Zeroth-01 Push-Recovery and Controlled Kneel Controller

Nicolas Burgos

1. Problem and Motivation

Humanoid robots face a fundamental challenge: when pushed beyond their balance limits, they fall catastrophically. Unlike wheeled robots that can simply stop, bipeds have a narrow stability margin and high center of mass. A fall can damage expensive hardware, injure nearby humans, or destroy the robot entirely.

This project addresses fall mitigation through two complementary strategies:

1. **Push Recovery**: Active stepping and ankle torque adjustments to regain balance after disturbances
2. **Controlled Kneel**: When recovery is impossible, lower the center of mass safely rather than toppling

The controlled kneel approach is inspired by biomechanics research on eccentric muscle control—the same mechanism that allows humans to sit down gently rather than collapse. The goal is a policy that knows when to fight for balance and when to yield gracefully.

2. System Overview

Simulation Framework

The project uses **MuJoCo** physics simulation via **K-Scale's KSIM** library, which provides:

- Vectorized environments for parallel rollouts
- MJCF robot model of the Zeroth-01 biped (20 actuated joints)
- Domain randomization (friction, mass, sensor noise)
- Push disturbance events during training

Training uses **Proximal Policy Optimization (PPO)** implemented in **JAX** with JIT compilation for efficient CPU/GPU execution.

Policy Architecture

Component	Specification
Policy Network	3-layer GRU (recurrent)
Hidden Size	64 units
Action Distribution	Mixture of 3 Gaussians per joint
Total Parameters	195,061
Observations	50 dimensions
Actions	20 joint position targets

The recurrent architecture allows the policy to integrate temporal information—essential for dynamic balance where history matters.

Observation Space (50 dims): - Joint positions: 20 dims - Joint velocities: 20 dims - IMU orientation: 4 dims (quaternion) - Velocity commands: 6 dims

Action Space (20 dims): - Position targets for each joint, output as mixture-of-gaussians for exploration

State Machine Concept

The full system design includes a deterministic state machine for mode switching:

```
RECOVER → KNEEL → STABLE
   ↑         │
   └─────────┘ (if conditions improve)
```

Transition triggers: - **RECOVER → KNEEL**: Tilt exceeds threshold (0.5 rad) for 10+ consecutive frames - **KNEEL → STABLE**: Robot reaches stable kneeling configuration

This project demonstrates the training pipeline; the state machine is designed but not yet integrated into training.

---

3. Key Methods

Reward Design

The reward function balances multiple objectives:

Reward Component	Weight	Purpose
`forward_velocity`	+5.0	Encourage locomotion
`stay_alive`	+1.0	Survival bonus
`feet_airtime`	+2.5	Alternating gait timing
`arm_pose`	-2.0	Keep arms stable
`lateral_velocity`	-1.0	Discourage sideways drift
`joint_limits`	-10.0	Penalize limit violations
`action_rate`	-0.1	Smooth control outputs

The reward weights were tuned empirically. Higher weights on `forward_velocity` and `feet_airtime` produced more stable gaits than uniform weighting.

Curriculum Considerations

Training includes randomized push disturbances via `LinearPushEvent`:

Stage	Force Range (N)	Interval (s)
Easy	±20	2-4
Medium	±40	1.5-3
Hard	±60	1-2.5
Kneel Trigger	±100	1-2

Pushes are applied to the robot's torso at random intervals, teaching the policy to maintain balance under perturbation.

Evaluation Battery

A systematic push battery (not yet implemented) would test:

```
PUSH_BATTERY = [
    {"force": (20, 0, 0), "label": "front_easy"},
    {"force": (-25, 0, 0), "label": "back_easy"},
    {"force": (50, 0, 0), "label": "front_medium"},
    {"force": (90, 0, 0), "label": "front_hard"},
    # ... additional scenarios
]
```

Metrics to track: - Recovery rate (% pushes survived) - Peak base acceleration during descent - Time to stable configuration

