Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

Design and Control of a Bipedal Robotic Character

Ruben Grandia*, Espen Knoop*, Michael A. Hopkins†, Georg Wiedebach†,
Jared Bishop‡, Steven Pickles‡, David Müller*, and Moritz Bächer*
*Disney Research, Switzerland, †Disney Research, USA, ‡Walt Disney Imagineering R&D, USA

***Abstract*—Legged robots have achieved impressive feats in dynamic locomotion in challenging unstructured terrain. However, in entertainment applications, the design and control of these robots face additional challenges in appealing to human audiences. This work aims to unify expressive, artist-directed motions and robust dynamic mobility for legged robots. To this end, we introduce a new bipedal robot, designed with a focus on character-driven mechanical features. We present a reinforcement learning-based control architecture to robustly execute artistic motions conditioned on command signals. During runtime, these command signals are generated by an animation engine which composes and blends between multiple animation sources. Finally, an intuitive operator interface enables real-time show performances with the robot. The complete system results in a believable robotic character, and paves the way for enhanced human-robot engagement in various contexts, in entertainment robotics and beyond.**

I. Introduction

Legged robotic platforms have gained widespread accessibility and are often envisioned as versatile mobile platforms suitable for navigating challenging and unstructured environments. Consequently, most robotic systems are engineered and controlled with utility and efficiency as the primary objectives [15]. This has led to remarkable achievements in dynamic legged systems that can now hike up mountains [28] and conquer obstacle courses [13].

Nevertheless, robots are increasingly being deployed beyond isolated environments, in places where they engage directly with humans. A rich set of applications can be found in collaborative robots [48], companion robots [36], art [22], and entertainment [7]. These applications introduce additional challenges to the design and control of robots, as the success of the robot is additionally dependent on the subjective perception of humans. Legged systems must perform these expressive motions while simultaneously satisfying the already complex kinematic and dynamic balancing requirements inherent to whole-body locomotion.

Animators have mastered the art of breathing life into digital characters [29], and a long-standing aspiration has been to endow robots with an equivalent level of expressiveness [45]. Learning-based approaches to physics-based characters have made it possible to learn control policies from animation or motion capture input [32, 33]. Still, significant effort is required to translate such a performance from simulation to the real world. Meanwhile, other studies explore how robots can spark an emotional response through facial expressions, visual cues, or body language, although many of the robots used have limited mobile capabilities [47].

Fig. 1. Three instances of our robotic character performing an unscripted show. Apart from their theming, they are identical. Each robot is remote-controlled by a separate operator.

In this work, we aim to bring expressive and dynamic motions onto a bipedal robotic character, and explore the intersection of legged robot design, control, and character animation. We present a new robot character, shown in Fig 1, with a mechanical design that is primarily driven by creative intent and simplicity rather than functional requirements. Additionally, we present a complete pipeline centered around reinforcement learning to bring animations onto the physical system. Our pipeline allows the robot to imitate artistic motions, and, based on user input, blend or seamlessly transition between them while remaining robust to uncertainty and external disturbances. Finally, we propose an intuitive *puppeteering* interface that combines direct motion control with expressive animations. Through this interface, an operator is able to author believable interactions in real-time. The complete system allows us to rapidly explore robotic performances in an entertainment context. Similar building blocks could be used to create more expressive autonomous robots.

Succinctly, in addition to the presented system as a whole, contributions of this work include:

- A workflow that integrates animation content, design, control, and real-time puppeteering, and enables the rapid development of custom robot characters.[1].
- A new robot whose morphology and kinematics are driven by creative intent, rather than by function.

[1]The presented robot was developed by the authors in less than a year

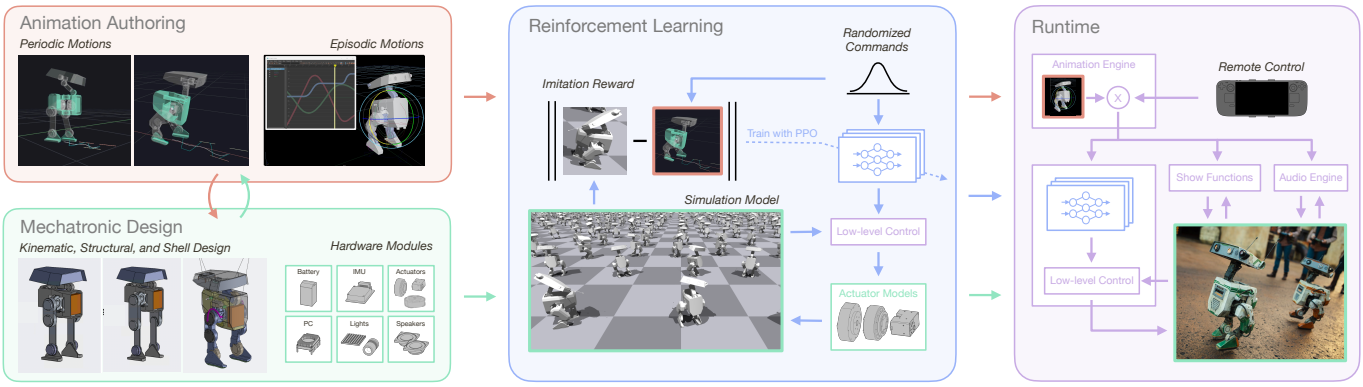

Fig. 2. Our character design and control pipeline consists of animation, mechatronic design, reinforcement learning, and run-time tools. Animation and mechatronic design form an iterative process to define the character and its motion repertoire. These inputs are used in reinforcement learning which, through imitation rewards, results in control policies that robustly execute the intended motions, conditioned on external commands. During run-time, the animation engine combines user-inputs and animation content and interfaces with the control policies. In parallel, the animation engine synchronizes show-functions with the motion.

- A breakdown of different motions into separate categories and control policies, switched between at runtime.
- A puppeteering interface that leverages conditional policy inputs, and enables robot performances by layering, blending and switching between different motion elements.

II. Related Work

Several robots, initially intended for high-performance dynamic locomotion, have been used to perform expressive motions [24], particularly in the context of dancing [3, 1]. On the other hand, various robots have been purposefully designed for social interaction with humans [18, 20]. Notably, humanoids such as iCub [27], NAO [9], and Pepper [31], have been widely used in the research community, serving both as platforms for motion control and human-robot interaction (HRI) research.

However, these robots in previous studies were conceived as general-purpose platforms, and their demonstrations were often conducted as part of distinct projects or by entirely different organizations. Furthermore, their humanoid form naturally drives the mechanical design and comes with abundant human reference motions to be used for control. In this work, we introduce a robot where both the mechanical design and the motion that it performs are co-developed and driven by an artistic vision to create a unified character.

A. Animating Robots

The influence of a robot's movements on human perception has been widely studied and acknowledged [46]. Van Breemen proposed to apply the principles of animation to bring robots to life [45], and presented an *animation engine* that executes, composes, and blends multiple animations based on external commands [44]. Similarly, Fujita *et al.* outlined a software architecture for entertainment robots [8], applied to robots like the quadrupedal AIBO [7] and the humanoid SDR-4X [6]. Later, for the NAO platform, a graphical tool was introduced to make behavior programming more accessible [35]. In this paper, we leverage the motion composition and blending principles outlined in these works. However, we do not present a fully autonomous system that plans and executes long-term behaviors. Instead, we leave such high-level decisions to a puppeteer. With the proposed puppeteering interface and control policies that accept high-level inputs, we strive for a balance between the imitation of fixed artist-created motion and real-time interactive show authoring. We compose continuous inputs like the robot gaze, triggered animations, and autonomous stylized motion that is embedded in our low-level controller.

B. Controlling Legged Systems

A key challenge in controlling legged robots are the complex requirements related to the kinematics and under-actuated dynamics of the system. To navigate the constrained space of feasible solutions, model-based optimization approaches have been proposed as a tool to convert animations into dynamically feasible reference trajectories [41, 30, 24, 10]. Still, an online controller is needed to stabilize the system as it inevitably deviates from the planned trajectory, for example through Model Predictive Control (MPC) [4]. We use model-based tools during the design phase to study the performance of the character and interpolate animated walk cycles.

Alternatively, Reinforcement Learning (RL) has become a popular choice for synthesizing closed-loop control policies directly from imitating reference motions [32, 33, 12]. In a similar spirit, RL has been used to imitate solutions from a model-based motion planner [19] or gait library [25]. Clustering motions and learning a mixture-of-experts have been proposed to scale these methods to a large and diverse set of motions. Alternatively, an adversarial reward can be used instead of tracking rewards to imitate behaviors from unstructured large scale human motion datasets [34, 5]. However, for our unique character such a dataset is not readily available.

While most related work strives for a single control policy, we adopt a divide-and-conquer strategy and train separate policies to imitate artist-authored walking, standing, and short animation sequences, conditioning the individual policies on

carefully chosen high-level control commands. When paired with a real-time compositing and blending layer, these commands then enable the seamless and intuitive puppeteering of the robot with a two-joystick remote controller.

