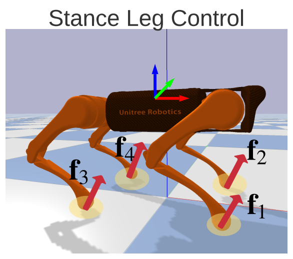
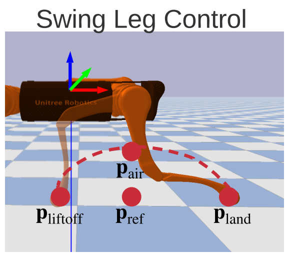
A Details of the Low-Level Convex MPC Controller



(a) The stance controller optimizes ground reaction forces $\boldsymbol{f}_{1,\ldots,4}$ to track a desired base trajectory.



(b) The swing controller tracks the leg on a quadratic curve, which is fitted using $(\boldsymbol{p}_{\text{lift-off}}, \boldsymbol{p}_{\text{air}}, \boldsymbol{p}_{\text{land}})$.

Figure 8: Our low-level convex MPC controller uses different controllers for stance (**left**) and swing (**right**) legs.

A.1 Stance Leg Controller

The stance leg controller optimizes for the ground reaction forces using Model Predictive Control (MPC) (Fig. 8a), where the objective is for the base to track a desired trajectory. The robot is modeled using the simplified centroidal dynamics model. We now describe our setup in detail:

Notation We represent the base pose of the robot in the world frame as $\boldsymbol{x} = [\boldsymbol{\Theta}, \boldsymbol{p}, \boldsymbol{\omega}, \dot{\boldsymbol{p}}] \in \mathbb{R}^{12}$. $\boldsymbol{\Theta} = [\phi, \theta, \psi]$ is the robot's base orientation represented as Z-Y-X Euler angles, where ψ is the yaw, θ is the pitch and ϕ is the roll. $\boldsymbol{p} \in \mathbb{R}^3$ is the Cartesian coordinate of the base position. $\boldsymbol{\omega}$ and $\dot{\boldsymbol{p}}$ are the linear and angular velocity of the base. $\boldsymbol{r}_{\text{foot}} = (\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \boldsymbol{r}_4) \in \mathbb{R}^{12}$ represents the four foot positions relative to the robot base. MPC optimizes for the ground reaction force $\boldsymbol{f}_{1,\ldots,4}$ at each foot, which we denote as $\boldsymbol{u} = (\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3, \boldsymbol{f}_4) \in \mathbb{R}^{12}$. $\mathbf{I}_n$ denotes the $n \times n$ identity matrix. $[\cdot]_\times$ converts a 3d vector into a skew-symmetric matrix, so that for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$, $\boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times \boldsymbol{b}$.

Centroidal Dynamics Model Our centroidal dynamics model is based on [3] with a few modifications. We assume massless legs, and simplify the robot base to a rigid body with mass m and inertia $\mathbf{I}_{\text{base}}$ (in the body frame). The rigid body dynamics in world coordinates are given by:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}_{\text{world}}\boldsymbol{\omega}) = \sum_{i=1}^{4} \boldsymbol{r}_i \times \boldsymbol{f}_i \tag{4}$$

$$\ddot{\boldsymbol{p}} = \frac{\sum_{i=1}^{4} \boldsymbol{f}_i}{m} + \boldsymbol{g} \tag{5}$$

where $\boldsymbol{g} = [0, 0, -9.8]^T$ is the gravity vector. To simplify Eq.4, note that when angular velocity is small, we can omit the centripetal forces and write the left hand side as:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}_{\text{world}}\boldsymbol{\omega}) = \mathbf{I}_{\text{world}}\dot{\boldsymbol{\omega}} + \omega \times (\mathbf{I}_{\text{world}}\omega) \approx \mathbf{I}_{\text{world}}\dot{\boldsymbol{\omega}} \tag{6}$$

Given the robot the orientation matrix in the world frame $\mathbf{R} \in SO(3)$, the world-frame inertia is:

$$\mathbf{I}_{\text{world}} = \mathbf{R}\mathbf{I}_{\text{base}}\mathbf{R}^T \tag{7}$$

When the robot is close to upright ($\theta, \phi \approx 0$), the relationship between the angular velocity and the change rates of Euler angles can be written as:

$$\dot{\boldsymbol{\Theta}} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \approx \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{\omega} = \mathbf{R}_z(\psi)\boldsymbol{\omega} \tag{8}$$

