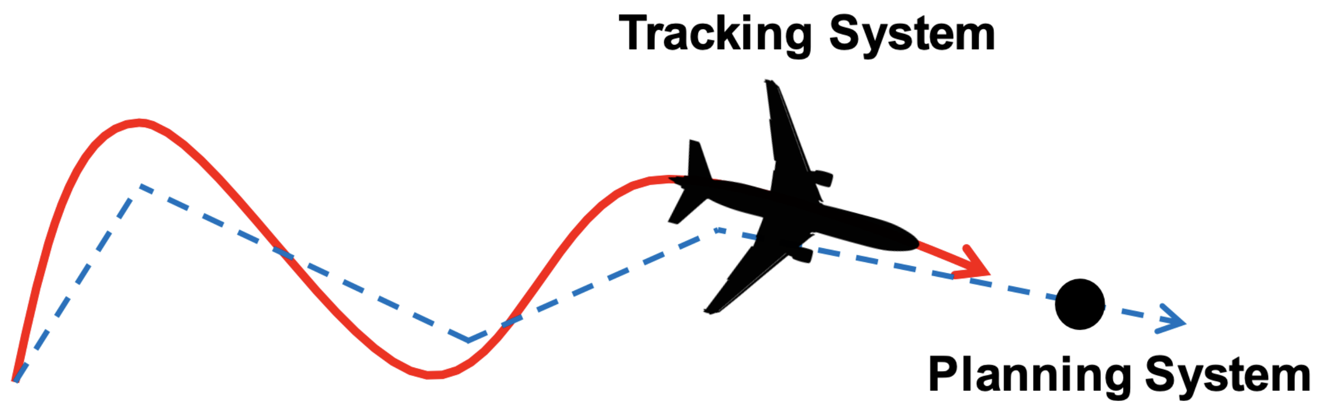
Introduction – Deep Tube MPC

- We generally use a simplified model of system dynamics/kinematics to plan trajectories through environments.
- The real robot cannot follow these paths precisely, which results in an error between the **tracking system** and the **planning system**.
- To safely navigate environments with obstacles, we must take this error into account while generating trajectories.



