6.8210 Final Project - Sk8t3r Bot

Howard Beck

***Abstract*—This paper seeks to develop a trajectory optimization algorithm for an ice skating. A lot of work has been done into walking or running robots, but skating is a completely different control domain. While ice skating, one is able to move with astonishing freedom without even picking up a foot. Further, with enough precision, one is able to move along the ice only with one leg - known as "power pulls." Here, we seek to show there is a feasible construction of a robot and a trajectory optimization algorithm to create a cyclic motion resulting in a one-legged skating robot moving forward.**

I. Introduction

Ice skating has a lot of differences compared to more well-studied control domains such as walking. Skating typically involves pushing a metal blade on the ice in order to induce a reaction force that pushes you forward. As such, one of the most important things is keeping track of the angle of the blade as it compares to the direction of travel. The blade is also long and sharp, meaning there is a lot more friction in the direction perpendicular to the blade than parallel to it.

An ice skating robot could do a lot more than a walking robot would be able to do because of the low friction environment. Human skaters are able to spin rapidly in place, effortlessly switch between going forwards and backwards, and despite being able to reach high speeds, are able to stop on a dime using the correct technique. Making an ice skating robot is therefore a great and flashy way to both test the limits of our control algorithms.

Here, we develop a simulation of a one-legged ice skating robot. The choice of having one leg versus two was done purely for computational reasons - having an extra leg adds more degrees of freedom, and skilled skaters can carefully alternate between moving the blade of one foot to go forwards, all while only being on one leg. The simulation is designed to be cyclic - the starting and ending states are nearly identical except for the elapsed distance in the travel direction. Therefore, it would be possible to traverse an arbitrarily large distance using this cycle. The robot's physical design was done using a URDF, and the simulation and optimization was done using a Python implementation of Drake.

II. Robot Model

A. Contact Forces

As a skater moves along the ice, the high pressure resulting from a sharp blade melts the top layer of the ice. This interaction results in a friction force applied to the blade. In this simulation, the friction force was modelled as being two separate friction forces, which acted on the directions parallel to the blade and along the short end of the blade.

As the length of the blade cuts through ice, it is exposed to a thin slice at the front, resulting in a low friction. However, if the blade is slid along the ice perpendicular to its long slice, there is a larger exposed slice, resulting in higher friction. Since the friction primarily comes from water melting, there should be more friction as the velocity increases, as more ice is being melted producing, pushing the blade more.

As such, the friction forces were modelled as viscous friction counteracting the velocity component along the respective direction. This is made explicit in the following equations, where R is a rotation matrix mapping the standard frame of the blade (+X is the short side, +Y is the long side, towards the front of the skater) into the world frame. We use the following:

$$\hat{n} = R\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\hat{b}_f = R\begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$\hat{b}_s = R\begin{bmatrix}1\\0\\0\end{bmatrix}$$

where $\hat{n}$ represents the direction from the blade towards the leg, $\hat{b}_f$ represents the direction towards the front of the blade, and $\hat{b}_s$ represents the direction from the side of the blade.

Then, if the blade was moving at a velocity v, this resulted in the following friction forces:

$$\vec{F}_f = -k_f(v \cdot \hat{b}_f)\hat{b}_f$$

$$\vec{F}_s = -k_s(v \cdot \hat{b}_s)\hat{b}_s$$

with k_f, k_s being the viscous friction coefficients for forwards and sideways movement. In the simulation, k_f was set to 10, and k_s was set to 0.05.

While it is possible to set the normal force $\vec{F}_n$ in the $\hat{n}$ direction explicitly, it was instead left as an optimization variable whose job was to ensure $R\hat{n}$ had no z component. That is, there would be a force balance:

$$mg + \vec{F}_a \cdot \hat{z} + \vec{F}_n \cdot \hat{z} = 0$$

with $\vec{F}_a$ being the applied force on the blade from actuators. The force balance constraint ensured the blade would be pushed by the ground as much as needed to not be enter the ground.

Note the importance of having a high perpendicular friction along the blade. The way skating forward on one leg works is by moving the blade in a sinusoidal pattern across the ice. Without a high friction, the velocity would take a long time to change. In that scenario, it would take a lot more actuation in order to not have high amplitudes of oscillation. Having a high friction means it is feasible to go forward with only a small amount of side-to-side motion.

