Abstract—This paper presents a framework that unifies control theory and machine learning in the setting of bipedal locomotion. Traditionally, gaits are generated through trajectory optimization methods and then realized experimentally—a process that often requires extensive tuning due to differences between the models and hardware. In this work, the process of gait realization via hybrid zero dynamics (HZD) based optimization problems is formally combined with preference-based learning to systematically realize dynamically stable walking. Importantly, this learning approach does not require a carefully constructed reward function, but instead utilizes human pairwise preferences. The power of the proposed approach is demonstrated through two experiments on a planar biped AMBER-3M: the first with rigid point feet, and the second with induced model uncertainty through the addition of springs where the added compliance was not accounted for in the gait generation or in the controller. In both experiments, the framework achieves stable, robust, efficient, and natural walking in fewer than 50 iterations with no reliance on a simulation environment. These results demonstrate a promising step in the unification of control theory and learning.

I. INTRODUCTION

Despite advancements within robotics, realizing dynamic bipedal locomotion on hardware [1] remains a benchmark problem across the fields of control, engineering, high-performance computing and machine learning. The dynamics and control community has historically approached the challenge of walking from theory applied to real-world platforms, for example Raibert’s seminal work on hopping robots [2]. Such theory includes locomotion stability, which has been well studied and realized experimentally from various control perspectives including zero moment point (ZMP) [3] and simplified models, such as LIP [4], SLIP [5], and centroidal dynamics [6]. These methods, although powerful, do not account for the full-order dynamics of the system.

Alternatively, the hybrid zero dynamics (HZD) framework reduces the full-order dynamics to a lower-dimensional zero dynamics manifold, through which stability of the overall system can be certified. This is accomplished by characterizing walking with hybrid systems that encode state jumps and Lyapunov methods robust to these jumps [7]–[9]. This approach has been demonstrated for walking [10], running [11], and quadrupedal locomotion [12]. To achieve experimental success, however, one needs more than the theoretic stability guarantees enjoyed by the theory—one must achieve robustness against unmodeled dynamics, which is difficult to formulate formally in the context of attributes of nonlinear controllers. This “last-mile mission” was historically solved by intensive parameter tuning, an arduous and nonintuitive process which inevitably affects the scalability of translating theory to hardware in a practical setting.

To circumvent this engineering empiricism, the field of machine learning has approached bipedal locomotion from many perspectives, including reinforcement leaning and imitation learning. Reinforcement learning simplifies the process of “learning to walk” [13] without prior knowledge [14]–[17], but because this method relies on a carefully crafted reward function, the behavior is exclusively determined by its construction. This motivates the second method, imitation learning, which infers the underlying reward function from expert demonstrations [18]–[20]. While both methods have demonstrated promising results, they heavily rely on physical engines such as Bullet [21], MuJoCo [22], and RaiSim [23]. As realistic as these rigid-body-dynamics based simulation environments have become, they still struggle with rough-terrain dynamics such as elastic impacts, slipping contacts, and granular media. These differences become more apparent when transferred to real-world systems.

As opposed to relying on just one field, this work explores combining the successes of both: the formality of stability from control theory and the ability to learn the relationship between complex parameter combinations and their resulting locomotive behavior from machine learning. This is accomplished by building upon our previous results [24], [25] and...
systematically integrating preference-based learning with gait generation via HZD optimization. The result is optimal walking on hardware based only on relative pairwise preferences from the human operator (i.e. the user prefers gait A over gait B). We demonstrate the power of this framework through two experiments on a planar biped AMBER-3M portrayed in Fig. 1. In both experiments stable, robust, efficient, and natural walking is achieved in fewer than 50 iterations with no reliance on a simulation environment. Notably, this is the first experimental demonstration of compliant walking through learning.

II. HZD GAIT GENERATION

The underlying control scheme of the proposed learning framework is based around two concepts: (1) hybrid zero dynamics (HZD) [7], [8], which theoretically addresses locomotion stability, and (2) trajectory optimization, namely direct collocation [26], which produces a walking trajectory (gait) that encodes the stability of the closed-loop system. We will briefly review this methodology in this section.