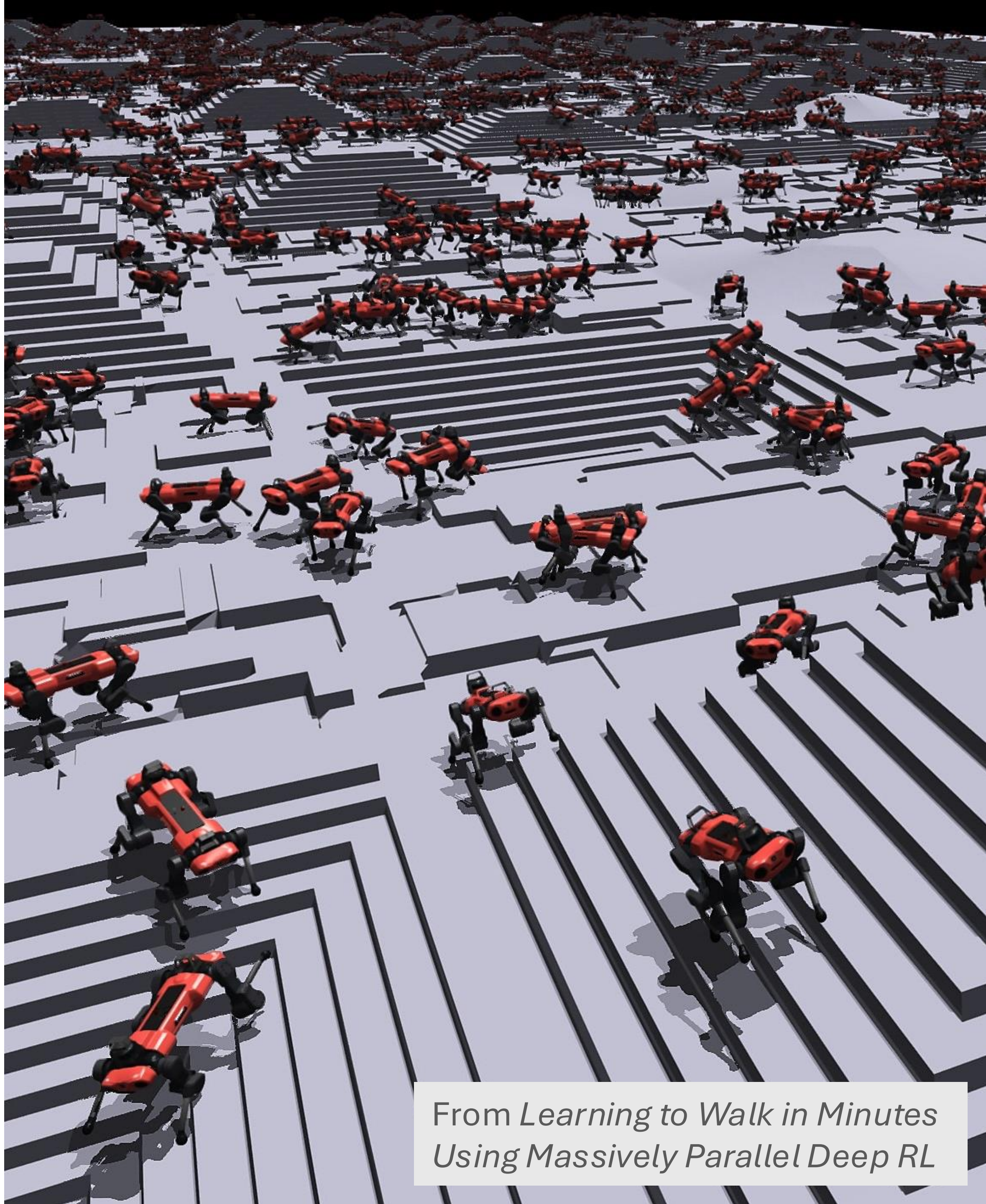
From *FaSTrack* ('17)

Introduction – Deep Tube MPC

- Prior Work
  - *FaSTrack ('17, '21)*: produces safety guarantees based on HJ reachability analysis, precomputing a tracking error & safety control function offline.
    - HJB must be solved over the state space, which is the same dimensionality as the system. Thus, the curse of dimensionality applies.
  - *DL Tubes for Tube MPC ('20)*: learns the distribution of MPC-derived trajectories in closed-loop to generate stochastic bounds recursively.
    - Tubes must grow throughout trajectory if using high confidence levels.
    - Lacks the ability to gather large-scale data for effective learning approach.
- Problem
  - The tracking system's **high-dimensionality** makes it infeasible to directly compute this error in planning.
  - Learning approaches can't produce guarantees with **high confidence** levels.

Introduction – RL Trajectory Tracking

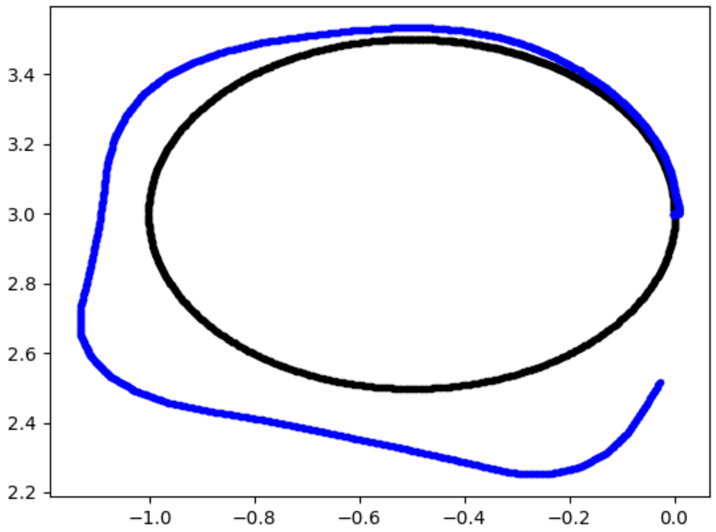
- To explore tubes, we need an environment that relates a planning and tracking model.
- Recently, many papers have used RL in simulation to synthesize velocity-based walking controllers.
    - *Cassie paper ('24)*
    - Some of them focus on massive parallelization (e.g., *legged_gym*)