B. Robot Design

The robot was made of 4 sections - a body, upper leg, lower leg, and a blade. The rotation and position of the body were taken to be at the central part of the blade's contact with the ice, so that calculating the contact forces was easy.

The robot had an actuator at the *ANKLE* joint meant to keep the blade parallel to the ice at all times, along with a *KNEE* joint. The knee joint is important for skating, as it allows the skater to push on the ice while keeping the center of mass of their body above their blade.

The upper leg was given 3 actuators, amounting to a yaw pitch and roll at the hip joint. One actuator moved the leg to the side of the robot, as would be done by a hip extensor muscle, and was thus called the *HIP* joint. Another one acted as a quadriceps/hamstring muscle, controlling the pitch of the upper leg, and was thus called the *QUAD*. The final actuator controlled the rotation of the leg along its axis, as would be done by hip rotation, and was thus called *INNER_HIP*.

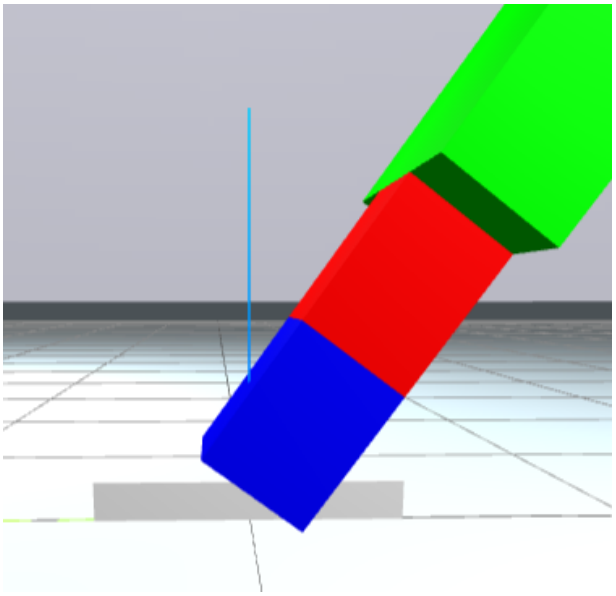

Fig. 1. Non-zero angle at ANKLE. Note that the robot's rotation frame is at the bottom of the blade, which is why the whole robot rotates with ankle rotation.

In order to give the upper leg all 3 degrees of rotational freedom, the upper leg was connected to an invisible, massless, link actuated by the QUAD. Another invisible, massless, link was attached to this one, and the joint was controlled by the HIP. Finally, this was connected to the body via the INNER_HIP.

Implementing the contact forces ended up being difficult. Using the standard implementation of friction via collision geometries was not feasible, as collisions did not produce differentiable dynamics to use for trajectory optimization. As

