Trajectory Generation for a Linearly Actuated Delta Quadruped Robot

Jennifer Yang
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA
jennifey@andrew.cmu.edu

Alfred Chang
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA
alfredc@andrew.cmu.edu

Jon Yeo
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA
jongwony@andrew.cmu.edu

***Abstract*—This paper introduces Delta Walker, a soft quadruped robot employing four 3D-printed prismatic delta robots. Initially designed as a hand-like manipulator, the Delta Walker operates within a diamond-shaped arrangement about its origin and exhibits compliant behavior due to the presence of soft links. The primary focus of this work lies in utilizing nonlinear trajectory optimization to generate walking trajectories for Delta Walker, which are subsequently verified through simulation using the inverse kinematics of the delta robots. The outcomes of this research contribute to advancements in control strategies for walking quadruped robots, particularly in trajectory optimization and simulation verification using inverse kinematics.**

***Index Terms*—prismatic delta robot, DIRCOL, trajectory optimization**

I. Introduction

Delta Walker, Fig. 1, is a quadruped robot that uses 4 soft, 3D printed prismatic delta robots arranged in a diamond shape about the origin of the robot. It is approximately 15 cm $\times$ 15 cm wide and ranges from approximately 15 cm to 17 cm tall, depending on the motor actuation amounts. The robot initially operates as a hand-like manipulator, where the delta robots are configured as fingers to simulate dexterous hands with parallel fingertips. The parallelism and translational motions of these delta robots facilitate fingertip control, and the presence of soft links adds a degree of compliance [1]. In this paper, we present a method for trajectory optimization for the **Delta Walker**. We generate multiple trajectories that can be repeated for many iterations on robot to create full walking gaits. We ran these trajectories in simulation to verify that they would not cause the robot to fall over.

II. Related Works

The field of walking quadruped robots has seen significant advancements in control strategies aimed at achieving agile and robust locomotion. Various approaches have been explored to address the complexities of controlling multi-legged systems in diverse environments.

One notable focus in recent research has been on reinforcement learning (RL) algorithms for controlling quadruped robots. RL offers the advantage of learning controllers without explicit modeling of system dynamics, making it suitable for handling noisy observations and controls. Platforms like RealAnt have been developed specifically for real-world reinforcement learning research, enabling the validation and evaluation of state-of-the-art RL algorithms on physical platforms [2].

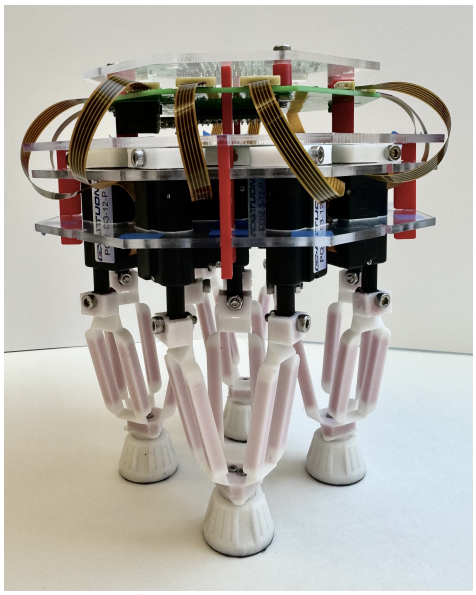

Fig. 1: **Delta Walker**

Optimization-based control strategies have also played a crucial role in achieving robust locomotion for quadruped robots. These strategies include predictive controllers that reason about consequences iteratively and reactive controllers that focus on immediate actions. Model-based approaches offer advantages in safety considerations and generalization but face challenges in integrating perceptual streams [3].

Efforts have also been made to enhance the mechanical and electrical components of quadruped robots. Designs such as HAMR-F leverage advances in manufacturing, sensing, and energy storage to improve mechanical robustness, variable gait control, and payload capacity. These advancements contribute to achieving faster locomotion, improved heading control, and dynamic gaits with aerial phases [4].