From *Learning to Walk in Minutes Using Massively Parallel Deep RL*

Introduction – RL Trajectory Tracking

- Problem
  - None of the approaches use massive parallelization to learn a trajectory-based policy
    - We found that PID-driven velocity controllers did not track paths well.
    - We need to collect a large database of tracking error for different trajectories.
  - We want to demonstrate sim-to-real on a robot in the lab – Hopper!
    - Requires custom dynamic models not supported.

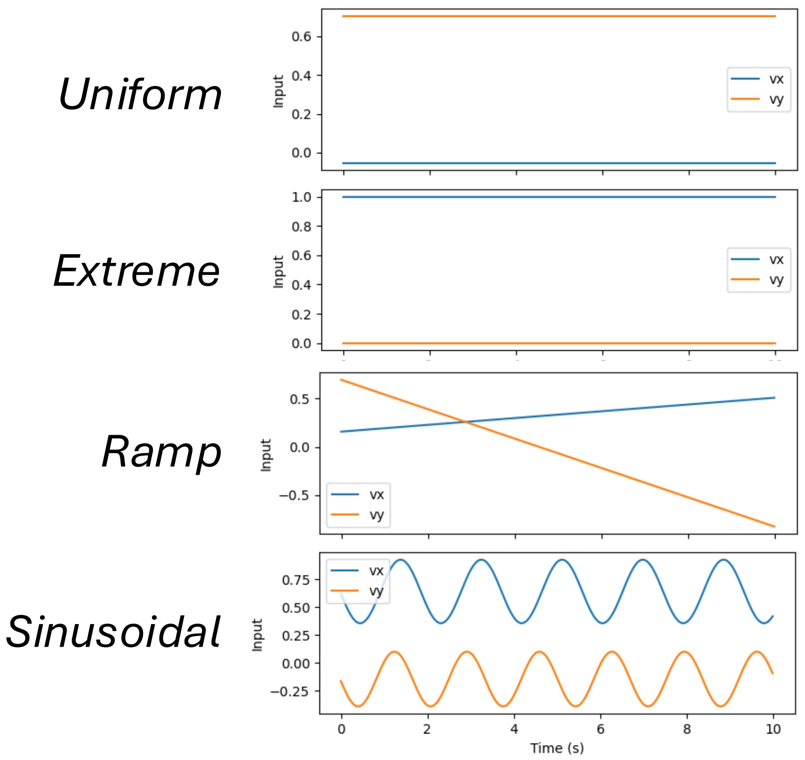


Massively Parallelized RL Trajectory Tracking

- Trajectory Generation

- Implemented ROMs
  - Single Integrator – n: $\{v_x, v_y\}$
  - Double Integrator
  - Unicycle
  - Lateral Unicycle
  - Extended Unicycle
  - Extended Lateral Unicycle