---

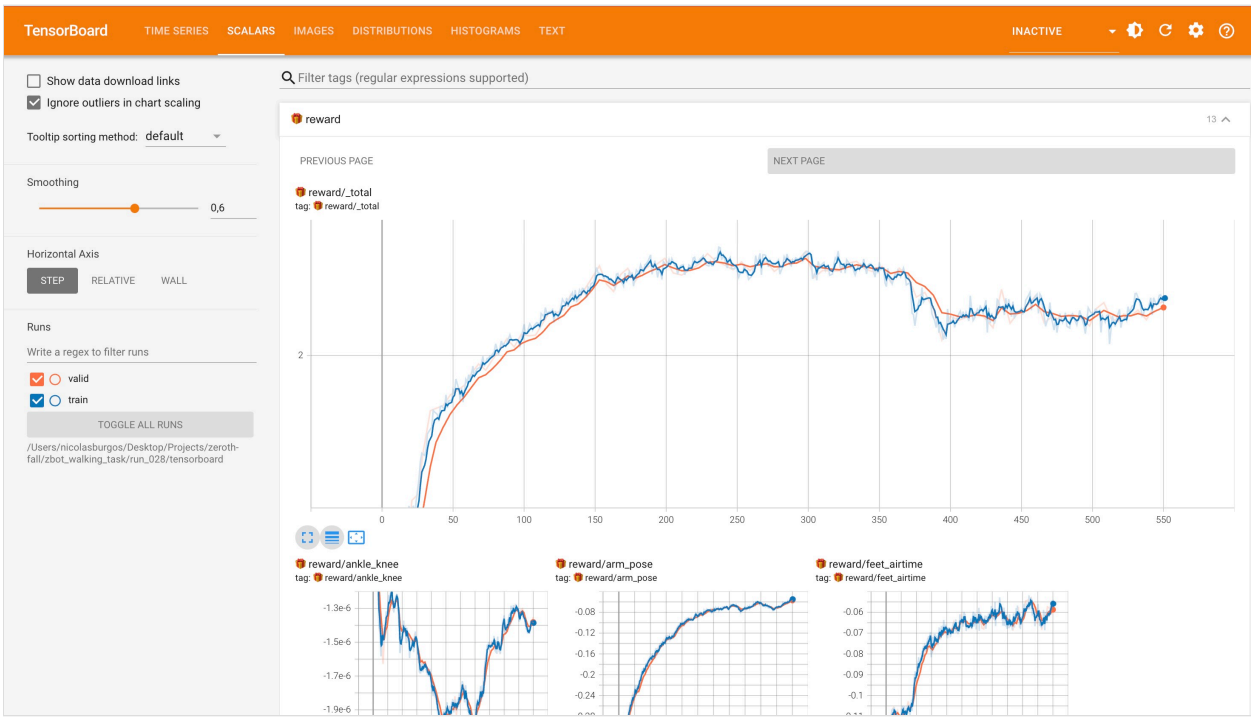
4. Results

Training Configuration

Parameter	Value
Environments	8 (parallel)
Batch Size	4
Rollout Length	2.0 seconds
Learning Rate	3×10^{-4}
Training Steps	~550
Hardware	MacBook Air M2 (CPU)

Learning Curves

Total Reward Progression



The policy converged from an initial reward of ~2.0 to a plateau around 2.2-2.4 over 550 steps. The plateau indicates the policy has reached a local optimum for the current reward weights.