Additionally, the development of low-cost and scalable

This work was funded by the NSF under project Grant No.CMMI-2024794.

quadruped robot platforms has opened avenues for research and education in control algorithms. Platforms like MIT mini-cheetah, Solo, and Stanford Doggo offer affordable options for exploring motion planning controllers and reinforcement learning in real-world settings [5]–[7].

The integration of advanced control algorithms with innovative hardware designs has paved the way for tackling challenges such as terrain adaptability, speed limitations, and payload capacity in walking quadruped robots. Ongoing research continues to focus on improving sample efficiency, safe exploration, and real-time adaptation of control strategies for agile and versatile locomotion in various environments.

Overall, the interdisciplinary efforts in controls for walking quadruped robots encompass a wide range of techniques, from reinforcement learning and optimization-based strategies to mechanical enhancements and low-cost platform development, driving advancements in robotic locomotion and autonomy.

III. Trajectory Optimization

A. IPOPT

We chose the Interior Point OPTimizer solver – also termed IPOPT – as our solver of choice. This solver is meant for large scale non-linear optimization problems of continuous systems. We chose this solver because it fit with our problem setup of a quadratic cost function with a quadratic constraint. It is worth noting that all but the body torque constraint were linear constraints. The body torque constraint as specified in the Constraints section later is a function of both relative feet positions and the ground contact forces. Future work could target methods for reframing this constraint as a linear constraint to speed up computation.

B. Dynamics

Our system is represented by five point masses: one for the body and one for each foot. The state comprises the positions and velocities in the x, y, and z axes of these masses, resulting in a total of 12 degrees of freedom and 30 states. The system is governed by four control inputs, corresponding to forces along each leg. The ordering of the state and control vectors is as follows:

$$x = \begin{bmatrix} p^{(b)} \\ p^{(1)} \\ p^{(2)} \\ p^{(3)} \\ p^{(4)} \\ v^{(b)} \\ v^{(1)} \\ v^{(2)} \\ v^{(3)} \\ v^{(4)} \end{bmatrix} \qquad u = \begin{bmatrix} F^{(1)} \\ F^{(2)} \\ F^{(3)} \\ F^{(4)} \end{bmatrix} \tag{1}$$

where $p^{(i)}$ is the position of the body or foot i in each axis, $v^{(i)}$ is the velocity of the body or foot i in each axis, and $F^{(i)}$ is the force along leg i.

We formulated our system dynamics as a linear equation governing forces:

$$\begin{bmatrix} m_b I & & & & & \\ & m_f I & & & & \\ & & m_f I & & & \\ & & & m_f I & & \\ & & & & m_f I & \end{bmatrix} \begin{bmatrix} \dot{v_b} \\ \dot{v_{f_1}} \\ \dot{v_{f_2}} \\ \dot{v_{f_3}} \\ \dot{v_{f_4}} \end{bmatrix} = \begin{bmatrix} -m\ddot{_b}g \\ -m\ddot{_{f_1}}g \\ -m\ddot{_{f_2}}g \\ -m\ddot{_{f_3}}g \\ -m\ddot{_{f_4}}g \end{bmatrix}$$
$$+ \begin{bmatrix} -I & -I & -I & -I \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} \tag{2}$$

where m_b is the mass of the body, m_f is the mass of each foot, and λ_i is the contact force on foot i.

C. Cost Function

We experimented with two different cost functions to evaluate their performance and found that Eqn. 4 yielded significantly better results.

$$J(x_{1:N}, u_{1:N-1}) = \sum_{i=1}^{N-1} \left[\frac{1}{2} ||x_i - x_g||_2^2 + \frac{1}{2} ||u_i||_2^2 \right] + \frac{1}{2} ||x_N - x_g||_2^2 \tag{3}$$

The first cost function comprises a cost term for the distance between the current state and the goal state, along with a cost term for the internal body-to-foot forces. The state error cost term encourages the solver to reach the goal state quickly, as earlier attainment results in reduced net state error. The internal force cost term promotes lower overall control costs to minimize control effort and the forces sustained by the actuators. However, this cost function required the solver to take 2.5 times longer to generate a trajectory with minimal error compared to the subsequent cost function.

$$J(x_{1:N}) = \sum_{i=1}^{N} \frac{1}{2} ||x_i - x_g||_2^2 \tag{4}$$

This cost function penalizes solely the error between the current state and the goal state, driving the solver to find a swift solution to reach the goal state without considering penalties for the control forces within the body. Utilizing this cost function, the solver produced the same trajectory, within minor tolerance, 2.5 times faster than Eqn. 3. Consequently, we employed this cost function for our optimization problem.