III. Overview

Our character design and control workflow, outlined in Fig. 2, starts with an iterative process between mechanical design and animation. Classical animation tools, which use a rig consisting of links and spherical joints, are used to study the character and view it in the context of a scene. These studies inform the general proportions and range of motion of the character. A procedural gait generation tool is used to create periodic walking cycles. This tool builds on the rigid body dynamics of the system, and therefore generates physically plausible motions. The joint positions, velocities, and torques feed back into the mechanical design to refine geometry, select actuators, and perform structural analysis. This combination of tools allows us to rapidly explore motions, walking styles, and mechanical designs, to ultimately find the right trade-off between the physical limits of available hardware modules and the creative intent.

We strive not for the best possible mechanical design, but for a simple design which satisfies the creative intent. By using off-the-shelf hardware modules and characterizing them well in software, we can reduce hardware complexity while maintaining system performance.

After converging on an initial set of motions and mechanical design, both are used to define a reinforcement learning problem. The mechanical design forms the simulation model, together with actuator models and domain randomization. Kinematic motion references are exported from the animation tools and used in imitation rewards that maximize the similarity between simulated and reference motion. A set of commands are defined that provide high-level control of the robot. We then train multiple policies, one per motion or motion type, and condition them on these commands. This process results in policies that can robustly execute the defined animations while providing control over the character through policy switches and command signals.

During run-time, the animation engine receives user-input from a remote control interface. The animation engine fuses these inputs with predefined animations to generate commands for the control policies. Additionally, the animation engine triggers policy switches. Show functions and audio are controlled elements of the robot that play a key role in expressing character, but do not affect the dynamics of the system. Their behavior is synchronized with the motion of the robot through animation signals from the animation engine and state feedback.

IV. Mechatronic Design

Our final bipedal robotic character design is shown in Fig. 3. The robot has 5 degrees of freedom (DoF) per leg and a 4 DoF neck and head assembly. The large workspace of the two legs enables a wide range of dynamic locomotion and lower body

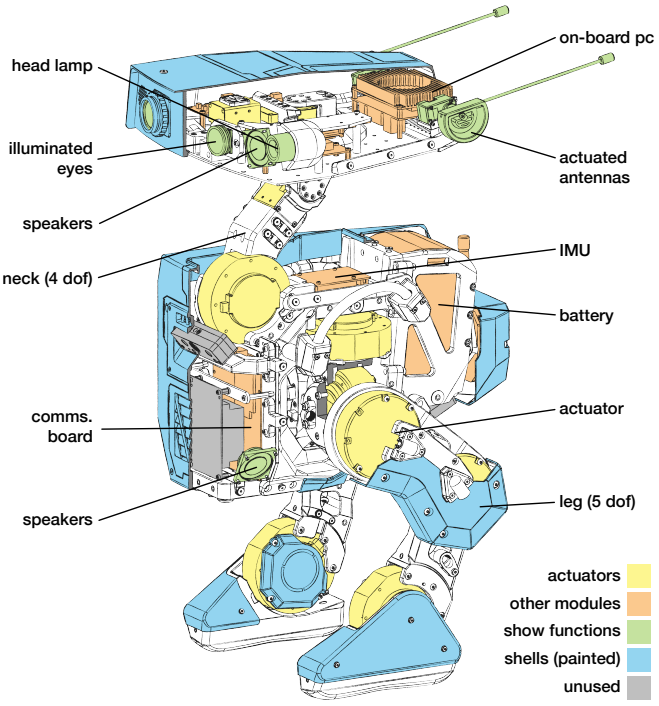

Fig. 3. Mechanical design of our robotic character. The robot has 5 degrees of freedom per leg. The neck and head assembly has 4 degrees of freedom. The torso contains a custom communication board, a battery module, and an IMU. The onboard PC, radio receiver, and show function board are located in the head. Show functions consists of antennas, LED arrays as eyes, and a head lamp. A pair of speakers is located in the head and another pair in the bottom of the torso.

motions while the head can be posed independently relative to the torso. Similar to ANYmal [16], we adopted a design where actuators are directly placed at joints. To build our custom robot quickly, we 3D printed the components that connect pairs of off-the-shelf actuators. As a result of this design philosophy, the robot has the ankle actuators directly placed at the two feet and is not equipped with ankle roll actuators. To allow for passive ankle roll, we rounded off the two foot soles. By molding them with urethane foam, foot impacts on hard ground are further dampened. The knee joints bend backwards as creatively envisioned.

The robot has a total mass of $15.4\,\mathrm{kg}$, with the torso weighing $5.8\,\mathrm{kg}$, the neck and head $2.4\,\mathrm{kg}$, and each leg $3.6\,\mathrm{kg}$. It is $0.66\,\mathrm{m}$ tall (excluding antennas) and the legs have a nominal length of $0.28\,\mathrm{m}$ and an extended length of $0.34\,\mathrm{m}$.

The hip-adduction-abduction, hip-flexion-extension, and knee actuators are strongest with a peak torque of $34\,\mathrm{N\,m}$ and maximum velocity of $20\,\mathrm{rad\,s^{-1}}$. The hip-rotation, ankle, and the lower neck actuator have a peak torque of $24\,\mathrm{N\,m}$ and maximum velocity of $30\,\mathrm{rad\,s^{-1}}$. These two types are quasi-direct drive actuators and support high-bandwidth open-loop torque control suitable for dynamic locomotion [49]. The 3 actuators located in the head have a high gear ratio with a peak torque of $4.8\,\mathrm{N\,m}$ and maximum velocity of $6.3\,\mathrm{rad\,s^{-1}}$.

A custom microcontroller-driven communications board

serves as the interface between the on-board PC, actuators, and an Inertial Measurement Unit (IMU), providing a communication rate of $600\,\mathrm{Hz}$. All actuators have integrated drives which implement a low-level control loop and report motor position as measured by their built-in encoders. The on-board PC communicates with the hand-held operator controller through redundant wireless communication using both WiFi and LoRa radios. A removable battery powers the robot for at least $1\,\mathrm{h}$ of continuous operation.

What sets our robot apart from other legged systems is a set of show functions: a pair of actuated antennas and illuminated eyes, and a head lamp. These functions provide animators with an additional means to express emotions. However, since they do not affect the system dynamics and are controlled in an open-loop fashion, we treat them separately from the main actuators. This allowed us to easily add and remove show functions during the design phase of the robot. In addition to these functions, we equipped the robot with a stereo pair of loudspeakers in both the body and the head.

V. Reinforcement Learning

As outlined in Fig. 2, we control the robot with multiple policies that we condition on a time-varying control input $\boldsymbol{g}_t$. At each time step, the agent produces an action $\boldsymbol{a}_t$ according to a policy, $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t, \phi_t, \boldsymbol{g}_t)$, provided with the observable state $\boldsymbol{s}_t$ and optional conditional inputs ϕ_t and $\boldsymbol{g}_t$, where ϕ_t is a phase signal. During training, the environment then produces the next state, $\boldsymbol{s}_{t+1}$, updates the phase signal, and returns a scalar reward $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \phi_t, \boldsymbol{g}_t)$. We reward the close imitation of artist-specified kinematic reference motions, and also dynamic balancing.

To bring structure into the broad range of possible character performances, we use differences in their temporal properties to define three motion types:

- *Perpetual* motions do not have a clear start and end. The robot maintains balance and responds to the measured state and a continuous stream of control inputs.
- *Periodic* motions are characterized by a periodic phase signal which is passed to the policy. In this mode, the phase signal cycles indefinitely.
- *Episodic* motions have a predefined duration. Policies receive a monotonically-increasing phase signal. Once the motion ends, a transition to a new motion is forced.

For a minimally complete walking character, it is advisable to train at least one policy for perpetual motion and one to imitate a periodic motion. Episodic policies are optional, but well-suited to have the robot express distinct emotions. We treat the training of each policy as a separate RL problem and randomize the control inputs, g_t, over their full range during training. This enables us to control the robot with arbitrary control inputs within their target range, and to switch between policies on the fly during runtime.

For the presented character, we trained one *perpetual* policy for standing while controlling the head and torso, one *periodic* policy for walking with separate head control, and several *episodic* policies that are each trained to imitate a single animation sequence. In addition, we perturb the simulation model with randomized disturbances and model parameters. In this way, the agent is able to robustly perform the intended motions under a broad distribution of states and user inputs. In the following, we describe these elements in more detail.

A. Animation Input

We interface with animation content by extracting kinematic motion references that define the character's time-varying target state,

$$\boldsymbol{x}_t = (\boldsymbol{p}_t, \boldsymbol{\theta}_t, \boldsymbol{v}_t, \boldsymbol{\omega}_t, \boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, c_t^L, c_t^R), \tag{1}$$

where $\boldsymbol{p}_t$ is the global position of the torso and $\boldsymbol{\theta}_t$ its orientation, represented with a quaternion. $\boldsymbol{v}_t$ and $\boldsymbol{\omega}_t$ are the torso's linear and angular velocity, and $\boldsymbol{q}_t$ and $\dot{\boldsymbol{q}}_t$ the joint positions and velocities. The indicator variables, c_t^L and c_t^R, define the contact state of the character's left and right foot.