A. Hybrid Zero Dynamics Method

Inherently, locomotion consists of alternating sequences of continuous-time dynamics and discrete-time impacts, which can be encoded as a hybrid control system [27]. Specifically, consider a robotic system of dimension $n$ with $q \in \mathbb{Q} \subset \mathbb{R}^n$ the configuration coordinates and $x = (q, \dot{q}) \in X \subset TQ$ the full system state. The continuous-time control system is given by

$$D(q)\ddot{q} + H(q, \dot{q}) = Bu$$

where $D(q) \in \mathbb{R}^{n \times n}$ is the inertia matrix, $H(q, \dot{q}) \in \mathbb{R}^n$ is the drift vector, $B \in \mathbb{R}^{n \times m}$ is the actuation matrix, and $u \in U \subset \mathbb{R}^m$ is the input. Here we present the “pinned” model for notional simplicity, but the “unpinned model” could similarly be considered [28]. Note that $m < n$ for underactuated robotic systems, such as the one under consideration in this paper.

As the robot’s foot strikes the ground, an instantaneous change in velocity occurs causing the system state to suddenly jump. Taking $z : Q \rightarrow \mathbb{R}$ to represent the height of the swing foot, the admissible states are given by the domain: $\mathcal{D} := \{(q, \dot{q}) \in \mathcal{X} : z(q) \geq 0\} \subset \mathcal{X}$. The region where this instantaneous change in velocity occurs is given by the switching surface $\mathcal{S} \subset \mathcal{D}$ defined by:

$$\mathcal{S} := \{(q, \dot{q}) \in \mathcal{X} : z(q) = 0, \dot{z}(q, \dot{q}) < 0\}.$$ (2)

Taking $x := (q, \dot{q})$, the discrete dynamics during this impact event are encoded by the reset map $\Delta : \mathcal{S} \rightarrow \mathcal{X}$, defined as:

$$x^+ = \Delta(x^-), \quad x^- \in \mathcal{S}$$

where the $x^+$ and $x^-$ denote the pre- and post-impact state respectively. Finally, one can convert (1) to a control system:

$$\dot{x} = f(x) + g(x)u$$

where when combined with (2) and (3) yields the single-domain hybrid control system:

$$\mathcal{HC} = \begin{cases} \dot{x} = f(x) + g(x)u & x \in \mathcal{D} \subset \mathcal{X} \\ x^+ = \Delta(x^-), & x^- \in \mathcal{S} \subset \mathcal{D} \end{cases}$$

which can be extended to the multi-domain case; for more details, refer to [7].

The HZD framework reduces the system $\mathcal{HC}$ to the lower-dimensional system. Consider the zero dynamics surface:

$$\mathcal{Z}_\alpha := \{x \in \mathcal{D} \mid y(q, \alpha) = 0, \dot{y}(q, \alpha) = 0\},$$

where $y : \mathcal{Q} \rightarrow \mathbb{R}^m$ is defined through the following outputs or virtual constraints (encoding desired behavior):

$$y(q, \alpha) = y^a(q) - y^d(\tau(q), \alpha).$$

Here, $y^a(q)$ is the actual measured output of the system, and $y^d(\tau(q), \alpha)$ is the desired output. For the following discussion, we take the desired output to be parameterized by the state-based timing variable $\tau(q)$ and a collection of Bézier coefficients $\alpha$. Through the use of a stabilizing controller $u^*(x)$, e.g., given by feedback linearization or control Lyapunov functions [8], [9], [27] one can drive $y \rightarrow 0$ exponentially. The end result is the closed-loop dynamics:

$$\dot{x} = f_\alpha(x) = f(x) + g(x)u^*(x).$$

In order to guarantee stability of a hybrid system, a hybrid invariance condition must be satisfied, encoded through the HZD condition [8]:

$$\Delta(S \cap Z_\alpha) \subset Z_\alpha.$$ (6)

The remaining step to achieving hybrid invariant walking is to generate an $\alpha$ that satisfies the HZD condition.

B. Trajectory Optimization

To obtain $\alpha$, we use a direct collocation-based optimization algorithm, FROST [26], which has been previously utilized for efficient gait generation of walking [29], running [11], and quadrupedal locomotion [30]. Direct collocation is an implicit Runge–Kutta method to approximate the numerical solution of certain dynamical systems, namely differential-algebraic equations and partial differential equations. The trajectory optimization problem is stated as

