents θ_2 and the blue line represents θ_1, as these graphs plotted angles instead. For figures 6 and 7, the plots are ordered as follows: the solver's trajectory is on top, followed by the solver's angles, followed by the reference trajectory, then the reference angles, and finally the contact forces.

The flip results were excellent. As expected, the contact forces were similar to the short hop, with a small force in the x direction upon landing and a large force to stop the falling motion. Moreover, the solver was able to complete both flips and land successfully on the foot.

Fig. 6: Front Flip

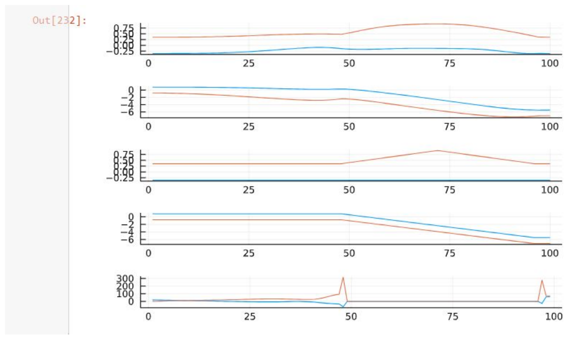
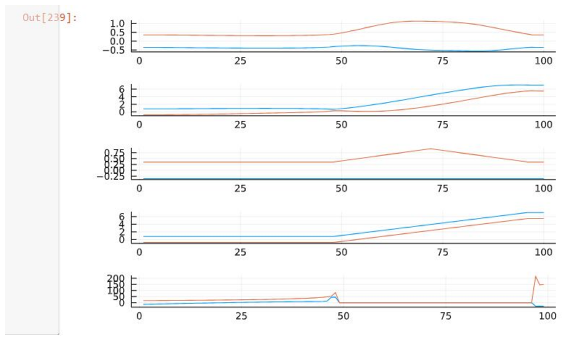


Fig. 7: Back Flip



V. Conclusion

Based on the results from the provided tests, the simulation, modelling, and trajectory optimization of a planar RR robotic leg was a success. For short amounts of time, IPOPT is able to solve several fairly common hopping motions to a reasonable degree. This allows the robot to model complex movements such as flips and forward leaps — movements with the potential to be combined with different gaits for high levels of mobility. Future work on this project would including speeding up the Jacobian with sparse matrices, as the naturally sparse structure of the Jacobians would allow IPOPT to solve longer trajectories in a reasonable amount of time. Another avenue for refinement is tolerance tuning. Due to the constraint tolerances, there were times when the contact forces oscillated wildly or simply didn't match the dynamics simulator; adjusting their tolerances might result in a more stable and quick-to-converge optimizer. Finally, refining the cost function would allow the solver to converge in fewer iterations. Combining these to greatly speed up the trajectory optimization and allow a real-life robot to perform the algorithm online would be a potential application of this work.

Acknowledgment

I'd like to thank Professor Manchester for helping me through this project. I'd also like to thank my computer for not giving up when solving large NLPs.