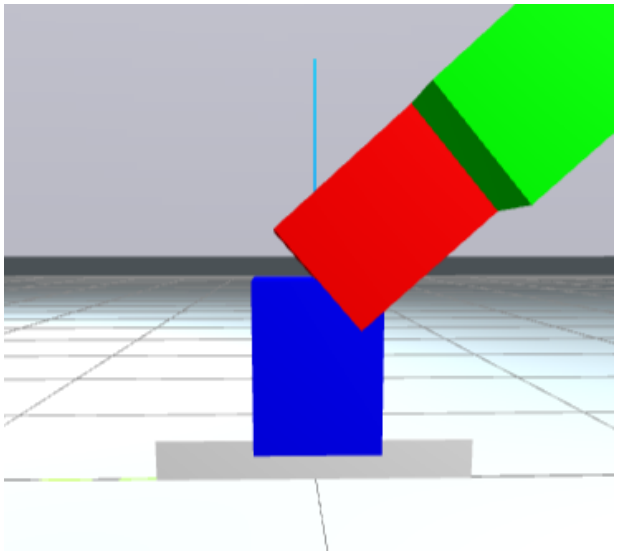

Fig. 2. Non-zero flexion at KNEE

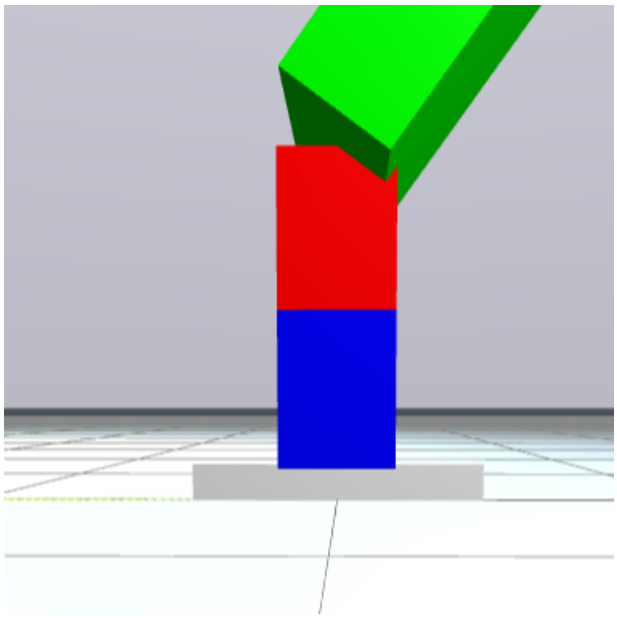

Fig. 3. Upper leg pitch rotation at QUAD

such, the contact forces were modelled using 3 propellers[1] which produced an arbitrary thrust in the given directions. These thrusts were specified as constraint variables in the trajectory optimization.

C. Physical Parameters

Each component was assumed to be a box with a uniform density, and their inertia tensors were derived as such. Below is a table with the masses and dimension of each component:

[1]Thank you to my friend Mohammed Ehab, who took the class last spring, for the idea.

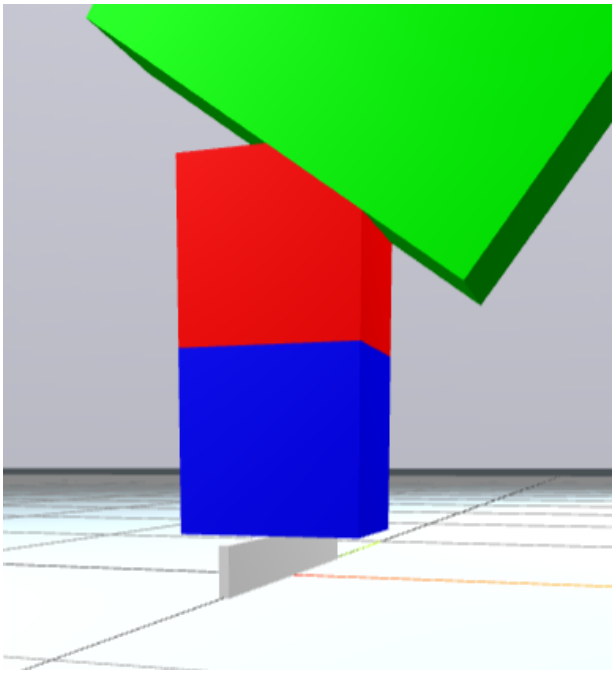

Fig. 4. Upper leg roll rotation at HIP

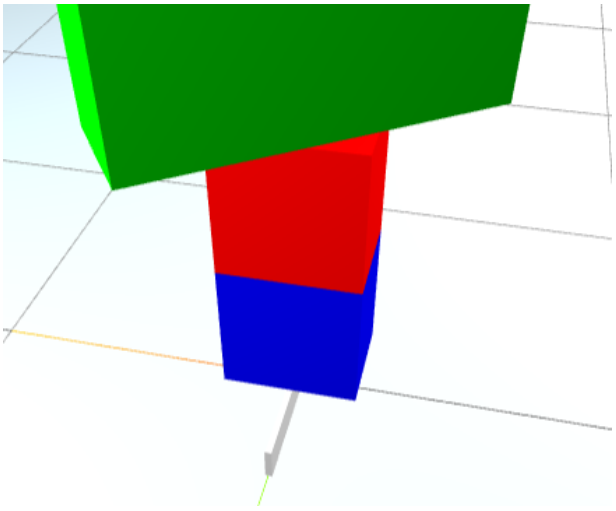

Fig. 5. Upper leg yaw rotation at INNER_HIP

	mass	height ($+z$)	width ($+x$)	depth ($+y$)
body	50	0.5	0.5	0.15
upper leg	10	0.2	0.2	0.15
lower leg	10	0.2	0.2	0.15
blade	2	0.05	0.01	0.4