D. Optimization Problem & Constraints

In our problem setup, we employ a predefined set of knot points where one foot is permitted to be lifted off the ground while the other feet remain fixed to the ground. These knot points are stored in vectors named M0, M1, M2, M3, and M4, where the number designates which foot is permitted to be lifted off the ground, with M0 indicating that all four feet

must maintain contact with the ground. The inclusion of M0 allows the robot sufficient time to shift its weight into the support polygon in preparation for the subsequent step.

The constraints applied to the optimization problem are as follows:

$$\min_{x_{1:N}} \quad J(x_{1:N})$$

1) Initial and goal state constraint:

$$\text{st} \quad x_1 = x_{ic} \tag{1}$$

$$x_N = x_g \tag{2}$$

Initial and goal constraints are used to dictate the starting and end states that the solver will optimize the trajectory for.

2) Dynamics constraint:

$$x_{k+1} = \text{dynamics}(x_k, u_k, \lambda_k) \quad \text{for } k \in [1, N-1] \tag{3}$$

To guarantee adherence to the modeled dynamics during simulation, we impose an equality constraint on each subsequent state, derived from the dynamics solution of the preceding state.

3) Body height constraint:

$$0.053 \leq p_z^{(b)} \leq 0.0605 \quad \text{for body z-height, } k \in [1, N] \tag{4}$$

To prevent the body from falling to the ground or flying into the sky, we apply a primal constraint on the z-height of the body position. The maximum z-height prevents scenarios where the robot flies into the sky and all feet lift off the ground. Conversely, the minimum z-height prevents the body from sinking too close to the ground, hindering the legs' ability to lift. We set the maximum range of motion from 4.8cm to 6.8cm in height, and constrain the z-height of the robot body between 5.3cm and 6.05cm. This buffer ensures that the robot remains in a state where the legs can be lifted off the ground.

4) Foot height constraint:

$$p_z^{(i)} \geq 0 \quad \text{for i=1,2,3,4, } k \in [1, N] \tag{5}$$

To prevent the foot from falling into the ground, we place a primal constraint on the z-height of the foot position.

5) Foot speed inequality constraint:

$$||v^{(i)}||_2 \leq 0.015 \quad \text{for i=1,2,3,4, } k \in [1, N] \tag{6}$$

Based on the physical specifications of the delta's linear actuators, we impose a maximum speed of 15 mm/s on each foot.

6) Grounded foot velocity constraint:

$$||v^{(i)}||_2 = 0 \quad \text{for grounded foot, } k \in [1, N] \tag{7}$$

To prevent the solver from modeling frictionless contact, we apply a primal bound zero-velocity constraint on the feet that are determined to be fixed to the ground. It is reasonable to implement a zero-velocity constraint because the contact forces realizable by the motors are small compared to the body mass. This assumption also simplifies computation by eliminating the need to model a friction cone constraint.

7) Grounded foot z-height constraint:

$$p_z^{(i)} \leq 0.00025 \quad \text{for the grounded feet, } k \in [1, N] \tag{8}$$

To further ensure that the robot never enters a dynamically unstable mode by lifting more than one foot, we enforce primal bound z-height constraint on the three feet that are meant to be grounded as dictated by the M's. Without this constraint, we noticed that the solver would create a trajectory where multiple feet are lifted.

8) Ground contact force constraint:

$$\lambda_z^{(i)} \geq 0 \quad \text{for i=1,2,3,4, } k \in [1, N-1] \tag{9}$$

Since ground forces can only apply a normal force to an object in contact and cannot apply any force when an object is not in contact, we apply a constraint where the ground forces are non-negative if a foot is in contact with the ground, and a constraint where the ground force is zero if a foot is in the air.