Principle Trajectories



Massively Parallelized RL Trajectory Tracking

- Trajectory Generation

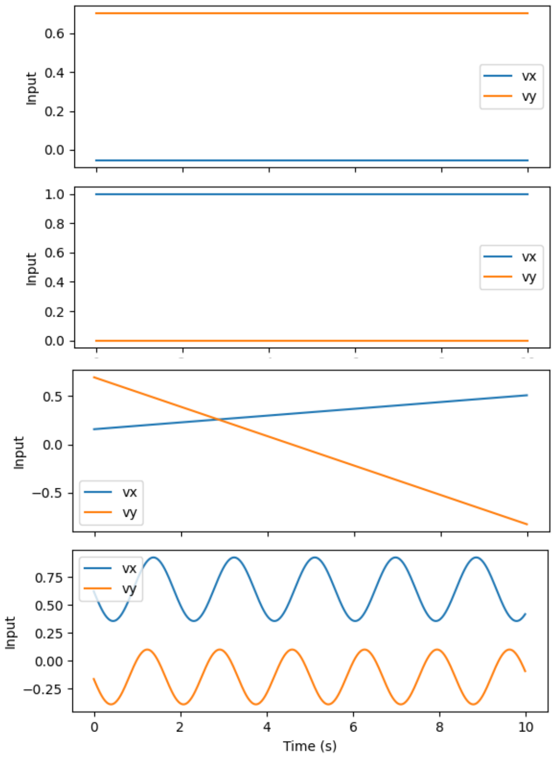
Principle Trajectories

Uniform

Extreme

Ramp

Sinusoidal



Massively Parallelized RL Trajectory Tracking

- Trajectory Generation

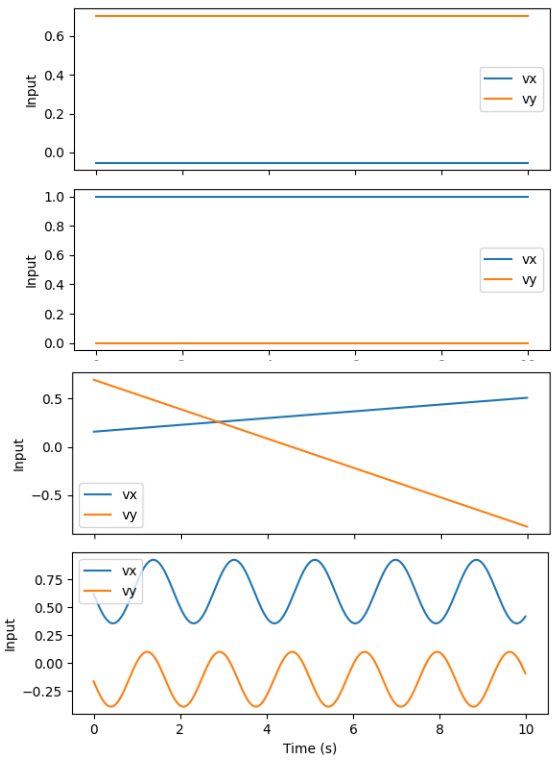
Principle Trajectories → **Rand. Linear Combination**