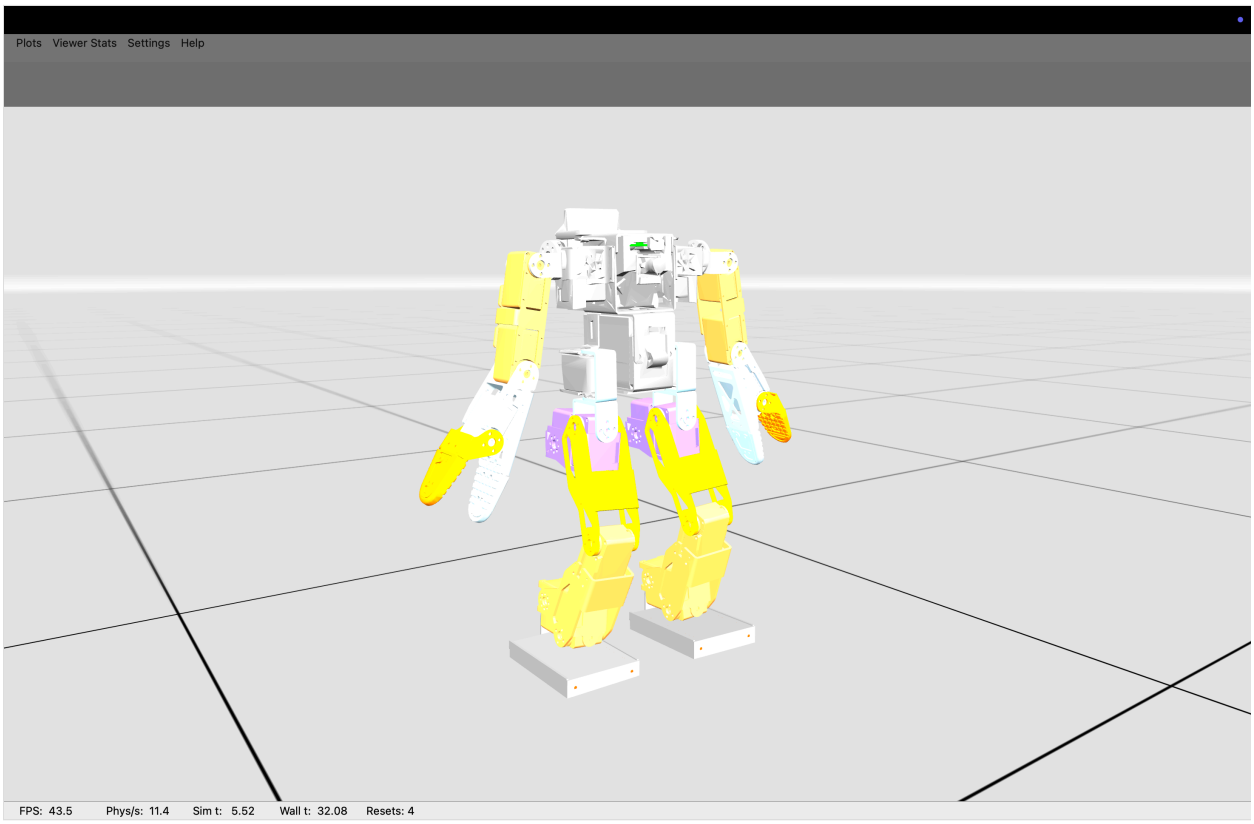
Forward Velocity Reward

Forward velocity improved from 0.64 to 0.78, demonstrating the policy learned to walk forward. Oscillations reflect exploration of different gait patterns.

Component Analysis

Reward	Start	End	Interpretation
`_total`	2.0	2.4	Overall improvement
`forward`	0.64	0.78	Better locomotion
`arm_pose`	-0.28	-0.08	Arms more stable
`feet_airtime`	-0.10	-0.06	Improved gait timing
`lateral_vel`	0.5	0.7	Less sideways drift

Simulation Demo



The trained policy demonstrates basic forward walking with reasonable gait timing. The robot maintains balance during normal operation and shows rudimentary push recovery behavior.

---

5. Limitations and Future Work

Current Limitations

Simulation Only: This project operates entirely in simulation. Real-world deployment would require: - Careful actuator modeling (the simulated Feetech servos may not match real dynamics) - Sensor noise characterization - Safety systems (e-stops, joint limit enforcement) - Tethered testing before untethered operation

Partial Implementation: Several designed components are not yet implemented: - State machine integration with training - Formal evaluation battery - Controlled kneel

mode (design complete, training not started) - Domain randomization sweep

Scale Constraints: Training was performed on laptop CPU (~30s per step) rather than GPU cluster. Results demonstrate the pipeline works but are not optimized for performance.

What I Would Investigate Next

1. **Kneel Mode Training**: Train a separate policy with `mode=1` input, starting from unstable initial conditions
2. **State Machine Integration**: Run combined training where the mode switches based on tilt/angular velocity
3. **Quantitative Evaluation**: Implement `push_battery.py` and measure recovery rates systematically
4. **Sim-to-Real Gap**: If hardware access were available, characterize how policy performance degrades on real robot

Open Questions

- How should the kneel reward balance "lower COM quickly" vs "minimize impact"?
- What tilt threshold optimizes the RECOVER→KNEEL transition?
- Can a single policy handle both modes, or are two separate policies more robust?

References

- K-Scale Labs. *ksim-gym-zbot*. https://github.com/kscalelabs/ksim-gym-zbot
- Schulman et al. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347
- MuJoCo Physics Engine. https://mujoco.org
- Zeroth Robotics. *Zeroth-01 Humanoid Platform*. https://github.com/zeroth-robotics/zeroth-bot

---

Code and training artifacts available at: https://github.com/[username]/zeroth-fall