9) Floating foot contact force constraint:

$$\lambda_z^{(i)} = 0 \quad \text{for lifted feet } k \in [1, N-1] \tag{10}$$

Since the ground cannot exert a normal force on a foot that it is not in contact with, we place a primal bound on the lifted feet to ensure that the force is zero.

10) Body torque equality constraint:

$$\sum_{i=1}^{4} \lambda^{(i)} \times (p^{(i)} - p^{(b)}) = 0 \quad \text{for } k \in [1, N-1] \tag{11}$$

To prevent the body from reaching a state where it could tip over, we apply a net-zero torque constraint around the body center of mass (COM). This ensures that the body remains within the support polygon and never tips over. It's worth noting that this constraint is the only quadratic constraint in our optimization problem. All other constraints in our optimization problem are linear constraints.

11) Foot position inequality constraints:

$$\|p_{x,y}^{(a_i)} - p_{x,y}^{(i)}\|_2 \leq 0.0135 \tag{12}$$

$$p_z^{(i)} \geq -0.0025 + (p_z^{(a_i)} - 0.058) \tag{13}$$

$$p_z^{(i)} \leq 0.0075 + (p_z^{(a_i)} - 0.058) \tag{14}$$

$$\text{for foot } i, \text{ for } k \in [1, N]$$

The Foot Position Inequality Constraints take the shape of a cylinder, with the circle in the x, y-plane, that moves in the global frame fixed relative to $p^{(a_i)}$ for each foot i. The $(p_z^{(a_i)} - 0.058)$ serves as a z-offset for the lower and upper z bounds of the cylinder to translate with, as $p^{(a_i)}$ is initially 0.058 meters above the ground. The cylindrical bounds are determined based on the workspace of the delta actuator's end effector, visualized as the space between two upward-facing convex paraboloids.

This cylindrical constraint restricts vertical motion of the body along the z axis in tandem with the other constraints.

As the cylinder moves in tandem with $p^{(a_i)}$ (and consequently with p^b), exceeding a 0.0025-meter lift would elevate the cylinder above ground level, potentially conflicting with the Grounded Foot Z-Height Constraint. Conversely, lowering the body by more than 0.0075 meters would submerge the cylinder entirely below ground, rendering no feasible $p^{(i)}$ according to the Foot Height Constraint.

IV. Simulation

The trajectory positions generated by the solver are in the world frame, where the ground plane is set to be $z = 0$. However, to calculate the motor actuation amounts to control the **Delta Walker**, these positions were shifted to the robot frame. For this, the $z = 0$ plane was shifted up to be coincident with the base plane of the robot, and the feet were placed on the negative z-axis. This adjustment was made to simplify the kinematic calculations. The motor actuation amounts for each delta were calculated using inverse kinematics for delta robots, considering the desired position of the COM and feet at each timestep. These actuation amounts were then input into the simulation environment.

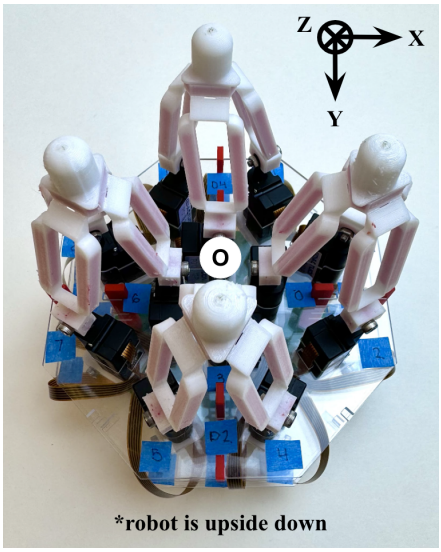

Fig. 2: Reference frame for the **Delta Walker**

The simulation of the **Delta Walker** was developed in Pybullet [8] using a simplified URDF model of the robot, as shown in Fig. 3, and set in the world frame. The objective of the simulation was to verify that the generated trajectories were quasi-statically stable and that the robot could reach the goal point. To simulate the motor actuation commands, we utilized Pybullet's built-in P controller. We inputted the desired motor actuation amounts and imposed limits on the maximum force and velocity of the actuators to ensure realistic movement. Although the real robot employs a PID controller to command the motors, the built-in P controller sufficed for simulation purposes. Given that the trajectories should be quasi-statically stable and the robot moves slowly in the real world, we did not prioritize optimizing the simulated robot speed. We were able to track the world frame positions of the COM and the feet of the robot in simulation and compared them to the desired positions of the robot from the generated trajectory.