For each motion type, we assume that we have access to a generator function f that maps a path frame, $\boldsymbol{f}_t$, and an optional phase signal and type-dependent control input to the kinematic target state

$$\boldsymbol{x}_t = f^{\text{perp}}(\boldsymbol{f}_t, \boldsymbol{g}_t^{\text{perp}}) \tag{2}$$

$$(\boldsymbol{x}_t, \dot{\phi}_t) = f^{\text{peri}}(\boldsymbol{f}_t, \phi_t, \boldsymbol{g}_t^{\text{peri}}) \tag{3}$$

$$\boldsymbol{x}_t = f^{\text{epis}}(\boldsymbol{f}_t, \phi_t). \tag{4}$$

The reference generator for periodic motions additionally outputs the phase rate, $\dot{\phi}_t$, that drives the phase signal. This allows us to vary the stepping frequency during walking as a function of the commands. For episodic motions, the phase rate is determined by the duration of the motion. See Fig. 4 left for a visualization of the path frame that we use to parameterize our motion references.

For the perpetual standing motion of our specific character, we found that a control input that consists of head and torso commands provides a versatile interface as we illustrate in Fig. 4 with the limits of each individual head (top row) and torso command (bottom row): We command the head's height and orientation relative to a nominal configuration with a height and orientation offset, Δh_t^{head} and $\Delta\boldsymbol{\theta}_t^{\text{head}}$. The torso position and orientation is commanded with a height, h_t^{torso}, and orientation, $\boldsymbol{\theta}_t^{\text{torso}}$, represented with ZYX-Euler angles, in path frame coordinates

$$\boldsymbol{g}_t^{\text{perp}} = (\Delta h_t^{\text{head}}, \Delta\boldsymbol{\theta}_t^{\text{head}}, h_t^{\text{torso}}, \boldsymbol{\theta}_t^{\text{torso}}). \tag{5}$$

For the periodic motion of our droid-like character, we use the same head commands, and use a 2D velocity vector, $\boldsymbol{v}_t^{\mathcal{P}}$, and angular rate, $\omega_t^{\mathcal{P}}$, both expressed in path frame coordinates

$$\boldsymbol{g}_t^{\text{peri}} = (\Delta h_t^{\text{head}}, \Delta\boldsymbol{\theta}_t^{\text{head}}, \boldsymbol{v}_t^{\mathcal{P}}, \omega_t^{\mathcal{P}}). \tag{6}$$

Note that for periodic motion we assume the head commands to be offsets relative to a nominal head motion that the animator specifies as part of the periodic walk cycle. This set of commands puts artists in control of the velocity-dependent lower-body motion while the corresponding references for the

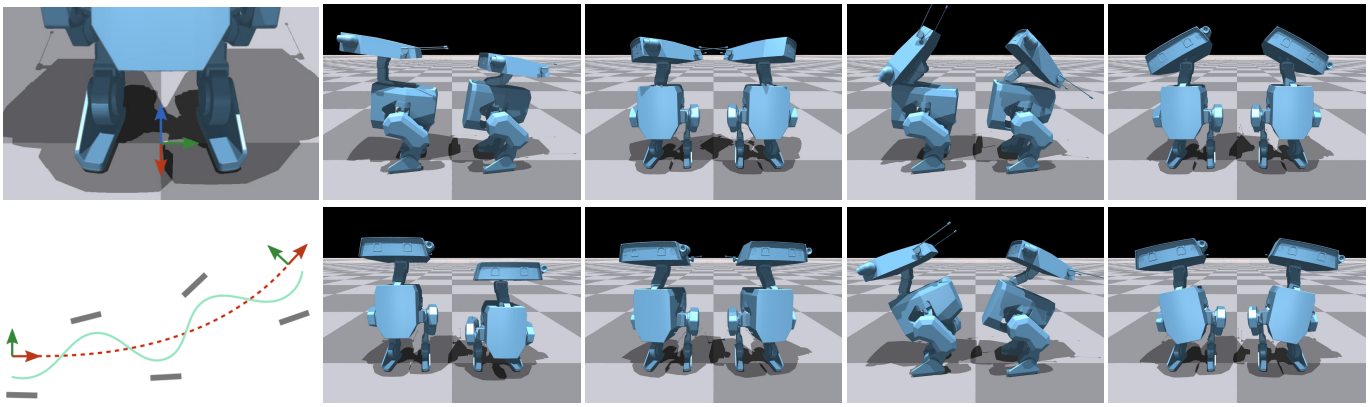

Fig. 4. Path frame illustrations for standing (top-left), and a top-view during walking (bottom-left). During standing, the path frame converges towards the average position and heading of the feet. During walking, the path frame is integrated according to the path velocity commands. The horizontal torso trajectory, shown in blue, will generally sway around the path frame to shift weight between the feet. The other images show the kinematic reference pose for the perpetual standing motion at minimum and maximum value for head commands (top) and torso commands (bottom). The input dimension from left to right are: up-down, yaw, pitch, roll.

neck-head assembly can be adapted to have the robot look in a particular direction during locomotion.

The path frame plays a fundamental role in maintaining consistency during motion transitions. Each artist-designed motion is stored in path coordinates and mapped to world coordinates based on the path frame state according to the generators f. During standing, the path frame slowly converges to the center of the two feet. During walking, the next frame is computed by integrating the path velocity commands. This is visualized in Fig. 4 left. For episodic motions, the path frame trajectory relative to the starting location is part of the artistic input. Finally, to prevent excessive deviation from the path, $\boldsymbol{f}_t$ is projected to a maximum distance from the current torso state.

Our processing is agnostic to the tool or technique that an artist uses to generate kinematic reference motion and a detailed description of our specific implementation is beyond the scope of this paper. Hence, we provide a high-level summary here. To generate perpetual references, we use inverse dynamics to find a pose that satisfies the commands $\boldsymbol{g}_t^{\text{perp}}$ and solves for optimal values for the remaining degrees of freedom such that the center of pressure is in the middle of the support polygon. For periodic walking motions, an artist provides reference gaits at several walking speeds, defined as task space trajectories for torso and end-effectors. These gait samples are procedurally combined [14] into a new gait based on the commands $\boldsymbol{g}_t^{\text{peri}}$. To translate the task-space reference to whole-body references, first a model predictive controller similar to [51] is used to plan the desired center of mass and center of pressure. Subsequently, these references are then tracked by an inverse dynamics controller similar to [21] to obtain whole-body trajectories. This processing is part of an interactive tool. The animator can directly preview the resulting gait and fine-tune the task-space references accordingly. The episodic motions are generated in Maya [2].

To prevent the rate at which we can generate reference motions from slowing down training, we densely sample the reference generators and implement the reference look-up during RL as interpolation of these samples.

B. Reward

The reward function combines a motion-imitation rewards with additional regularization and survival rewards,

$$r_t = r_t^{\text{imitation}} + r_t^{\text{regularization}} + r_t^{\text{survival}} \tag{7}$$

Following previous work on tracking-based imitation learning [23, 32], we compute $r_t^{\text{imitation}}$ directly by comparing the simulated to the target pose of the character. The agent receives additional rewards if the foot contact states match the reference states. To mitigate vibrations and unnecessary actions, we apply regularization rewards that penalize joint torques and promote action smoothness. A final survival reward provides a simple objective that motivates the character to stay alive and prevents it from seeking early termination of the episode at the beginning of training. We apply early termination when either head or torso are in contact with the ground and also if we detect a self-collision between the head and torso. A detailed description of the weighted reward terms are provided in Tab. I where we use a hat for target state quantities from the reference pose $\mathbf{x}_t$ and omit the current time index t but add spatial indices where necessary.