HZD Optimization:

$$\alpha^*, X^* = \arg\min_{\alpha, X} \Phi(X)$$

s.t. $\dot{x} = f_\alpha(x)$ (Closed-loop Dynamics) $\Delta(S \cap Z_\alpha) \subset Z_\alpha$ (HZD Condition) $X_{\min} \leq X \leq X_{\max}$ (Decision Variables) $c_{\min} \leq c(X) \leq c_{\max}$ (Physical Constraints) $a_{\min} \leq p(X) \leq a_{\max}$ (Essential Constraints)

where $X = (x_0, ..., x_N, T)$ is the collection of all decision variables with $x_i$ the state at the $i^{th}$ discretization and $T$ the duration. $\Phi(X)$ is the cost function, and $c(X)$ is the set of physical constraints on the optimization problem. These physical constraints are included in every gait generation framework to encode the physical laws of real-word, such as the friction cone condition, workspace limit, and motor capacity [29]. In this work, we specify a subset of these constraints, $p(X)$, termed in this work as essential constraints, which are discussed further in Sec. II-C. With this optimization formulation, we can use nonlinear programming (NLP) solvers such as IPOPT [31], and SNOPT [32] to
efficiently synthesize an optimal walking gait. The end result is a stable periodic solution to the walking dynamics that is parameterized by some static parameter $\alpha^*$. 

C. Essential Constraints

Traditionally, operators manually select the bounds on key constraints such as walking frequency, walking velocity, step length, foot clearance, and impact velocity. These constraints are essential to achieving experimental robustness, and hence are termed here essential constraints. Often, practitioners derive intuition from years of experience on how to shape these constraints. One example of how this intuition relates to stability is Raibert-type controllers [2], which tune constraints such as walking frequency, walking velocity, step length, foot clearance, and impact velocity.

C. Essential Constraints

*essential constraints defined as an action in Sec. III.

Essential constraints are termed here essential constraints. Often, practitioners derive intuition from years of experience on how to shape these constraints. One example of how this intuition relates to stability is Raibert-type controllers [2], which tune constraints such as walking frequency, walking velocity, step length, foot clearance, and impact velocity.

Algorithm 1 LINECOSPARNLP

1: function LINECOSPARNLP($v$ essential constraints, $a_{\min}$, $a_{\max}$, $d_\kappa$ for $\kappa = 1, ..., v$, $n$)
2: $D_0 = \emptyset \triangleright D_1$: Preference feedback including iteration $i$
3: $E_0 = \emptyset \triangleright E_1$: Executed actions including iteration $i$
4: Obtain $n$ uniformly-random actions: $A_i = \{a_{ij}, ..., a_{in}\}$
5: Execute outputs of NLP for $A_i$ on the system
6: Query operator for preference feedback: $y_i$
7: Append executed actions: $E_i = E_{i-1} \cup A_i$
8: Append preference feedback: $D_i = D_{i-1} \cup y_i$
9: for $i = 2, 3, ..., i$ do
10: Obtain covariance $[\Sigma]_{j,k} = K(a_j, a_k)$ for $a \in E_{i-1}$
11: Update posterior over $E_{i-1}$ to obtain $(\mu_{i-1}, \Sigma_{i-1})$
12: Update $a^*_{i-1} = \text{argmax}_\{a \in E_{i-1}\} \mu_{i-1}(a)$
13: $\mathbf{L}_i = \text{random line through } a^*_{i-1}$, discretized via $m_v$
14: Update subset $S_i = \mathbf{L}_i \cup E_{i-1}$
15: Obtain covariance $[\Sigma]_{j,k} = K(a_j, a_k)$ for $a \in E_i$
16: Update posterior over $S_i$ to obtain $(\mu_i, \Sigma_i)$ given $D_{i-1}$
17: Draw $j = 1, ... , n$ samples: $f_j \sim \mathcal{N}(\mu_i, \Sigma_i)$
18: Update actions $A_i = \{a_j = \text{argmax}_a f_j(a) \mid j = 1, ... , n\}$
19: Run NLP for actions $A_i$, to obtain
20: Execute outputs of NLP for $A_i$ on the system
21: Query operator for preference feedback $y_i$
22: Append executed actions: $E_i = E_{i-1} \cup A_i$
23: Append preference feedback: $D_i = D_{i-1} \cup y_i$
24: end for
25: Obtain covariance $[\Sigma]_{j,k} = K(a_j, a_k)$ for $a \in E_f$
26: Update posterior over $D_f$ to obtain $(\mu_f, \Sigma_f)$
27: Obtain $a^*_f = \text{argmax}_a \mu_f(a)$
28: end function

In this work, we extend the learning framework to include the nonlinear optimization problem directly which eliminates the need for pre-computed gait libraries and allows for a more diverse set of behaviors. We term this slightly modified algorithm LINECOSPARNLP where the goal is to identify the optimal action $a^* \in A$ that maximizes $f(a)$ of the human operator in as few iterations as possible. For example, $f(a)$ may account for stability, and robustness to perturbations and model uncertainty. This multivariate utility function often admits no mathematical description; rather, the operator has intuition about what looks “right”. Information about $f(a)$ is obtained through pairwise preferences. To learn from these preferences, we adopt the dueling bandit setting [37] in which the algorithm selects actions to execute on the system and receives a user’s relative preferences between the executed actions.