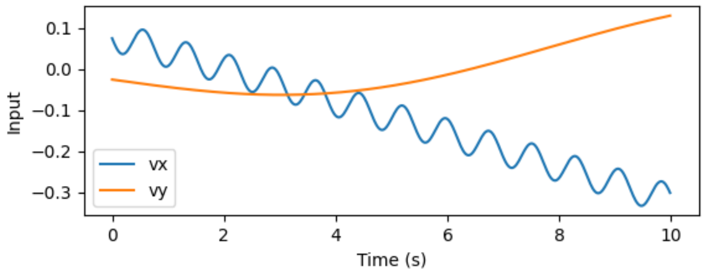
Uniform

Extreme

Ramp

Sinusoidal



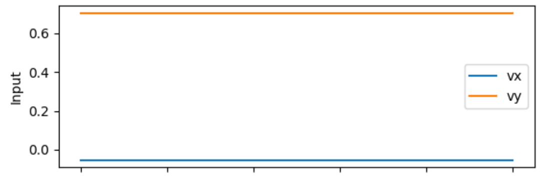


Massively Parallelized RL Trajectory Tracking

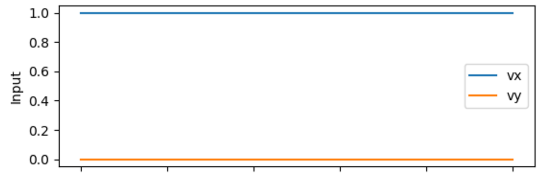
- Trajectory Generation

Principle Trajectories → **Rand. Linear Combination**

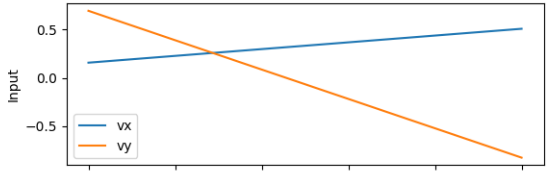
Uniform



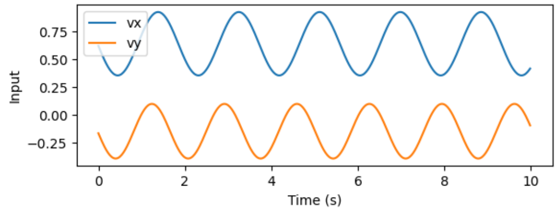
Extreme

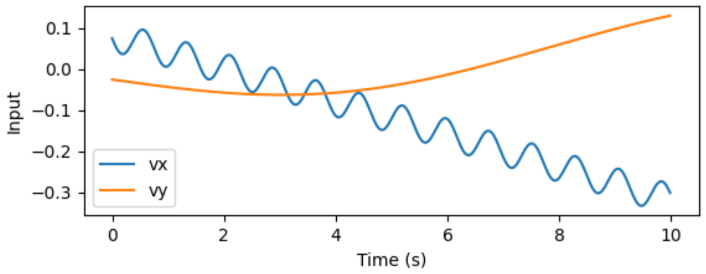


Ramp

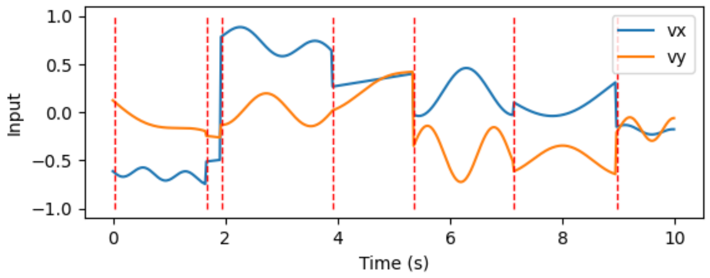


Sinusoidal





Rand. Resampling



Massively Parallelized RL Trajectory Tracking

- Trajectory Generation

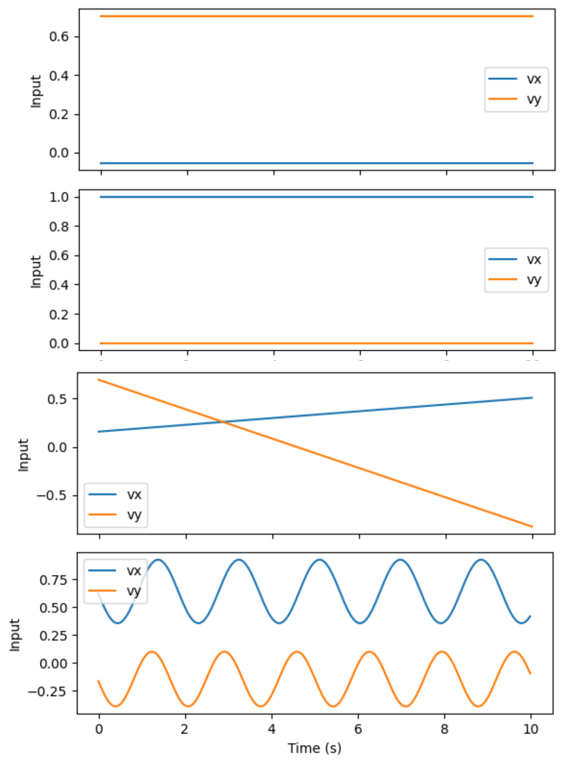
Principle Trajectories

Uniform