TABLE I
PHYSICAL PARAMETERS OF THE SYSTEM

III. TRAJECTORY OPTIMIZATION

The state vector $\vec{q}$ consisted of 23 states. The first 4 were the quaternionic rotation of the blade, followed by x, y, z, ANKLE flexion, KNEE flexion, the QUAD angle, HIP angle, and INNER_HIP angle. The next 3 states modelled the time derivative of the quaternionic rotation, and the next 8 were time derivatives of x through the INNER_HIP angle.

The robot was optimized to go as fast as possible in the y direction, which was achieved by a running cost:

$$L = -\int_0^T v_y^2 dt$$

There was no additional control cost.

It was constrained to the initial position at the origin:

$$x(0) = 0$$

$$y(0) = 0$$

Further, apart from y, the state q was constrained to be the same at the start and finish. The state without a q component in the following is represented by $q_y^\perp$, the vector in $\mathbb{R}^{23}$ that is the projection of q to the complement of the y axis.

$$q_y^\perp(0) = q_y^\perp(T)$$

This constrained allowed the trajectory to be cyclic - at the end of a cycle, the robot would've just moved forwards.

The following two constraints enforced that the normal force from the ice to the blade, $\hat{F}_n$, was correct:

$$(R\hat{n}) \cdot \hat{z} = 0$$

$$z = 0$$

These constraints are redundant, but numerical computation was more reliable with both. The rotation matrix R was computed using the quaternionic state.

Then, there were the friction forces, which were difficult to add in the simulation. To enforce they were correct, they were added as a constraint:

$$F_f = -0.05(v \cdot \vec{b}_s)\vec{b}_f$$

$$F_s = -10(v \cdot \vec{b}_s)\vec{b}_s$$

The following constraints were added for balance:

$$\hat{b}_f \cdot \hat{y} \geq \cos(45\,\text{deg})$$

$$\hat{n} \cdot \hat{z} \geq \cos(45\,\text{deg})$$

The first one ensures the robot mostly stays on track, limiting the blade angle to be within 45 degrees of the y axis. The second constraint limits the blade normal to be within 45 degrees of the z axis, so that the robot is encouraged to stay upright.

To ensure the robot did not deviate too much from the y axis, the following constraints were also added:

$$-1.5 \leq x \leq 1.5$$

Finally, there were physical constraints added on the knee angle k, quad angle ϕ, and hip angle h:

$$0 \leq k \leq 50\,\text{deg}$$

$$-15\,\text{deg} \leq \phi \leq 0$$

$$-15\,\text{deg} \leq h \leq 15\,\text{deg}$$

These ensured that the knee joint didn't bend into the body, and that there were reasonable limits on what the joints could do.

The trajectory was optimized using a Direct Collocation program provided by Drake, with 21 state samples. The samples were spaced out between 0.05 to 1 seconds apart, with equal spacing. Drake's optimization determined the optimal time between samples. The starting trajectory was started with an identity quaternionic rotation state, with all states being zero except for $y = 0.5t$, $v_y = 0.5$.

IV. Results

The trajectory-optimized-robot did incredibly well at replicating what skaters do. It started with the blade at an angle in order to achieve a slight forward push. Then, it curved the blade angle in an alternating fashion between each edge. Both of the times it switched the edge of the blade closest to the ice, it would also bend the knee joint. Further, among the entire trajectory, the body component stayed aligned with the z axis. This almost made it into the code as a required constraint, but wasn't implemented after the robot did it anyways. Interestingly, it did bend the knee 3 times in the duration of one cycle, instead of the expected 2. Because of the three bends, it is difficult to decide whether the robot was following the common wisdom to bend the knee before changing the angle of the blade. The generated cycle lasted about 1.6 seconds, with the average y velocity being about 0.6 meters per second.