A. The LINECOSPARNLP Algorithm

The procedure of the LINECOSPARNLP algorithm (Alg. 1) is as follows. First, the algorithm begins by randomly selecting and executing $A_1$ where $A_1 \in \mathbb{R}^{v \times n}$ is the $n$ actions selected to execute on the system for iteration $i$. The parameter $n$ can be changed depending on how many actions the operator would like to sample in each iteration. Since the actions are compared in pairs, $n$ actions equates to $m = \binom{n}{2}$ pairwise preferences. These actions are given to the NLP and $n$ corresponding gaits are generated, which are then executed on hardware. We define the set of actions
executed on hardware up to and including those sampled in iteration \( i \) as \( \mathbf{E}_i := \bigcup_{j=1,...,i} \mathbf{A}_j \subset \mathbf{A} \).

After demonstrating the gaits on hardware, the human operator is queried for \( m \) pairwise preferences, denoted as \( y_{ij} \) for iteration \( i \). \( \mathbf{D}_i = \bigcup_{j=1,...,i} y_{ij} \in \mathbb{R}^{N_p^{(i)}} \) is defined as the dataset of all preference feedback up to and including iteration \( i \) with corresponding number of preferences \( N_p^{(i)} \). Preference feedback is omitted when all sampled actions do not converge, or when the user chooses to give feedback of “no preference”. Thus, \( N_p^{(i)} = \text{inv} - N_{\text{omit}} \) where \( N_{\text{omit}} \) is number of omitted preferences.

In each iteration, the posterior is updated over the previously executed actions \( \mathbf{E}_{i-1} \) given the dataset \( \mathbf{D}_{i-1} \) by modeling the underlying utilities of \( \mathbf{E}_{i-1} \), denoted as \( f_{\mathbf{E}_{i-1}} \), through the Gaussian posterior:

\[
P(f_{\mathbf{E}_{i-1}}|D_{i-1}) \propto P(D_{i-1}|f_{\mathbf{E}_{i-1}})P(f_{\mathbf{E}_{i-1}}).
\]

Thus, to calculate this posterior, we first calculate the Gaussian process prior defined over the utilities \( f_{\mathbf{E}_{i-1}} \) as:

\[
P(f_{\mathbf{E}_{i-1}}) = \frac{1}{(2\pi)^{N_p'/2} |\Sigma_p'|^{1/2}} \exp\left(-\frac{1}{2} f_{\mathbf{E}_{i-1}} (\Sigma_p')^{-1} f_{\mathbf{E}_{i-1}}\right),
\]

where \( \Sigma_p' \in \mathbb{R}^{|\mathbf{E}_{i-1}| \times |\mathbf{E}_{i-1}|} \), \( |\Sigma_p'|_{j,k} = K(a_{ij}, a_{jk}) \), and \( K \) is a kernel of choice. In this work we utilize a squared exponential kernel. Second, the likelihood \( P(D_{i-1}|f_{\mathbf{E}_{i-1}}) \) is computed as:

\[
P(D_{i-1}|f_{\mathbf{E}_{i-1}}) = \prod_{k=1}^{N_{\text{pr}}^{(i)}} g_{\text{log}}(f(a_{k1}) = f(a_{k2}))
\]

with \( g_{\text{log}}(x) \) as the chosen link function. This link function can be replaced with any monotonically-increasing function, but we found empirically that the heavy-tailed distribution \( g_{\text{log}}(x) := \frac{1}{1 + e^{-x}} \) improves performance. Finally, the posterior (7) is estimated via the Laplace approximation as in [38] which yields a multivariate Gaussian, \( \mathcal{N}(\mu_{i-1}, \Sigma_{i-1}) \). The posterior is then used to update the optimal action:

\[
a^*_i := \arg\max_{a \in \mathcal{C}} \mu_{i-1}(a).
\]

A random linear subspace \( \mathbf{L}_i \subset \mathbf{A} \) is then generated to intersect \( a^*_i \). The underlying utilities of the actions within the subset \( \mathcal{S}_i := \mathbf{L}_i \cup \mathbf{E}_{i-1} \subset \mathbf{A} \), denoted as \( f_{\mathcal{S}_i} \), are then modeled through the Gaussian posterior:

\[
P(f_{\mathcal{S}_i}|D_{i-1}) \propto P(D_{i-1}|f_{\mathcal{S}_i})P(f_{\mathcal{S}_i}),
\]

which is calculated as with the posterior over \( \mathbf{E}_{i-1} \).