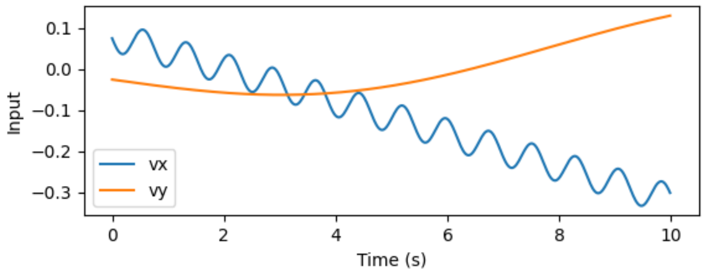
Extreme

Ramp

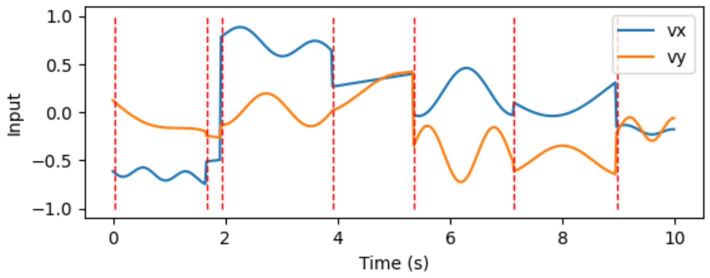
Sinusoidal



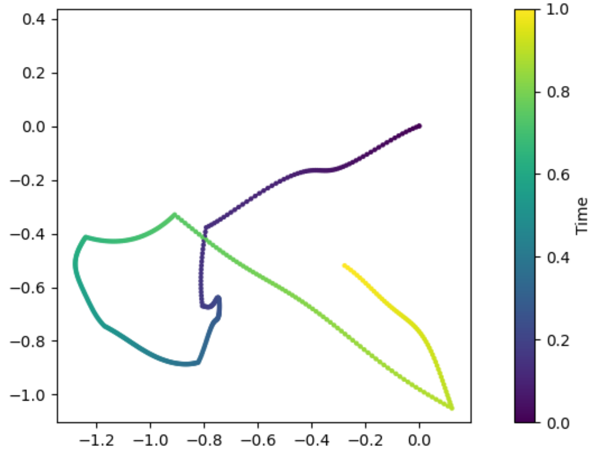
Rand. Linear Combination



Rand. Resampling



Final Trajectory



Massively Parallelized RL Trajectory Tracking

- Curriculum Learning, Raibert Heuristic, Reference Trajectories

- For different robots, different rewards throughout training accelerate the learning process
- E.g., rewards for mimicking behavior of simplified models to kickstart walking movement

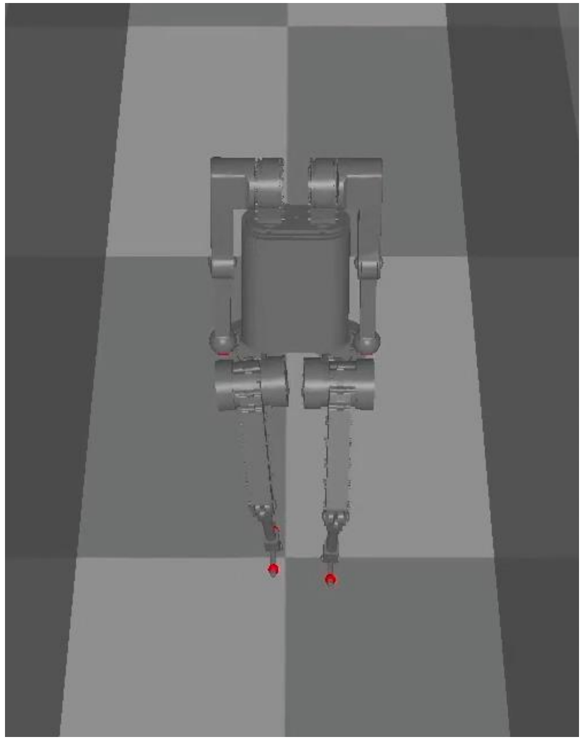
Raibert Heuristic

$$\psi_{roll} = -K_p e_x - K_v e_{v,x}$$

$$\psi_{pitch} = -K_p e_y - K_v e_{v,y}$$

(Hopper)