C. Policy

Our policy actions, $\boldsymbol{a}_t$, are joint position setpoints for proportional-derivative (PD) controllers. In addition to an optional motion-specific phase and control command, our policies receive a state

$$\boldsymbol{s}_t = (\boldsymbol{p}_t^{\mathcal{P}}, \boldsymbol{\theta}_t^{\mathcal{P}}, \boldsymbol{v}_t^{\mathcal{T}}, \boldsymbol{\omega}_t^{\mathcal{T}}, \boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{a}_{t-1}, \boldsymbol{a}_{t-2}) \tag{8}$$

as input. To make the state and policy invariant to the robot's global location, we represent the torso's horizontal (xy-plane) position, $\boldsymbol{p}_t^{\mathcal{P}}$, and orientation, $\boldsymbol{\theta}_t^{\mathcal{P}}$, in path frame coordinates, and the torso's linear and angular velocities, $\boldsymbol{v}_t^{\mathcal{T}}$ and $\boldsymbol{\omega}_t^{\mathcal{T}}$, in body coordinates. We also append the joint positions, joint velocities, and the actions at the previous two time-steps. All

TABLE I
WEIGHTED REWARD TERMS

Name	Reward Term	Weight
	Imitation	
Torso position xy	$\exp\left(-200.0 \cdot \|\boldsymbol{p}_{x,y} - \hat{\boldsymbol{p}}_{x,y}\|^2\right)$	1.0
Torso orientation	$\exp\left(-20.0 \cdot \|\boldsymbol{\theta} \boxminus \hat{\boldsymbol{\theta}}\|^2\right)$	1.0
Linear velocity xy	$\exp\left(-8.0 \cdot \|\boldsymbol{v}_{x,y} - \hat{\boldsymbol{v}}_{x,y}\|^2\right)$	1.0
Linear velocity z	$\exp\left(-8.0 \cdot (v_z - \hat{v}_z)^2\right)$	1.0
Angular velocity xy	$\exp\left(-2.0 \cdot \|\boldsymbol{\omega}_{x,y} - \hat{\boldsymbol{\omega}}_{x,y}\|^2\right)$	0.5
Angular velocity z	$\exp\left(-2.0 \cdot (\omega_z - \hat{\omega}_z)^2\right)$	0.5
Leg joint positions	$-\|\boldsymbol{q}_l - \hat{\boldsymbol{q}}_l\|^2$	15.0
Neck joint positions	$-\|\boldsymbol{q}_n - \hat{\boldsymbol{q}}_n\|^2$	100.0
Leg joint velocities	$-\|\dot{\boldsymbol{q}}_l - \hat{\dot{\boldsymbol{q}}}_l\|^2$	$1.0 \cdot 10^{-3}$
Neck joint velocities	$-\|\dot{\boldsymbol{q}}_n - \hat{\dot{\boldsymbol{q}}}_n\|^2$	1.0
Contact	$\sum_{i \in \{L,R\}} \mathrm{I}\left[c_i = \hat{c}_i\right]$	1.0
	Regularization	
Joint torques	$-\|\boldsymbol{\tau}\|^2$	$1.0 \cdot 10^{-3}$
Joint accelerations	$-\|\ddot{\boldsymbol{q}}\|^2$	$2.5 \cdot 10^{-6}$
Leg action rate	$-\|\boldsymbol{a}_l - \boldsymbol{a}_{t-1,l}\|^2$	1.5
Neck action rate	$-\|\boldsymbol{a}_n - \boldsymbol{a}_{t-1,n}\|^2$	5.0
Leg action acc.	$-\|\boldsymbol{a}_l - 2\boldsymbol{a}_{t-1,l} + \boldsymbol{a}_{t-2,l}\|^2$	0.45
Neck action acc.	$-\|\boldsymbol{a}_n - 2\boldsymbol{a}_{t-1,n} + \boldsymbol{a}_{t-2,n}\|^2$	5.0
	Survival	
Survival	1.0	20.0

policies are trained with PPO [38]. The policy architectures and additional RL details are described in App. A.

D. Low-level Control

The policy outputs actions at a rate of 50 Hz, while our actuator communication operates at 600 Hz. To bridge this gap, we perform a first-order-hold, i.e. a linear interpolation of the previous and current policy action, followed by a low-pass filter with a cut-off frequency of 37.5 Hz. The low-level controller also implements the path frame dynamics and phase signal propagation described in Sec. V-A. These low-level control aspects are identically implemented in the RL and runtime environments.

E. Simulation

From the CAD model of the robot, we derive a simulation model that accurately describes the physics of the robot, its actuators, and the interaction of the robot with the environment. The rigid body dynamics of the robot is simulated with Isaac Gym [26]. To accurately describe its full dynamics, we add custom actuator models [17, 42]. The employed models are derived from first principles with parameters obtained from system identification experiments of the individual actuators (see App. B). We randomize the parameters of our actuator models within their experimentally observed range. We also add noise to the state that the policy receives, and randomize mass properties and frictional coefficients. In addition to this domain randomization, we apply random disturbance forces and torques on the torso, head, hips, and feet of the robot. During training our walking policy, we additionally randomize the terrain.

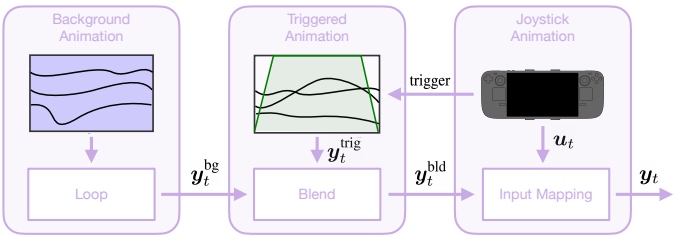

Fig. 5. The animation engine procedurally generates the animation command, $\boldsymbol{y}_t$, based on three layers: background animation, triggered animations, and animations derived from joystick inputs. A triggered animation is blended in and out as illustrated by the green curve. In contrast, the background animation remains continuously active.

TABLE II
SHOW FUNCTION PARAMETERS

Function Parameters	Dimensionality	Units
Antenna positions	2×1	[rad]
Eye colors	2×3	[RGB]
Eye radii	2×1	[%]
Head lamp brightness	1	[%]

VI. RUNTIME

After offline training, the weights of the neural control policies are frozen and the policy networks are deployed onto the onboard computer of the robot. Instead of interacting with a simulator, the deployed policies and low-level controllers interface with the robot hardware and a standard estimator of the state $\mathbf{s}_t$, which fuses IMU and actuator measurements [11].

The proposed runtime system (compare with Fig. 2) enables an operator to puppeteer the character using an intuitive remote control interface. The *Animation Engine* maps the associated puppeteering commands (including policy switching, triggered animation events, and joystick input) to policy control commands, show function signals, and audio signals. A complete list of puppeteering commands is provided in App. C.

We differentiate between artist-specified motions that are used in imitation objectives during RL training, and artist-defined animations that are part of an animation library that a puppeteer interfaces with during runtime. The output of the animation engine is an animation target state for a robot that we then use to form control inputs, $\boldsymbol{g}_t^{\text{perp}}$ and $\boldsymbol{g}_t^{\text{peri}}$, for our policies.

A. Perpetual & Periodic Motions

During standing and walking, show function and policy commands are computed by combining event-driven animation playback with live puppeteering. To procedurally generate animation states, we define the robot configuration, $\boldsymbol{c}_t = (\boldsymbol{p}_t^{\mathcal{P}}, \boldsymbol{\theta}_t^{\mathcal{P}}, \boldsymbol{q}_t)$, from which we extract control inputs. We also define an extended animation command, $\boldsymbol{y}_t = (\boldsymbol{\nu}_t, \boldsymbol{c}_t)$, where $\boldsymbol{\nu}_t$ represents all show function commands as summarized in Tab. II. Fig. 5 provides a high-level diagram of the proposed animation pipeline which computes a target output, $\boldsymbol{y}_t$, by combining three functional animation layers:

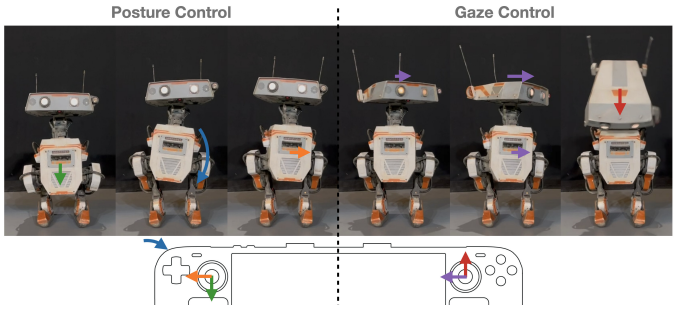

Fig. 6. A selection of joystick commands during standing. Posture control (left), moves the torso without affecting the gaze. Gaze control (right), changes the gaze primarily by moving the head, but also commands additive torso movement to extend the range. See Appendix C for details.

1) Background Animation: This layer relies on looped playback of a periodic background animation, $\boldsymbol{y}_t^{\text{bg}}$, that is always visible in the absence of additional input. The background animation conveys a basic level of activity that includes intermittent eye-blinking and antenna motion.

2) Triggered Animations: This layer blends operator-triggered animation clips on top of the background animation. We select them from a library of artist-specified clips and map them to buttons on the remote control. For our specific character, they range from simple *yes-no* animations to complex *scan* clips. Representing the current state of a triggered animation with $\boldsymbol{y}_t^{\text{trig}}$, we blend the background and triggered targets,

$$\boldsymbol{\nu}_t^{\text{bld}} = (1-\beta)\boldsymbol{\nu}_t^{\text{bg}} + \beta\boldsymbol{\nu}_t^{\text{trig}} \tag{9}$$

$$\boldsymbol{c}_t^{\text{bld}} = \text{interp}(\boldsymbol{c}_t^{\text{bg}}, \boldsymbol{c}_t^{\text{trig}}, \alpha), \tag{10}$$

where the configuration interpolation is linear for position and joint angles, and uses *slerp* for body orientation. β and α are blend ratios that vary as a function of playback time. Both ratios ramp linearly from 0 to 1 for a given duration at the beginning of an animation and back to 0 over the same duration prior to the end of the animation. We use the durations $T_\beta = 0.1\,\text{s}$ and $T_\alpha = 0.35\,\text{s}$, such that the facial expressions associated with the show functions naturally blend faster than the body animation.