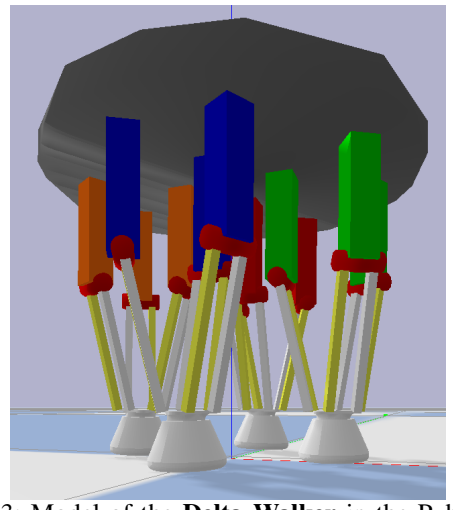
Fig. 3: Model of the **Delta Walker** in the Pybullet

V. Results and Discussion

We generated three different trajectories with varying goal states and knot points. In the simulation, we tracked the positions of each point mass and calculated the mean square error of their positions. These variations and their resulting errors are listed in Table I. The feet were stepped in the order of Foot 3, Foot 2, Foot 4, and Foot 1.

TABLE I: Trajectory Generation Results

Trajectory Info			**Mean Square Error** (mm)				
Label	Goal	Knot Points	E_b	E_1	E_2	E_3	E_4
A	(20, 20, 53)	40	8.0	7.7	7.7	7.0	7.0
B	(10, 10, 53)	40	1.8	1.6	1.9	1.8	1.6
C	(20, 20, 53)	80	3.0	3.8	4.6	3.0	3.1

Goal the goal position (x, y, z) of the body COM in mm.

For trajectories A and B, we opted to use 40 knot points, corresponding to each foot taking one step. For trajectory C, we chose to use 80 knot points, corresponding to each foot taking two steps. In trajectories A and C, we set the desired COM position to be 0.02 m away from the origin in the x and y axes. In trajectory B, we set the desired COM position to be closer to the origin, at 0.01 m away in the x and y axes. Trajectories B and C would take steps of size 0.01 m to reach the final point, whereas trajectory A would take a step of size 0.02 m.

Trajectory A, as shown in Fig. 4, exhibits the largest error. This occurred because the desired end point was too distant from the starting point, causing the actuators to reach their limits and the robot to take the maximum possible step. Additionally, the support polygon decreased in size after the first two steps, leading to tilting by the time the third foot (Foot 4) stepped in the sequence. Although it was meant to be

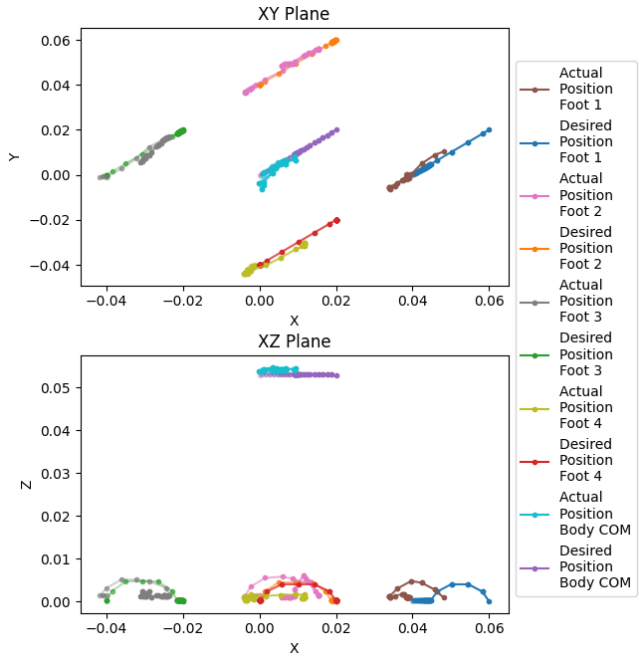

Fig. 4: Trajectory A Results

in the air, the robot leaned on that foot, resulting in instability and pushing rather than a stable step.