Next, \( n \) new actions are sampled using the Self Sparring approach [39] to Thompson sampling, a regret minimization sampling method. This method selects actions to execute by first sampling \( n \) utility functions, denoted \( f_j \) for \( j = 1, ..., n \), from the posterior distribution \( \mathcal{N}(\mu_i, \Sigma_i) \). The selected actions, \( \mathbf{A}_i \), are those which maximize the sampled functions. These are then given to the NLP, whereby corresponding gaits are generated, the outputs are executed on the robot, and \( \mathbf{A}_i \) is appended to \( \mathbf{E}_i \). The operator is again queried for preference feedback which is added to \( \mathbf{D}_i \).

**IV. LEARNING TO WALK IN EXPERIMENTS**

We experimentally deploy LINECoSPARNLP on the planar bipedal robot, AMBER-3M [40]. This custom research platform has three interchangeable lower-limb configurations (flat-foot, point-foot, and spring-foot) and was originally built to study how leg configurations affect energy efficiency. We specifically selected this platform because of its engineering reliability [12], enabling consistent data collection to isolate the effects of various gaits in the learning process. The controller for AMBER-3M is implemented on an off-board i7-6700HQ CPU @ 2.6GHz with 16 GB RAM, which computes desired torques and communicates them with the
ELMO motor drivers. The motor driver communication and the control logic ran at \( \sim 1 \text{kHz} \), each on a separate core.

A. Experimental Procedure

In the experiments (video: [41]), walking gaits are generated by the HZD trajectory optimization method presented in Sec. II. We take \( y^d(q) := q_a \in \mathbb{R}^4 \) as the position of the four motorized joints of AMBER-3M, \( \tau(q) \) to be the linearized forward hip position, and use a 5th-order Bézier polynomial \((\alpha \in \mathbb{R}^{4 \times 6})\) to describe the desired output trajectories. Additionally, the cost function is selected to be the mechanical cost of transport (MCOT), a common metric for locomotion efficiency defined by:

\[
MCOT = \int_{t_0}^{T_f} \frac{P(t) \| u(t) \|}{mgv} dt, \tag{8}
\]

where \( P(t) = \| u(t)^T q_a(t) \|_2 \), the 2-norm sum of positive power. The essential constraints selected to tune through learning are:

1. Average forward velocity of the torso \((m/s)\)
2. Phase variable value at which to enforce minimum foot clearance, \( \tau_c \)
3. Minimum nonstance foot clearance enforced at \( \tau_c \) \((m)\)
4. Downward velocity enforced at impact \((m/s)\)
5. Step length, i.e. the forward distance between swing foot and stance foot at impact \((m)\)

The average optimization run time is 0.1 second per iteration, with each gait averaging 160 iterations. The experimental procedure is illustrated in Fig. 2. In our experiments, the learning was conducted for \( n = 2 \), corresponding to two gaits being compared in each iteration. This was chosen because fewer pairwise comparisons are easier to give feedback between, and thereby result in less noise in the preferences. Note that other applications may benefit in a higher \( n \), which would increase the rate of learning.

Each trial began by initializing AMBER-3M in a stance configuration, starting the treadmill, and attempting to push the robot into its periodic orbit. If the robot seemed like it would fall, extra precaution was taken to give the gait the best chance at succeeding. Once the gait reached its orbit, the robot was released and the robustness of the gait to various disturbances was investigated. After both gaits were executed on the physical robot, preferences were collected from the human operator observing the physical realization of the walking. In some iterations, video footage was also reviewed before giving a preference. The criteria used to determine preferences between gaits were the following (in order of prioritization):

- Capable of walking
- Robust to perturbations in treadmill speeds
- Robust to external forces
- Does not exhibit harsh noise (e.g. during impact)
- Is visually appealing (intuitive judgment from operator)