With the above simplifications, we get the linear, time-varying dynamics model:

$$\underbrace{\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{\Theta}\\ \boldsymbol{p}\\ \boldsymbol{\omega}\\ \dot{\boldsymbol{p}}\end{bmatrix}}_{\dot{\boldsymbol{x}}_{\text{base}}} = \underbrace{\begin{bmatrix}\mathbf{0}_3 & \mathbf{0}_3 & \mathbf{R}_z(\psi) & \mathbf{0}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{1}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3\\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3\end{bmatrix}}_{\mathbf{A}}\underbrace{\begin{bmatrix}\boldsymbol{\Theta}\\ \boldsymbol{p}\\ \boldsymbol{\omega}\\ \dot{\boldsymbol{p}}\end{bmatrix}}_{\boldsymbol{x}_{\text{base}}} + \underbrace{\begin{bmatrix}\mathbf{0}_3 & \dots & \mathbf{0}_3\\ \mathbf{0}_3 & \dots & \mathbf{0}_3\\ \mathbf{I}_{\text{world}}^{-1}[\boldsymbol{r}_1]_\times & \dots & \mathbf{I}_{\text{world}}^{-1}[\boldsymbol{r}_1]_\times\\ \mathbf{I}_3/m & \dots & \mathbf{I}_3/m\end{bmatrix}}_{\mathbf{B}}\underbrace{\begin{bmatrix}\boldsymbol{f}_1\\ \boldsymbol{f}_2\\ \boldsymbol{f}_3\\ \boldsymbol{f}_4\end{bmatrix}}_{\boldsymbol{u}} + \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \boldsymbol{g}\end{bmatrix} \tag{9}$$

We then discretize the continuous time dynamics equation, which we use in our MPC formulation.

$$\boldsymbol{x}_{t+1} = \mathbf{A}'\boldsymbol{x}_t + \mathbf{B}'\boldsymbol{u}_t + \boldsymbol{g}' \tag{10}$$

where $\mathbf{A}'$, $\mathbf{B}'$, $\boldsymbol{g}'$ are the discrete time counterpart of $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{g}$ in Eq. (9).

Reference Trajectory Generation Given the desired linear velocity of the base $\bar{\boldsymbol{v}}_{\text{base}}$, we compute a desired trajectory $\bar{\boldsymbol{x}}_t$ for the next T timesteps, where T is the MPC planning horizon. In each reference state, we set $\dot{\bar{\boldsymbol{p}}}$ to $\bar{v}_{\text{base}}$ and set $\bar{\boldsymbol{p}}$ to numerically integrate $\bar{v}_{\text{base}}$ for speed-tracking, and set the desired orientation $\bar{\boldsymbol{\Theta}}$ and angular velocity to $\bar{\boldsymbol{\omega}}$ to 0 to ensure stable walking.

MPC Formulation Given the reference trajectory $\bar{\boldsymbol{x}}_{1,\dots,T}$, we solve for the ground reaction forces $\boldsymbol{u}_{1,\dots,T}$ by solving the following Quadratic Program (QP):

$$\begin{aligned} \min_{\boldsymbol{u}_{1,\dots,T}} \sum_{t=1}^{T} \|\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t\|_{\mathbf{Q}} + \|\boldsymbol{u}_t\|_{\mathbf{R}} & & \\ \text{subject to } \boldsymbol{x}_{t+1} = \mathbf{A}'\boldsymbol{x}_t + \mathbf{B}'\boldsymbol{u}_t + \boldsymbol{g}' & \quad & \text{Eq. (10)}\\ f_{i,t}^z = 0 & & \text{if leg } i \text{ is a swing leg at } t\\ f_{\min} \le f_{i,t}^z \le f_{\max} & & \text{if leg } i \text{ is a stance leg at } t\\ -\mu f_{i,t}^z \le f_{i,t}^x \le \mu f_{i,t}^z & & \forall i, t\\ -\mu f_{i,t}^z \le f_{i,t}^y \le \mu f_{i,t}^z & & \forall i, t \end{aligned} \tag{11}$$

where $\mathbf{Q}$, $\mathbf{R}$ are diagonal weight matrices. The constraints include the centroidal dynamics, the contact schedule of each leg and the approximated friction cone conditions. The optimized contact forces are then converted to motor torques using Jacobian transpose: $\boldsymbol{\tau} = \mathbf{J}^T\boldsymbol{f}$.