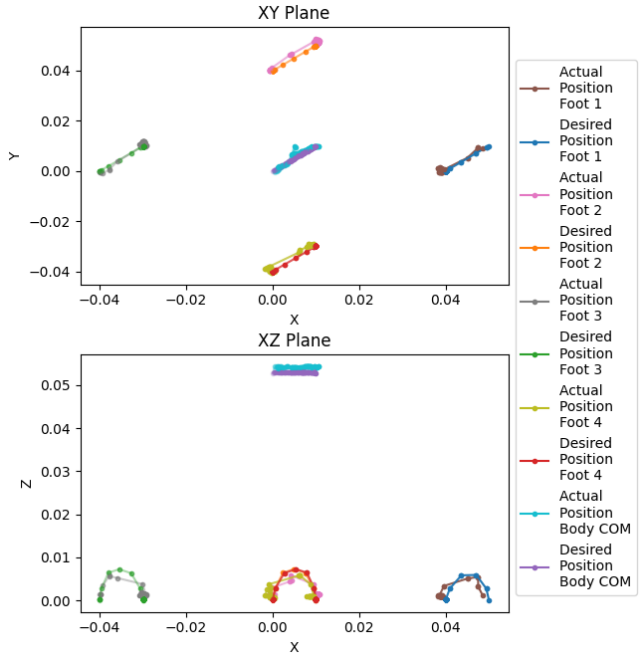

Fig. 5: Trajectory B Results

The error significantly improved with a smaller step size. Trajectory B, as depicted in Fig. 5, exhibits the smallest error among all trajectories. This improvement is attributed to the fact that the desired end point was within reach within the trajectory, ensuring that the actuator limits were not exceeded. Additionally, the support polygon remained wider, reducing the risk of the robot tipping over. Each foot lifted as expected, resulting in a stable step.

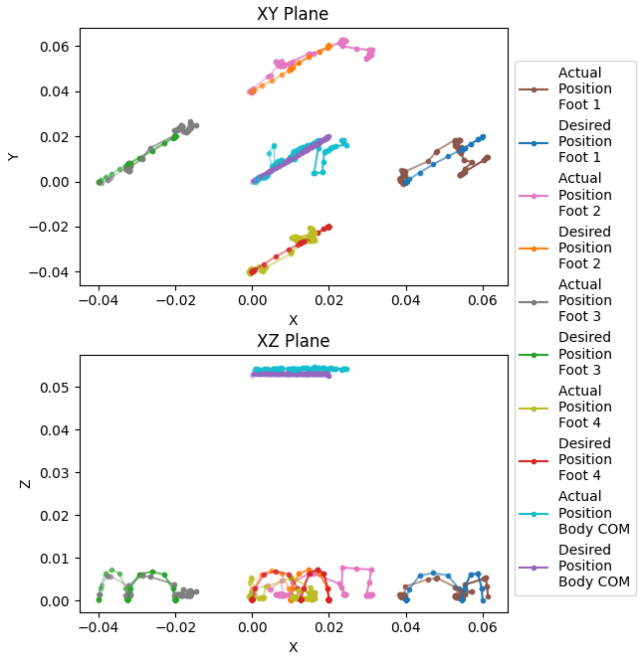

Fig. 6: Trajectory C Results

The error worsened when taking two steps despite the smaller step size. Trajectory C, illustrated in Fig. 6, exhibited better performance than Trajectory A but worse than Trajectory B. During the first set of steps, the second foot to lift (Foot 2) did not actually lift as the robot tilted towards it. Likewise, during the second set of steps, the third foot to lift (Foot 4) encountered the same issue. The steps were less stable because the generated trajectory had varying step sizes for each foot.