3) Joystick Animation. The final layer transforms the blended animation state, $\boldsymbol{y}_t^{\text{bld}}$, based on joystick input from the puppeteer. Let $\boldsymbol{u}$ represent the joystick axes, triggers, and button modifiers streamed from the controller. While standing, the target robot configuration is computed as

$$\boldsymbol{y}_t = \mathcal{J}^{\text{perp}}\left(\boldsymbol{y}_t^{\text{bld}}, \boldsymbol{u}_t\right), \tag{11}$$

where $\mathcal{J}^{\text{perp}}$ is a non-linear mapping that modifies the current animation state based on the commanded inputs. The joystick axes are mapped to additive offsets to the animated head and torso pose contained in $\boldsymbol{c}_t^{\text{bld}}$, modifying the robot's gaze and posture while idling or executing a triggered animation. Illustrative examples are provided in Fig. 6.

While walking, the target robot configuration is computed using a similar mapping,

$$(\boldsymbol{y}_t, \boldsymbol{v}_t^{\mathcal{P}}, \omega_t^{\mathcal{P}}) = \mathcal{J}^{\text{peri}}\left(\boldsymbol{y}_t^{\text{bld}}, \boldsymbol{u}_t\right), \tag{12}$$

that additionally produces path velocity commands for the periodic policy. For ease of use, we maintain identical gaze controls; however, the joystick axes used for posture control are remapped to forward, lateral, and turning velocity commands during walking. We also introduce show function modulation based on the path velocity. As the robot reaches top speed, the antennas duck back and the eye radii narrow to convey exertion during fast walking.

Once the animation output, $\boldsymbol{y}_t$, has been computed, the show functions are controlled directly with $\boldsymbol{\nu}_t$, while the policy command signals are derived from $\boldsymbol{c}_t$. For the head, we compare $\boldsymbol{c}_t$ to the nominal configuration of the robot and extract the relative head commands Δh_t^{head} and $\Delta\boldsymbol{\theta}_t^{\text{head}}$. During standing, the torso height and orientation in $\boldsymbol{g}_t^{\text{perp}}$ are extracted directly from the target configuration. While walking, the lower body motion is determined entirely by the periodic policy and commanded path velocities. Note that in both cases, we ignore the leg joint positions, which are not part of the policy inputs.

B. Episodic Motions

When an episodic motion is triggered, the animation engine initiates a transition to the corresponding policy and triggers an associated animation clip that syncs the appropriate show function animation, similar to the triggered animation layer during perpetual and periodic motions. There is no additional user input until the episodic motion finishes.

C. Audio Engine

An onboard audio engine processes and mixes all audio on the robot. A message-based interface enables the operator to trigger short sound clips, e.g. vocalizations, at any time while puppeteering. When an animation or episodic motion has associated audio, the animation engine relays a synchronous playback command to the audio engine. We also support sound effects which are modulated by robot actuator speeds, for creating artificial gear sounds.

VII. Results

We first evaluate the performance and robustness of the individual control policies on which the rest of the control stack builds. Afterward, we show how the animation engine translates user commands into policy control signals and ultimately turns the technical capabilities of the system into a compelling character. Finally, we demonstrate and discuss the deployment of the full system.

A. Evaluating Control Policies

Standing: Each policy input corresponds to the range of motion of one controllable dimension, e.g. the torso yaw. This enables expressive motion during standing, including direct control of the torso. The accompanying video shows the robot going through the full range of each policy input.

Walking: To evaluate walking performance, we demonstrate that the system accurately tracks commanded walking velocities. The walking style has a maximum longitudinal velocity

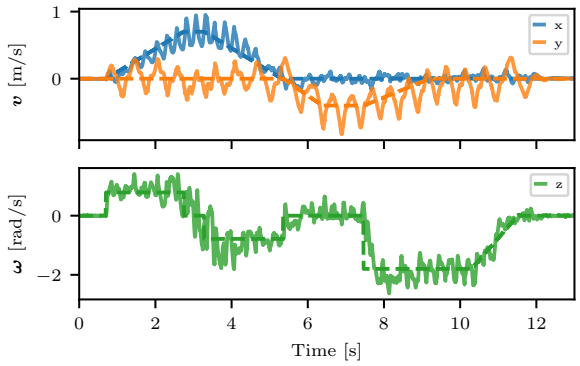

Fig. 7. Commanded path velocities (dashed) and measured torso velocities (solid) in path frame.

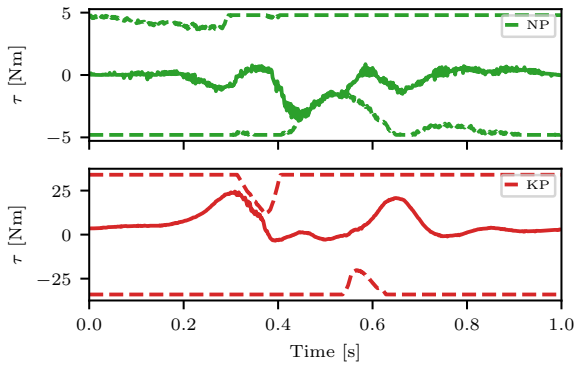

Fig. 8. Measured joint torques (solid) and velocity-dependent torque limits (dashed) computed by the actuator model for the measured joint velocities, during the episodic "jump" motion. The top plot shows the neck pitch (NP) actuator and the bottom plot shows the left knee pitch (KP) actuator.

of $0.7\,\mathrm{m\,s^{-1}}$, lateral velocity of $0.4\,\mathrm{m\,s^{-1}}$, and turning rate of $1.8\,\mathrm{rad\,s^{-1}}$. Fig. 7 shows the estimated torso velocity of the robot in path frame compared to the commanded walking velocities. The robot is responsive and follows all commands closely.

Episodic Policies: In the video we show several examples of episodic motions, namely a "happy dance", "excited motion", "jump", and "tantrum". They demonstrate a diverse set of motions and have a high level of coordination between joints, at a level best achieved with a specialized policy. Fig. 8 shows the torque of both the neck pitch and left knee pitch joints during the jump motion, together with their velocity-depended torque limits as predicted by the actuator model. It can be seen that while pushing off, the robot reaches the torque limits of the knees, which rapidly decrease due to increasing joint velocities. During the jump, the robot makes an upward pitching movement with the head, causing the actuator in the head to also reach its limit. Tracking errors for these and motions from previous sections are reported in Tab. III.

Robustness: We demonstrate the robot's ability to stabilize under challenging conditions by subjecting it to external pushes and by walking over small obstacles (see video). The policy deviates from the reference trajectory and contact schedule to recover and maintain balance. This highlights the strength of using an RL approach. In contrast, optimization-based approaches struggle to simultaneously plan motion and contact schedules in real-time, and are often constrained to follow the contact reference [50].

TABLE III
MEAN ABSOLUTE TRACKING ERROR (MAE) OF JOINT POSITIONS

Type	Name	MAE [rad]
Perpetual	Standing	0.035
Periodic	Walking	0.123
Episodic	Excited Motion	0.029
	Happy Dance	0.027
	Jump	0.043
	Tantrum	0.032

Policy Transitions: Fig. 9 provides insight into the policy transitions for a short sequence shown in the video. The plots are colored according the active policy. The top two plots show the policy actions, i.e. joint position setpoints, for all neck joints and left leg joints. Because each policy receives the two previous actions and smoothness is encouraged during training, the actions remain continuous and the switch of policies is virtually undetectable for an outside observer.

While it is possible to transition in and out of the walking gait at any point, the transition looks more natural when done at the appropriate, non-zero phase within the gait cycle. When starting to walk, we initialize the phase based on the commanded turning direction: when turning left, we start the phase at a left step, and vice-versa for a right turn. When transitioning out of walk (see dashed line in Fig. 9), the policy switch is delayed until the start of the next double support phase. This way the walking policy finishes the swing phase as intended and the standing policy starts with both feet already on the ground.