B. Procedure specific to AMBER-P and AMBER-S

In this work, we leverage two configurations of the robot: 1) the point-foot configuration \((1.373 \text{ m tall, } 21.3 \text{ kg})\), AMBER-P; and 2) the spring-foot configuration \((1.430 \text{ m tall, } 23.5 \text{ kg})\), AMBER-S. We first demonstrate the learning framework on AMBER-P, with the corresponding robot model used in the gait generation. To emphasize the scalability of LINECO\textsuperscript{SPARNL\textsubscript{P}}, we also repeat the exact same procedure applied to AMBER-S while still using the AMBER-P model in the trajectory optimization process. We intentionally do not account for these changes and instead gaits were was still generated assuming the rigid body model and executed on hardware using the same controller with unmodified PD gains. Even though we don’t account for these changes, the addition of compliance in the legs increases the degrees of freedom of the system, adds a double support domain to the hybrid dynamics, and increases the stiffness of the dynamics. Historically, robots with compliance are difficult to generate gaits for because of these added complexities, with past success relying on sophisticated models [42]. Therefore, the fact that the algorithm converges despite these unmodeled complexities highlights the effectiveness of the LINECO\textsuperscript{SPARNL\textsubscript{P}} learning method towards achieving experimental robustness.

C. Results

During the experiment on AMBER-P, the gaits quickly met the first criterion of being able to walk; therefore, as the trials progressed, the efficiency, robustness, and naturalness became the key criteria in determining preferences between gaits. The experiment using the rigid model was run for a total of 30 iterations and sampled 27 unique gaits. The final posterior over the 27 sampled actions is illustrated in the top row of Fig. 3. To elucidate the success of the LINECO\textsuperscript{SPARNL\textsubscript{P}}, three gaits are selected from the experiment for careful investigation corresponding to the
minimum, a middle, and the maximum posterior utility. The iteration numbers corresponding to when these gaits were first sampled is 1, 21, and 26, respectively.

The power of this method is truly demonstrated by its ability to facilitate the experimental realization of robust gaits. The initial gaits tried on hardware, although optimal subject to the imposed constraints, resulted in inferior trajectory tracking and power consumption. As the algorithm progressed, the gaits became significantly smoother, more robust to disturbance, and energy efficient. This is exemplified in Figure 4 by noting the significantly lower velocity overshoot for all of the limbs and tighter tracking shown in the phase portraits for the gaits with higher posterior utility. The improvement in energy efficiency is illustrated by the decreasing MCOT values, which were 0.74, 0.95, and 0.26 corresponding to the three gaits, respectively.

When the procedure was repeated on the AMBER-S platform, many of the initial tested gaits were unable to walk due to the unmodeled compliance. Thus, gaits exhibiting periodic walking behavior were strongly preferred. This second experiment was conducted for 50 iterations and sampled 37 unique gaits with the obtained posterior illustrated in the bottom row of Fig. 3. As with the previous experiment, three gaits are selected for further discussion corresponding to the minimum, a middle, and the maximum posterior utility values. Gait tiles and phase portraits for these are again shown in Figure 4. The iterations when these gaits were first sampled are 4, 10, and 42. Once again, the algorithm converges to gaits with superior trajectory tracking and lower MCOT (1.16, 0.38, and 0.33, respectively).

V. CONCLUSION

In this work, we present and experimentally demonstrate a preference-based learning framework, LINECoSPARNL, specifically designed for use towards gait generation via HZD optimization. The success of the proposed learning method is demonstrated through its ability to experimentally realize gaits that are stable, robust to model uncertainty, robust to external perturbations, efficient, and natural looking with no requirement for simulation within 50 experimental iterations.

LINECoSPARNL incorporates preference-based learning with HZD optimization to leverage the theoretical benefits of HZD without the challenge of parameter tuning. Furthermore, preference-based learning is a sample-efficient learning method that does not require the user to mathematically define a metric for “good” walking. Instead, the framework relies on easy to provide pairwise preferences. The success of LINECoSPARNL is demonstrated by achieving robust walking with unmodeled compliant legs, a historically challenging control task.

Future work includes extending this framework to more robotic platforms, such as quadrupeds and 3D bipedal robots, as well as improving the sample-efficiency of the framework through additional qualitative feedback mechanisms such as ordinal labels [43]. The experimental results presented in this paper demonstrate the rich potential lying in the boundary between machine learning and control theory. It is well-known that control theory provides necessary structure to bipedal platforms, but machine learning can play a critical role in shaping the final behavior of the system.