VI. Conclusions

We utilized trajectory optimization to generate three different trajectories for walking for the **Delta Walker**. We varied the goal states and knot points, allowing experimentation with the number of steps and step sizes. We found that using a smaller step size and generating only one set of steps was better for creating a quasi-statically stable gait. By repeating this gait multiple times, the robot has the potential to move far. Although taking larger steps or more sets of steps was not quasi-statically stable, they were still trajectories stable enough that the robot did not fall over, and the robot did progress forward towards the goal state. The gaits we generated and the parameters we experimented with will inform how to approach generating more gaits in the future.

VII. Future Work

One significant area of improvement is reducing solve time. Currently, the solver takes around 25 minutes to generate a trajectory with 40 knot points and 75 minutes to generate a trajectory with 80 knot points. Speed improvements can be found in a couple of areas. First, it may be possible to

reformulate the Body Torque constraint, which is present to prevent the body from tipping over, into a linear constraint form. This would help simplify the complexity of the problem and allow us to use a linear optimization solver. Second, there may be other nonlinear optimizers that perform better for our problem size. The main reason we chose IPOPT over other solvers was that we were most familiar with it.

Additional work can be done to make the robot walk faster. We can explore relaxations on the body height constraint and foot position constraints to allow the robot to actuate closer to its limits. Additionally, for a more accurate model, we could model the contact patches more accurately using the geometric shape of the robot feet as opposed to just the current point mass model. Furthermore, we can explore different foot orderings to find a better walking gait or discover other types of gaits.

Beyond simulation, we can also apply this to the real robot and run tests to quantify the movement of the robot. We can run multiple iterations of the same trajectory and observe how far the robot moves not just in one set of steps but in multiple sets of steps, or combine different trajectories to create more complicated paths.

Code and Videos

This research has not been published yet, so we cannot share the code at this time. However, videos of the trajectories in simulation, as well as some preliminary testing that has been done on the real robot, can be found here: https://tinyurl.com/ocrl-delta-walker.

Acknowledgment

This research is associated with CMU's Zoom Lab, led by Professor Zeynep Temel. The authors sincerely thank Professor Zeynep Temel, Professor Zachary Manchester, Arun Bishop, Zilin Si, and Sarvesh Patil for their assistance with this project.

References

[1] Z. Si, K. Zhang, O. Kroemer and F. Z. Temel, "DeltaHands: A Synergistic Dexterous Hand Framework Based on Delta Robots," in IEEE Robotics and Automation Letters, vol. 9, no. 2, pp. 1795-1802, Feb. 2024, doi: 10.1109/LRA.2024.3349920.

[2] R. Boney, et al., "RealAnt: An Open-Source Low-Cost Quadruped for Education and Research in Real-World Reinforcement Learning," 2022. [Online]. Available: https://arxiv.org/abs/2011.03085.

[3] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard and A. D. Prete, "Optimization-Based Control for Dynamic Legged Robots," in IEEE Transactions on Robotics, vol. 40, pp. 43-63, 2024, doi: 10.1109/TRO.2023.3324580.

[4] B. Goldberg et al., "Power and Control Autonomy for High-Speed Locomotion With an Insect-Scale Legged Robot," in IEEE Robotics and Automation Letters, vol. 3, no. 2, pp. 987-993, April 2018, doi: 10.1109/LRA.2018.2793355.

[5] B. Katz, J. D. Carlo and S. Kim, "Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 6295-6301, doi: 10.1109/ICRA.2019.8793865.

[6] Kim, D., Carlo, J.D., Katz, B., Bledt, G., & Kim, S. (2019). Highly Dynamic Quadruped Locomotion via Whole-Body Impulse Control and Model Predictive Control. ArXiv, abs/1909.06586.

[7] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger and R. Siegwart, "Control of dynamic gaits for a quadrupedal robot," 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 2013, pp. 3287-3292, doi: 10.1109/ICRA.2013.6631035.

[8] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.