A.2 Swing Leg Control

The swing leg controller calculates the swing foot trajectories and uses Proportional-Derivative (PD) controllers to track these trajectories (Fig. 8b). To calculate a leg's swing trajectory, we first find its lift-off, mid-air and landing positions $(\boldsymbol{p}_{\text{lift-off}}, \boldsymbol{p}_{\text{air}}, \boldsymbol{p}_{\text{land}})$ (Fig. 8b). The lift-off position $\boldsymbol{p}_{\text{lift-off}}$ is the foot location at the beginning of the swing phase. The mid-air position $\boldsymbol{p}_{\text{air}} = \boldsymbol{p}_{\text{ref}} + (0, 0, z_{\text{des}})$ is a fixed distance above the normal standing position $\boldsymbol{p}_{\text{ref}}$. We use the Raibert Heuristic [38] to estimate the desired foot landing position:

$$\boldsymbol{p}_{\text{land}} = \boldsymbol{p}_{\text{ref}} + \boldsymbol{v}_{\text{CoM}} T_{\text{stance}}/2 \tag{12}$$

where $\boldsymbol{v}_{\text{CoM}}$ is the projected robot's CoM velocity onto the $x-y$ plane, and T_{stance} is the expected duration of the next stance phase, which can be calculated using the stepping frequency and swing ratio from the gait policy (Section 3.2). Raibert's heuristic ensures that the stance leg will have equal forward and backward movement in the next stance phase, and is commonly used in locomotion controllers [4, 3, 13].

Given these three key points, $\boldsymbol{p}_{\text{lift-off}}, \boldsymbol{p}_{\text{air}}$, and $\boldsymbol{p}_{\text{land}}$, we fit a quadratic polynomial, and computes the foot's desired position in the curve based on its progress in the current swing phase. Given the desired foot position, we then compute the desired motor position using inverse kinematics, and track it using a PD controller. We re-compute the desired foot position of the feet at every step (500Hz) based on the latest velocity estimation.

B Modeling Motor Power Consumption

B.1 DC Motor Model

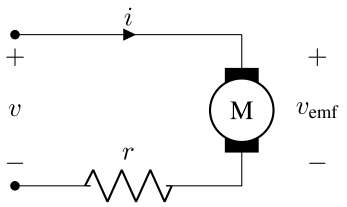


Figure 9: Schematic drawing of DC motor model.

We model a DC motor circuit as in Fig. 9, which includes a motor with internal resistance r and torque constant k. To apply a torque τ_m, the motor controller applies a voltage v to the motor, which generates a current i. As the motor rotates with angular velocity ω_m, it also generates a back-emf voltage v_{emf}. We aim to express the battery power consumption $p = vi$ in terms of the motor velocity ω_m and applied motor torque τ_m. If we ignore motor inductance and only consider steady-state behaviors, the circuit characteristic can be written as:

$$\tau_m = ki \tag{13}$$

$$v_{\text{emf}} = k\omega_m \tag{14}$$

$$i = \frac{v - v_{\text{emf}}}{r} \tag{15}$$

where Eq.13 and Eq.14 models steady-state motor behavior, and Eq.15 is derived from Ohm's law. Solving for v in terms of τ_m, ω_m and motor constants k, r, we get:

$$v = k\omega_m + \frac{\tau_m r}{k} \tag{16}$$

The power supplied by the battery can be computed by:

$$p = vi = \left(k\omega_m + \frac{\tau_m r}{k}\right)\frac{\tau_m}{k} = \tau_m\omega_m + \frac{r}{k^2}\tau_m^2 \tag{17}$$

Note that the first term is the *mechanical* power delivered by the motor, and the second term is the extra *heat* dissipation in the motor circuit. Since Unitree's battery management system does not support regenerative braking, we lower-bound the power consumption by 0, and get:

$$p_{\text{actual}} = \max\left(\tau_m\omega_m + \frac{r}{k^2}\tau_m^2, 0\right) \tag{18}$$

B.2 Power Consumption of A1 motors

Based on the motor characteristic curve of A1 (Fig. 10), at $\tau_m = 4$Nm and the output power is approximately 400w, with an efficiency of approximately 50%. Therefore, $\tau_m\omega_m \approx \frac{r}{k^2}\tau_m^2 \approx 400$w. We can then deduce that $\frac{r}{k^2} \approx 25$ and express power consumption as:

$$p_{\text{actual}} \approx \max(\tau_m\omega_m + 25\tau_m^2, 0) \tag{19}$$

The motor of A1 have a gear reduction ratio of 9.1. Therefore the joint velocity and joint torque (τ, ω) can be expressed in terms of motor velocity and motor torque (τ_m, ω_m) as:

$$\tau = 9.1\tau_m$$

$$\omega = \frac{\omega_m}{9.1}$$

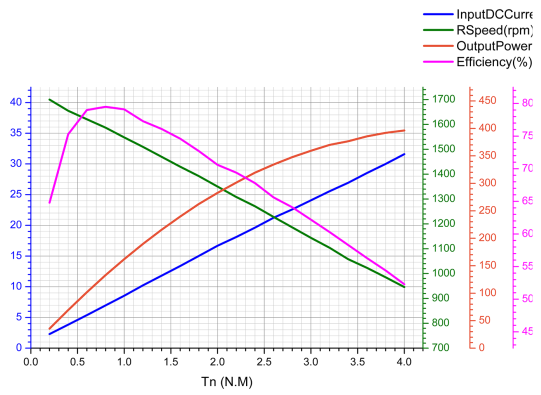


Figure 10: Characteristic curve for A1 motors [14] from robot manufacturer. The angular velocity and output torque are measured at the motor level without gear reduction.

Substituting into Eq. 19, we can express the power consumption in terms of joint torque and velocity:

$$p_{\text{actual}} \approx \max\left(\tau\omega + \frac{25}{9.1^2}\tau^2, 0\right) \approx \max(\tau\omega + 0.3\tau^2, 0) \tag{20}$$

C Experiment Details

C.1 Comparison with Different Learning Algorithms

CMA-ES Setup We obtain the CMA-ES implementation from Pycma [54]. We represented the policy using a fully connected neural network with 1 hidden layer of 256 units and tanh non-linearity. We initialize the algorithm with a mean of 0 and standard deviation of 0.03, and perform each update using a population size of 32.

ARS Setup In ARS, we represented the policy using a fully connected neural network with 1 hidden layer of 256 units and tanh non-linearity. The policy parameter is initialized to be all 0 at the start of training. At each iteration, we estimate the gradient by sampling 16 policy perturbations with standard deviation of 0.03, and update the policy using a step size of 0.02.

PPO and SAC Setup We obtain the PPO and SAC implementation from the Stable-baselines-3 [55] repo. For both algorithms, we represent the actor and the critic using a fully connected neural network with 2 hidden layers of 64 units each and tanh nonlinearity. For PPO, we additionally normalize the observation and reward using a moving average filter, which increases the total reward by 3x. We list the hyperparameters used in each algorithm in Table 3 and 4.

Parameter	Value
Learning rate	0.0003
# env steps per update	800
Batch size	64
# epochs per update	10
Discount factor	0.99
GAE λ	0.95
Clip range	0.2

Table 3: Hyperparameters used for PPO.

Parameter	Value
Learning rate	0.0003
Replay buffer size	10^6
Batch size	256
Discount factor	0.99
Entropy coefficient	Auto learned
# env steps per update	10
# gradient steps per update	1

Table 4: Hyperparameters used for SAC.

Parallel Rollouts to Reduce Training Time Since the low-level controller involves solving MPC problems, data collection in our environment is computationally heavy, and takes up a significant portion of the training time. To speed up training, we use multi-processing to parallelize rollouts whenever possible. For CMA-ES and ARS, we parallelize rollouts across 32 cpu cores, and collect 1

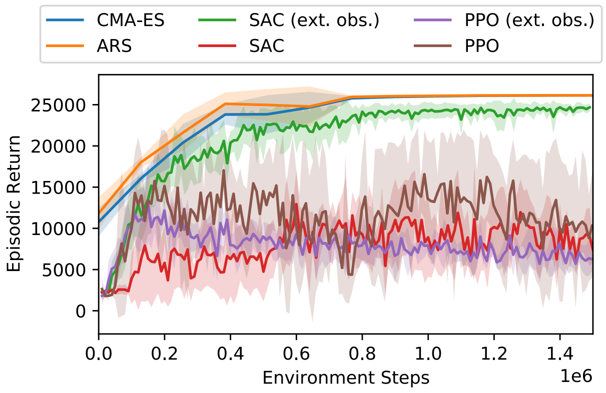


Figure 11: Learning curve for hierarchical policies trained by different algorithms. Results show average over 5 random seeds. Error bar indicates 1 standard deviation.

episode (400 steps) from each core per training iteration. For PPO, we parallelize rollouts across 32 cpu cores, and collect 25 steps from each core per training iteration. We do not parallelize rollouts for SAC since the algorithm does not benefit signficantly from parallel data collection.