B. Alternative RL Formulations

We compare our RL formulation against related alternatives in the literature. First, we compare against an approach that directly tracks the commanded walking velocities with the torso [17]. Specifically, we use the commands as reference in the torso related rewards and set the leg joint position, leg joint velocity, and contact weights to zero. The phase signal is removed from the policy inputs. As shown in the video, this results in a policy that rapidly shuffles the feet. *Foot clearance* and *slip penalties* are commonly used to suppress this behavior [28]. However, they are tedious to tune and the resulting gait remains far from natural. Second, we compare against approaches that provide the policy with a phase signal and corresponding contact reference [39, 40]. We add back the phase signal to the inputs and activate the contact reward. As shown in the video, the resulting policy follows the stepping pattern well. However, the effect of directly tracking the velocity commands remains visible, resulting in a stiff and upright torso motion. Finally, we evaluated using the current and future kinematic reference poses as the policy inputs in place of the phase signal [33, 25], with the rest of the formulation remaining identical to ours. For both the walking

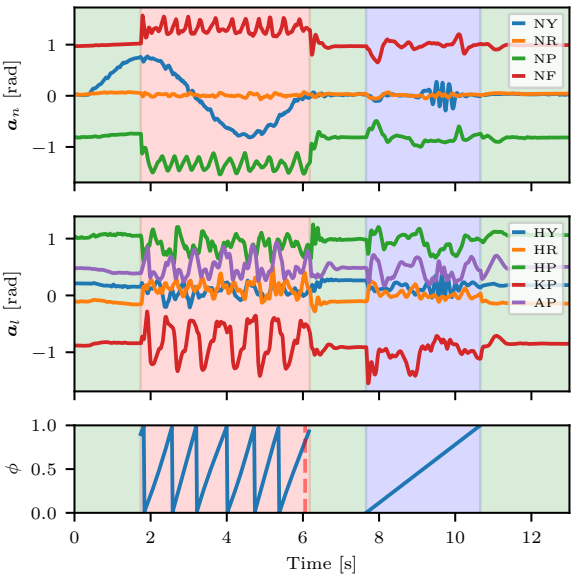

Fig. 9. Policy actions across policy transitions during a short motion sequence. The active policy is indicated with the background color of the plot (green: standing, red: walking, blue: episodic). The top plot shows the actions for the neck joints: neck yaw (NY), neck roll (NR), neck pitch (NP), and neck forward (NF). The middle plot shows actions form the left leg: hip yaw (HY), hip roll (HR), hip pitch (HP), knee pitch (KP), and ankle pitch (AP). The bottom plot shows the phase signal when applicable. The dashed red line shows the moment a request to transition from walking to standing is received. The transition is made when the gait reaches the next double support phase.

and episodic motions this formulation converges to the same reward and produces a visually identical motion. This is not surprising: the phase signal and kinematic reference contain the same information. The benefit of using a phase signal is that we do not need to store and reproduce the reference motions on the robot.

C. Animation Engine

In the accompanying video, we show the build up of the individual layers of the animation engine. First showing only the background animation, then adding the triggered animation layer, and finally adding the joystick commands. The combination of triggered animation content and joystick-driven targeting enables simple, intuitive, and expressive puppeteering of the robotic character. With direct teleoperation, the operator can deliver a wide range of performances adapted to the robot's environment. Meanwhile, the operator can trigger animation clips that enable expressions and interactions with subtle or complex movement that cannot be achieved through joystick control alone. As an example, the puppeteer can direct the robot's gaze toward a person posing a question, then trigger a stylized *yes* or *no* animation while simultaneously maintaining eye contact and modifying the body posture. Fig. 10 includes stills from the video demonstrating the effectiveness of the proposed approach for puppeteering the robot to act out a short scene and interact with a human.

Gaze and Posture Control: For the remote control interface, we selected a two-joystick device with a layout familiar to operators with video game experience. Designing an appropriate joystick input mapping is challenging due to the high dimensionality of the robot configuration and limited joystick axes. To reduce cognitive load on the puppeteer, the joystick controls were designed to provide independent control of the robot's gaze and body posture during standing. As illustrated in Fig. 6, the left joystick modifies the body posture (torso yaw and pitch), while counter-rotating the head to maintain a fixed gaze. Meanwhile, the right joystick modifies the robot's gaze (head yaw and pitch), while introducing additive rotation of the torso when the neck reaches it's kinematic limits. When the operator directs the robot's gaze to an extreme angle, either yawing to the side or pitching towards it's feet, the body naturally follows the head. The video results demonstrate each control independently, followed by a combined puppeteering sequence. The functional separation allows the puppeteer to more easily direct the robot's line of sight while simultaneously modifying the body posture to convey emotion. The detailed interface described in Appendix C was continuously refined throughout the project, including the training of several operators without a background in robotics.

D. System Deployment

At the time of writing, we have conducted several public deployments of the robot with up to three robots simultaneously, resulting in the order of 10 h of robot runtime without a single fall. Audiences quickly become captivated by the robotic character and often will not notice the presence of the puppeteer. However, some feedback has also called out the presence of a puppeteer as a distraction or reducing believability of the character. Frequently bystanders assume the robot can perceive its environment: *"can the robot really see me?"*, or *"how does it know what I'm saying?"*.

VIII. Conclusion

In this work we propose a robot design and control workflow that targets the intricate challenges associated with legged robots for entertainment applications. We present a new bipedal robotic character and demonstrate the integration of expressive, artist-directed motions with robust dynamic mobility. Multiple RL policies, trained to imitate artistic motions, and conditioned on low-dimensional input signals, provide a robust foundation on which we build in the animation engine. Together, they allow for real-time show performances through an intuitive operator interface.

Our work has also demonstrated that it is possible to build dynamic legged robots where the kinematics and the mechanical design are driven by a creative target rather than by functional requirements. Taken together with the general formulation of the presented pipeline, our work enables the creation of expressive robot characters outside the typical anthropomorphic or zoomorphic morphologies, paving the way towards more general and fantastical robotic characters.

While the separation into multiple policies has provided us with precise control over the behavior of the robot, it results in training overhead especially when scaling up the amount

Fig. 10. (Top) The operator uses the proposed puppeteering interface to act out a scene where the robot discovers a paper roll and decides to kick it under the bench. We make use of the head lamp, triggered head animations, and gaze controls to convey initial curiosity. Episodic policies are triggered following the run-up and kick to convey excitement and celebration. (Bottom) The operator puppeteers the robot to interact with a human. The robot is at first shy, then approaches, extends its head, and is rewarded with a head scratch.

of episodic motions. Here, we are eager to explore if a single policy can learn several skills with the same level of accuracy as the specialized policies. Finally, there is a natural limit to how many buttons a puppeteer can use effectively. To further expand the expressive capabilities of the character, we see an opportunity for embedding autonomy in the animation engine.

References

[1] Evan Ackerman. How Boston Dynamics taught its robots to dance. *IEEE Spectrum*, Jan. 7 2021. URL https://spectrum.ieee.org/how-boston-dynamics-taught-its-robots-to-dance.

[2] Autodesk, INC. Maya, 2023. URL https://autodesk.com/maya.

[3] Thomas Bi, Péter Fankhauser, Dario Bellicoso, and Marco Hutter. Real-time dance generation to music for a legged robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[4] Marko Bjelonic, Ruben Grandia, Moritz Geilinger, Oliver Harley, Vivian S Medeiros, Vuk Pajovic, Edo Jelavic, Stelian Coros, and Marco Hutter. Offline motion libraries and online mpc for advanced mobility skills. *The International Journal of Robotics Research*, 41(9-10): 903–924, 2022.

[5] Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. Adversarial motion priors make good substitutes for complex reward functions. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 25–32. IEEE, 2022.

[6] M. Fujita, Y. Kuroki, T. Ishida, and T.T. Doi. Autonomous behavior control architecture of entertainment humanoid robot SDR-4X. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 960–967 vol.1, 2003. doi: 10.1109/IROS.2003.1250752.

[7] Masahiro Fujita. Aibo: Toward the era of digital creatures. *The International Journal of Robotics Research*, 20(10):781–794, 2001.

[8] Masahiro Fujita and Koji Kageyama. An open architecture for robot entertainment. In *Proceedings of the first international conference on Autonomous agents*, pages 435–442, 1997.

[9] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of NAO humanoid. In *2009 IEEE international conference on robotics and automation*, pages 769–774. IEEE, 2009.

[10] Ruben Grandia, Farbod Farshidian, Espen Knoop, Christian Schumacher, Marco Hutter, and Moritz Bächer. Doc: Differentiable optimal control for retargeting motions onto legged robots. *ACM Trans. Graph.*, 42(4), jul 2023. ISSN 0730-0301.

[11] Ross Hartley, Maani Ghaffari, Ryan M Eustice, and Jessy W Grizzle. Contact-aided invariant extended kalman filtering for robot state estimation. *The International Journal of Robotics Research*, 39(4):402–430, 2020. doi: 10.1177/0278364919894385.

[12] Leonard Hasenclever, Fabio Pardo, Raia Hadsell, Nicolas Heess, and Josh Merel. CoMic: Complementary task learning & mimicry for reusable skills. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4105–4115. PMLR, 13–18 Jul 2020.

[13] David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for

quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024.

[14] Michael A. Hopkins, Georg Wiedebach, Kyle Cesare, Jared Bishop, Espen Knoop, and Moritz Bächer. Interactive design of stylized walking gaits for robotic characters. *ACM Transactions On Graphics (TOG)*, 2024.

[15] Jonathan Hurst. Walk this way: To be useful around people, robots need to learn how to move like we do. *IEEE Spectrum*, 56(3):30–51, 2019. doi: 10.1109/MSPEC.2019.8651932.

[16] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C. Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, Remo Diethelm, Samuel Bachmann, Amir Melzer, and Mark Hoepflinger. ANYmal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44, 2016. doi: 10.1109/IROS.2016.7758092.