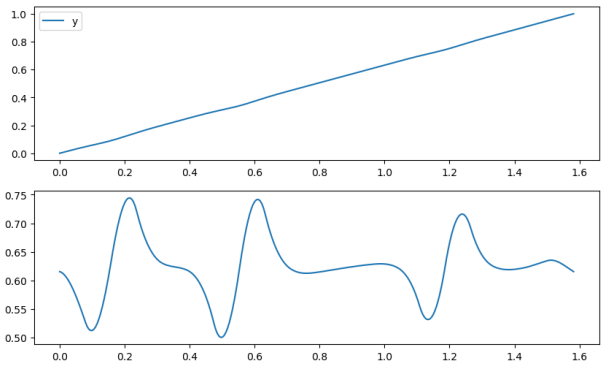

Fig. 6. y position and velocity of the robot. It hovered at around 0.6 meters per second

An animation of the cycle can be found at https://github.com/howard-beck/6-8210-Sk8t3r-Bot/animation.gif. The entire code can be found in the same GitHub repository: https://github.com/howard-beck/6-8210-Sk8t3r-Bot

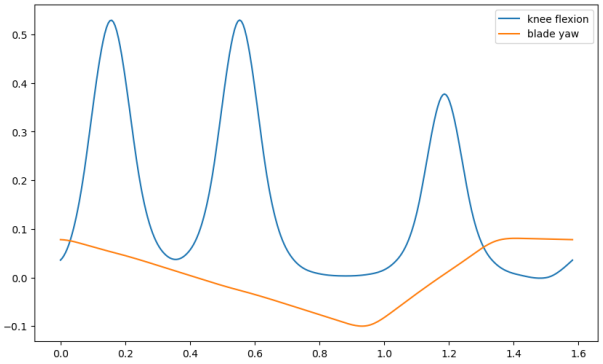

Fig. 7. Comparison of the knee flexion and blade yaw over time.

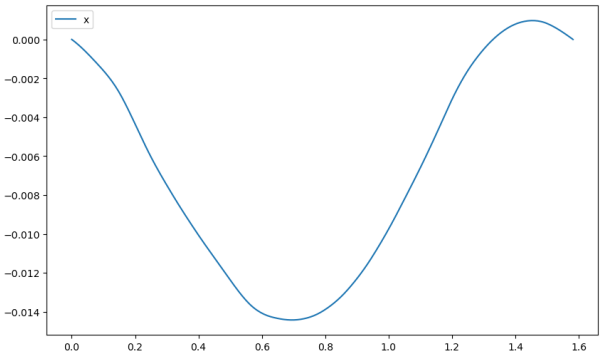

Fig. 8. x position of the blade over time. As predicted, it follows a sinusoidal-like trajectory so that the blade can keep pushing forwards.

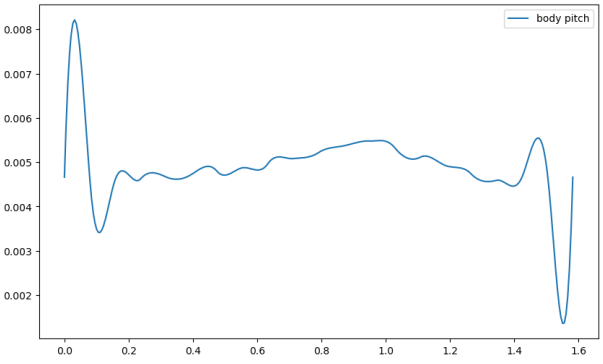

Fig. 9. Pitch of the body over time. The body stays remarkably upright, without having a cost or constraint incentive to do so.

V. Next Steps

There are a variety of places where this work can go from here. The most obvious one is to integrate the trajectory optimization into a control algorithm. The natural of direct collocation means the dynamics are optimized with less accuracy than they are simulated, so it would be good to make sure the trajectory is controllable.

Further, there are a lot of ways to manipulate the one-legged skating robot. One particularly hard skill is to skate

and subsequently stop with one foot, without tripping, and it would be interesting to see if the robot can do so. Other things it could potentially do are to turn - by pushing the leg outwards, applying enough force quickly enough, and carefully retracting the leg, the robot should be able to perform a figure skating-like spin. Arm joints could be added to make this look better, since bringing in the arms decreases the moment of inertia of the skater and then causes faster spinning.