Parallelized data collection greatly reduces the wall-clock training time, as CMA-ES (2 hours), ARS (2 hours) and PPO (12 hours) take significantly less time to reach 1.5 million environment steps compared to SAC (40 hours). In addition, ES-based algorithms (CMA-ES and ARS) update their policies less frequently and do not require back-propagation for policy update, which explains their wall-clock efficiency compared to PPO.

Learning Curves We plot the learning curves for different algorithms in Fig. 12. CMA-ES and ARS consistently out-performed other algorithms. When using the original observation space, which contains only the desired and current velocity, both PPO and SAC fail to complete the task, and show high variance in their learning curves. This is likely due to the lack of sufficient information to accurately estimate the value function. With the extended observation space, SAC learns to walk forward, but achieves a lower return compared to CMA-ES and ARS.

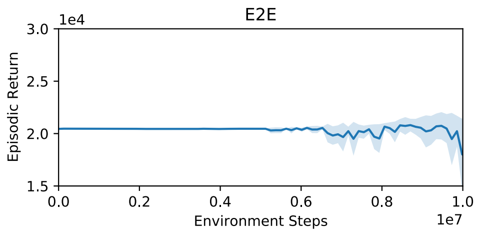
C.2 Comparison with Non-Hierarchical Policies

Reward Function for TG Policy We find the original reward function used in hierarchical environment (Eq. 3) to be ineffective in training the TG policy. Specifically, the TG policy usually incurs a significantly higher cost-of-transport compared to the hierarchical policy, especially in early stages of training. As a result, the reward (Eq. 3) is mostly negative, and CMA-ES learns to maximize return by terminating each episode early. Therefore, we reduce the energy penalty weight (w_e in Eq.3) from 0.37 to 0.037 to ensure that the reward stays positive.

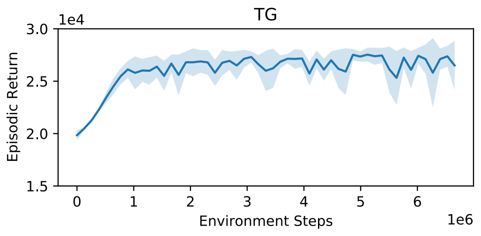
Reward Function for E2E Policy In addition to reducing the energy penalty weight, we found it necessary to perform further reward shaping for the E2E policy. Without the cyclic trajectory priors defined by TG, the E2E policy struggles to maintain balance and often walks forward in unstable, tilted pose. To encourage stable, up-right walking, we include additional terms based on the height and orientation of the robot, which is similar to the cost function used in the low-level convex MPC controller (Eq. 11). We end up with the following reward function:

$$r = c - w_v \underbrace{\left\|\frac{\bar{v}_{\text{base}} - v_{\text{base}}}{\bar{v}_{\text{base}}}\right\|^2}_{\text{Speed Penalty}} - w_e \underbrace{\frac{\sum_{i=1}^{12} \max(\tau_i \omega_i + \alpha\tau_i^2, 0)}{mg\bar{v}_{\text{base}}}}_{\text{Energy Penalty (Cost of Transport)}} - w_o \underbrace{(\text{roll}^2 + \text{pitch}^2)}_{\text{Orientation Penalty}} - w_h \underbrace{(\bar{h} - h)^2}_{\text{Height Penalty}} \tag{21}$$

where the speed penalty and energy penalty is the same as defined in Eq. 3. The orientation penalty penalizes the robot's deviation from an upright posture based on the IMU reading. The height



(a) E2E



(b) TG

Figure 12: Learning curves for non-hierarchical policies. Results are averaged over 5 random seeds. Error bar shows 1 standard deviation.

penalty penalizes the robot's deviation from a desired walking height, where $\bar{h}$ and h are the desired and actual height of the robot's center of mass.

We set $\bar{h} = 0.26$, which is the same as the reference height in the low-level convex MPC controller in the hierarchical setting (Section A.1). We use the same weight for the alive bonus $c = 3$ and speed penalty $w_v = 1$ as in the hierarchical setup (Eq. 3, reduced w_e from 0.37 to 0.037, and set $w_o = 10, w_h = 200$ for E2E experiments.

Policy Representation and Training To train the non-hierarchical policies, we followed the same setup as the hierarchical policy (Section. 4.3). We represent each policy using a neural network with 256 units and tanh non-linearity, and trained the policy using the same CMA-ES algorithm.

Learning Curves We plot the learning curves of non-hierarchical policies in Fig. 12. As noted in section 5.4, E2E only learns to stand for the entire episode, and the learning curve eventually becomes unstable. Although TG policy slowly learns to walk, the resulting gait is jaggy and only uses 3 legs.