[17] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[18] Hiroshi Ishiguro, Tetsuo Ono, Michita Imai, Takeshi Maeda, Takayuki Kanda, and Ryohei Nakatsu. Robovie: an interactive humanoid robot. *Industrial robot: An international journal*, 28(6):498–504, 2001.

[19] Dongho Kang, Jin Cheng, Miguel Zamora, Fatemeh Zargarbashi, and Stelian Coros. Rl + model-based control: Using on-demand optimal control to learn versatile legged locomotion. *IEEE Robotics and Automation Letters*, 8(10):6619–6626, 2023. doi: 10.1109/LRA.2023.3307008.

[20] Jan Kedzierski, Robert Muszyński, Carsten Zoll, Adam Oleksy, and Mirela Frontkiewicz. Emys—emotive head of a social robot. *International Journal of Social Robotics*, 5:237–249, 2013.

[21] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas De Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt. Design of a momentum-based control framework and application to the humanoid robot atlas. *International Journal of Humanoid Robotics*, 13 (01):1650007, 2016.

[22] Christian Kroos, Damith C Herath, and Stelarc. Evoking agency: attention model and behavior control in a robotic art installation. *Leonardo*, 45(5):401–407, 2012.

[23] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302104.

[24] Zhongyu Li, Christine Cummings, and Koushil Sreenath. Animated cassie: A dynamic relatable robotic character. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3739–3746. IEEE, 2020.

[25] Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2811–2817. IEEE, 2021.

[26] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu based physics simulation for robot learning. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021.

[27] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes Von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, et al. The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural networks*, 23(8-9):1125–1134, 2010.

[28] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), 2022. doi: 10.1126/scirobotics.abk2822.

[29] Franck Multon, Laure France, Marie-Paule Cani-Gascuel, and Giles Debunne. Computer animation of human walking: a survey. *The journal of visualization and computer animation*, 10(1):39–54, 1999.

[30] Shin'ichiro Nakaoka, Shuuji Kajita, and Kazuhito Yokoi. Intuitive and flexible user interface for creating whole body motions of biped humanoid robots. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1675–1682. IEEE, 2010.

[31] Amit Kumar Pandey and Rodolphe Gelin. A mass-produced sociable humanoid robot: Pepper: The first machine of its kind. *IEEE Robotics & Automation Magazine*, 25(3):40–48, 2018. doi: 10.1109/MRA.2018.2833157.

[32] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

[33] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *Robotics: Science and Systems*, 2020.

[34] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

[35] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 46–51. IEEE, 2009.

[36] Hayley Robinson, Bruce MacDonald, Ngaire Kerse, and Elizabeth Broadbent. The psychosocial effects of a companion robot: a randomized controlled trial. *Journal of the American Medical Directors Association*, 14(9): 661–667, 2013.

[37] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[39] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7309–7315. IEEE, 2021.

[40] Jonah Siekmann, Kevin Green, John Warila, Alan Fern, and Jonathan Hurst. Blind bipedal stair traversal via sim-to-real reinforcement learning. *Robotics: Science and Systems XVII*, 2021.

[41] Wael Suleiman, Eiichi Yoshida, Fumio Kanehiro, Jean-Paul Laumond, and André Monin. On human motion imitation by humanoid robot. In *2008 IEEE International conference on robotics and automation*, pages 2697–2704. IEEE, 2008.

[42] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.010.

[43] Valve Corporation. Steam deck, 2023. URL https://www.steamdeck.com/.

[44] Albert J.N. van Breemen. Animation engine for believable interactive user-interface robots. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2873–2878. IEEE, 2004.

[45] Albert J.N. van Breemen. Bringing robots to life: Applying principles of animation to robots. In *Proceedings of Shapping Human-Robot Interaction workshop held at CHI*, volume 2004, pages 143–144. Citeseer, 2004.

[46] Rik van den Brule, Ron Dotsch, Gijsbert Bijlstra, Daniel HJ Wigboldus, and Pim Haselager. Do robot performance and behavioral style affect human trust? a multi-method approach. *International journal of social robotics*, 6:519–531, 2014.

[47] Gentiane Venture and Dana Kulić. Robot expressive motions: A survey of generation and evaluation methods. *J. Hum.-Robot Interact.*, 8(4), nov 2019. doi: 10.1145/3344286.

[48] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018. ISSN 0957-4158. doi: https://doi.org/10.1016/j.mechatronics.2018.02.009.

[49] Patrick M. Wensing, Albert Wang, Sangok Seok, David Otten, Jeffrey Lang, and Sangbae Kim. Proprioceptive actuator design in the mit cheetah: Impact mitigation and high-bandwidth physical interaction for dynamic legged robots. *IEEE Transactions on Robotics*, 33(3):509–522, 2017.

[50] Patrick M Wensing, Michael Posa, Yue Hu, Adrien Escande, Nicolas Mansard, and Andrea Del Prete. Optimization-based control for dynamic legged robots. *IEEE Transactions on Robotics*, 2023.

[51] Pierre-brice Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 137–142, 2006. doi: 10.1109/ICHR.2006.321375.

APPENDIX A
POLICY TRAINING

Each policy is trained for $100\,000$ iterations, which corresponds to a computation time of around 2 days on a Nvidia RTX 4090. Despite the long training time for the final policy, we note that most of the behavior is learned in the first few iterations. In our experience, similar to related work [37], 1500 iterations or $30\,\mathrm{min}$ are enough to preview the nominal behavior. We use the PPO hyperparameters shown in Tab. IV, with and adaptive learning rate [37]. For all policies, we use three fully connected layers of 512 hidden units and ELU activation functions. For the critic, we use a separate network with the same amount of hidden units. In addition to the policy inputs, the critic receives the simulation state without noise and the randomized friction parameter as privileged information.

Moreover, we apply fixed transformations, i.e. without learnable parameters, on both the input and output side of the policy. We first normalize all inputs by their expected range. For policies that take phase as an input, we replace the phase with a feature vector derived from the phase. For walking, we add the first two harmonics $(\sin(k \cdot 2\pi\phi), \cos(k \cdot 2\pi\phi))$, $k \in \{1, 2\}$. The first harmonic corresponds to the walking cycle and the second harmonic makes it easier for the policy to learn the head-bob which occurs at twice the gait frequency. For the episodic motions, we use $N = 50$ Gaussian basis functions, $\exp\left(-(\phi - \phi_i)^2/(2N^2)\right)$, where the ϕ_is are equally spaced between 0 and 1. These features are highly local in time and allow the policy to form a rich phase-dependent feed-forward signal. At the output, we apply a linear transform to the actions such that 0 maps to the nominal joint positions of the robot and 1 to the expected range per joint. Finally, joint position setpoints are clipped to a maximum deviation around the measured joint position. This deviation is chosen large enough such that the maximum actuator torque can still be produced.

We use the same rewards listed in Tab. I for all policies, with small modifications for the episodic motions "excited motion" and "jump". For both motions, we emphasize a key aspect of

TABLE IV
RL HYPERPARAMETERS

Param.	**Value**
Num. iterations	100 000
Batch size (envs. × steps)	8192×24
Num. mini-batches	4
Num. epochs	5
Clip range	0.2
Entropy coefficient	0.0
Discount factor	0.99
GAE discount factor	0.95
Desired KL-divergence	0.01
Max gradient norm	1.0

the motion by increasing selected weights between ϕ_{start} and ϕ_{end},

$$\tilde{w}(\phi) = w_0 + \mathrm{I}\left[\phi_{\text{start}} < \phi < \phi_{\text{end}}\right] \cdot w_{\text{extra}}, \tag{13}$$

where $\mathrm{I}[\cdot]$ is an indicator function. For the "excited motion", we increase the weight on angular velocity tracking during the part where the robot rapidly shakes its torso. For the "jump", we add a reward for torso height and add extra weight on the torso orientation during the jump. Additionally, we increase the contact reward during the jump. Without this weight adjustment, the policy tends to cheat by keeping the toes on the ground.

Details for the applied disturbance forces are given in Tab. V. All parameters refer to the minimum and maximum magnitude drawn from a uniform distribution. For each specified body, a random force and torque drawn from a uniform distribution per dimension is applied for a random "on" duration. Afterward, a random "off" duration is selected until the next disturbance is applied. We organize the parameters in three separate categories, but the process is applied independently per body. The forces are gradually introduced according to a linear curriculum over the first 1500 iterations.

TABLE V
DISTURBANCE PARAMETERS

Param.		Short / small	Long / small	Short / large
Body		Hips, Feet	Pelvis, Head	Pelvis
Force [N]	XY	[0.0, 5.0]	[0.0, 5.0]	[90.0, 150.0]
	Z	[0.0, 5.0]	[0.0, 5.0]	[0.0, 10.0]
Torque [N m]	XY	[0.0, 0.25]	[0.0, 0.25]	[0.0, 15.0]
	Z	[0.0, 0.25]	[0.0, 0.25]	[0.0, 15.0]
Duration [s]	On	[0.25, 2.0]	[2.0, 10.0]	[0.1, 0.1]
	Off	[1.0, 3.0]	[1.0, 3.0]	[12.0, 15.0]

APPENDIX B
ACTUATOR MODEL

We describe the actuator models that we use to augment the physics simulator. System identification was performed by mounting single actuators on a test-bench that measures output torque. All identified parameters are provided in Tab. VI.

The proportional derivative motor torque equation for the quasi-direct drives used in this work is computed through,

$$\tau_m = k_{\text{P}}(a - \tilde{q}) - k_{\text{D}}\dot{q}, \tag{14}$$

where k_{P} and k_{D} are gains, a is the joint setpoint, $\dot{q}$ is the joint velocity, and $\tilde{q}$ is the joint position with an added encoder offset, ϵ_q,

$$\tilde{q} = q + \epsilon_q. \tag{15}$$

This encoder offset is drawn from a uniform distribution, $\mathcal{U}(-\epsilon_{q,\max}, \epsilon_{q,\max})$, at the beginning of each RL episode and accounts for inaccuracy in our joint calibration.

The actuators used in the head do not implement the same PD equation and instead use,

$$\tau_m = k_{\text{P}}(a - \tilde{q}) + k_{\text{D}}\frac{\mathrm{d}}{\mathrm{d}t}(a - \tilde{q}), \tag{16}$$

where the derivative term operates on the numerical differentiation of the setpoint error.

Friction in the joint is modelled as,

$$\tau_f = \mu_s \tanh\left(\dot{q}/\dot{q}_s\right) + \mu_d\dot{q}, \tag{17}$$

where μ_s is the static coefficient of friction, with activation parameter $\dot{q}_s$, and μ_d is the dynamic coefficient of friction.

The torque produced at the joint can then be computed by applying torque limits to τ_m and subtracting friction forces,

$$\tau = \text{clamp}_{[\underline{\tau}(\dot{q}), \overline{\tau}(\dot{q})]}\left(\tau_m\right) - \tau_f, \tag{18}$$

where $\underline{\tau}(\dot{q})$, and $\overline{\tau}(\dot{q})$, are the velocity-dependent minimum and maximum torques of the motor. These limits consists of a constant limit torque for braking and low velocities, $\tau_{\max}$, and a linear limit ramping down the available torque above an identified velocity, $\dot{q}_{\tau_{\max}}$. The linear limit crosses the velocity, $\dot{q}_{\max}$, when the limit torque is $0\,\mathrm{N\,m}$.

The measured joint position reported by the actuator model contains the encoder offset, a term that models backlash and noise,

$$\hat{q} = \tilde{q} + 0.5 \cdot b \cdot \tanh\left(\tau_m/\tau_b\right) + \mathcal{N}\left(0, \sigma_q^2\right), \tag{19}$$

where b is the total motor backlash with activation parameter τ_b. b is drawn from a uniform distribution, $\mathcal{U}(b_{\min}, b_{\max})$, at the start of each episode. This term models the effect that a motor moves to the side of the backlash in the same direction as the applied motor torque. For the noise model, we fit a normal distribution with the standard deviation proportional to the motor rotating velocity

$$\sigma_q = \sigma_{q,0} + \sigma_{q,1}|\dot{q}|. \tag{20}$$

Finally, the reflected inertia of the actuator, I_m, is added to the simulation model. In Isaac Gym, this is done by setting the *armature* value of the corresponding joint. This value is randomized up to a $20\,\%$ offset at the start of each episode.

APPENDIX C
PUPPETEERING INTERFACE

The button layout of the remote controller used in this work is shown and annotated in Fig. 12. The effect of each button is described in Tab. VII. The simple touchscreen buttons turn on and off the robot. During operation, the screen displays vital system information: battery voltage, motor temperatures, and connection stability.

TABLE VI
ACTUATOR GAINS AND MODEL PARAMETERS

Param.	Unitree A1	Unitree Go1	Dynamixel XH540-V150	Units
k_P	15.0	10.0	5.0	$[\mathrm{N\,m\,rad^{-1}}]$
k_D	0.6	0.3	0.2	$[\mathrm{N\,m\,s\,rad^{-1}}]$
τ_max	34.0	23.7	4.8	$[\mathrm{N\,m}]$
$\dot{q}_{\tau_\mathrm{max}}$	7.4	10.6	0.2	$[\mathrm{rad\,s^{-1}}]$
$\dot{q}_\mathrm{max}$	20.0	28.8	7.0	$[\mathrm{rad\,s^{-1}}]$
μ_s	0.45	0.15	0.05	$[\mathrm{N\,m}]$
μ_d	0.023	0.016	0.009	$[\mathrm{N\,m\,s\,rad^{-1}}]$
b_min	0.005	0.002	0.002	[rad]
b_max	0.015	0.005	0.005	[rad]
$\epsilon_{q,\mathrm{max}}$	0.02	0.02	0.02	[rad]
$\sigma_{q,0}$	$1.80 \cdot 10^{-4}$	$1.89 \cdot 10^{-4}$	$4.31 \cdot 10^{-4}$	[rad]
$\sigma_{q,1}$	$3.61 \cdot 10^{-5}$	$5.47 \cdot 10^{-5}$	$2.43 \cdot 10^{-5}$	[s]
I_m	0.011	0.0043	0.0058	$[\mathrm{kg\,m^2}]$

Fig. 11 illustrates how the torso and head yaw offsets are computed as a function of the left and right joystick inputs to achieve independent torso movement with the left joystick, and additive gaze and torso coordination with the right joystick. The same principle applies to the pitch and roll directions as summarized in Tab. VII.

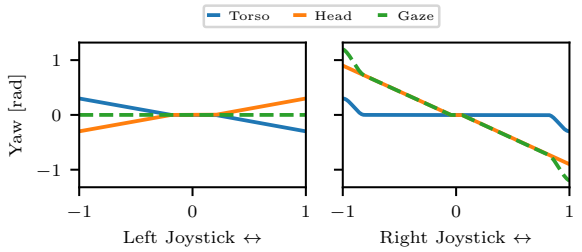

Fig. 11. Joystick mapping during standing for the torso and head yaw offset depending on the left and right joystick inputs. The resulting gaze offset is the sum of the torso and local head offset.

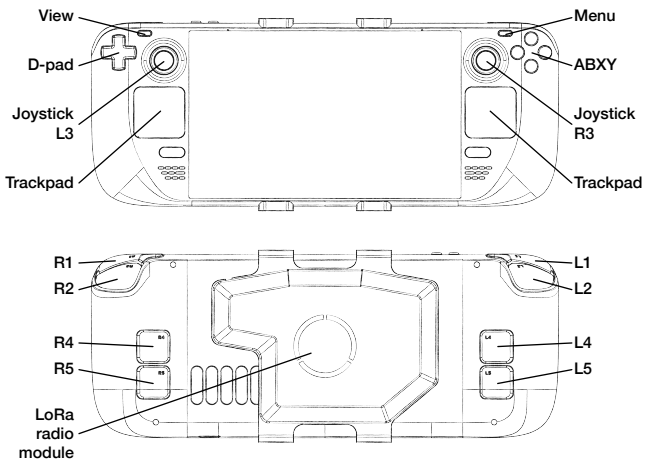

Fig. 12. Steam Deck [43] layout.

TABLE VII
PUPPETEERING BUTTON MAPPING

Button		Effect
Menu		Trigger a safety mode called *motion stop*. This forces a transition to standing and freezes the joint setpoints with high position gains after waiting 0.5 s.
View		Slowly move all joints to the default pose. Only available at startup or while in *motion stop*.
D-pad	↕	Move the head up-down.
	↔	Roll the head left-right.
Left Joystick	↕	During walking: Longitudinal walking velocity. During standing: *Up* pitches the torso forward while the head remains stationary, and *Down* lowers the torso height.
	↔	During walking: Turning rate. During standing: Torso yaw while the head remains stationary.
	L3	Pressing the left joystick triggers a scanning animation.
Right Joystick	↕	Pitches the head. During standing, this additionally commands torso pitch.
	↔	Yaws the head left-right. During standing, the end of the range additionally commands torso yaw.
	R3	Pressing the right joystick toggles the audio level.
ABXY	A	Transition to standing.
	B	Fully tuck the neck in, turn off the eyes, and retract the antennas. While standing, the torso height is also lowered.
	X	Cancel all active animations.
	Y	Turn on the background animation layer.
Left Trackpad		Trigger an episodic motion. Each quadrant of the trackpad maps to a different motion.
Right Trackpad		Like the left trackpad. Reserved to trigger four additional episodic motions.
Backside	L1	Turn the head lamp on and off.
	R1	Single press: Start and stop walking, Hold: increase the walking velocity gain to 100 %. Without holding R1, all velocity commands are scaled to 50 % of the maximum.
	L2 / R2	During walking: Lateral walking velocity. During standing: Roll the torso while the head remains stationary.
	L4	Short press: trigger a happy animation. Long press: trigger an angry animation.
	L5	Short press: trigger an anxious animation. Long press: trigger a curious animation.
	R4	Short press: trigger a “yes” animation. Long press: trigger a “no” animation.
	R5	Trigger an expressive audio clip.