Reference Trajectories

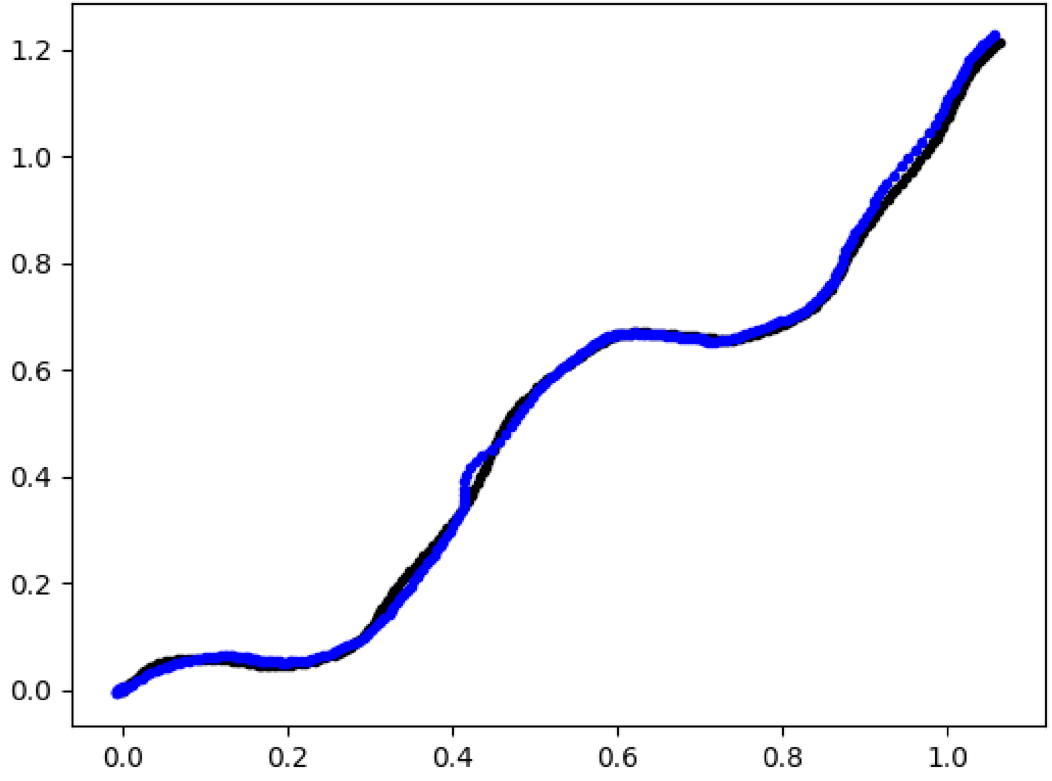


(Adam)

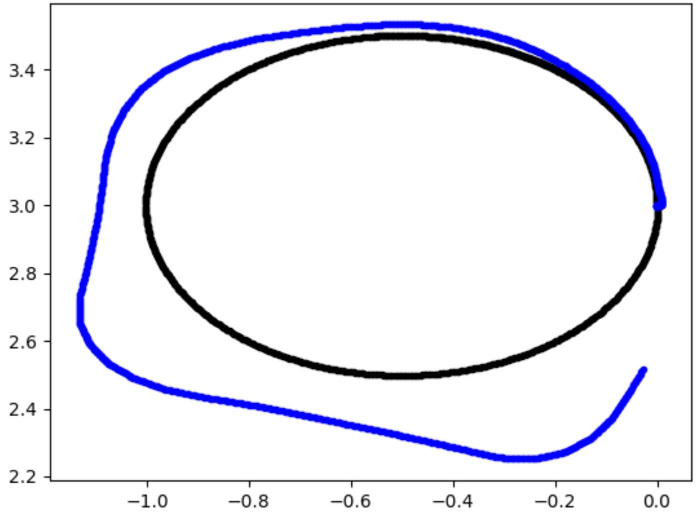
Massively Parallelized RL Trajectory Tracking

- Results

Trajectory Tracking Policy

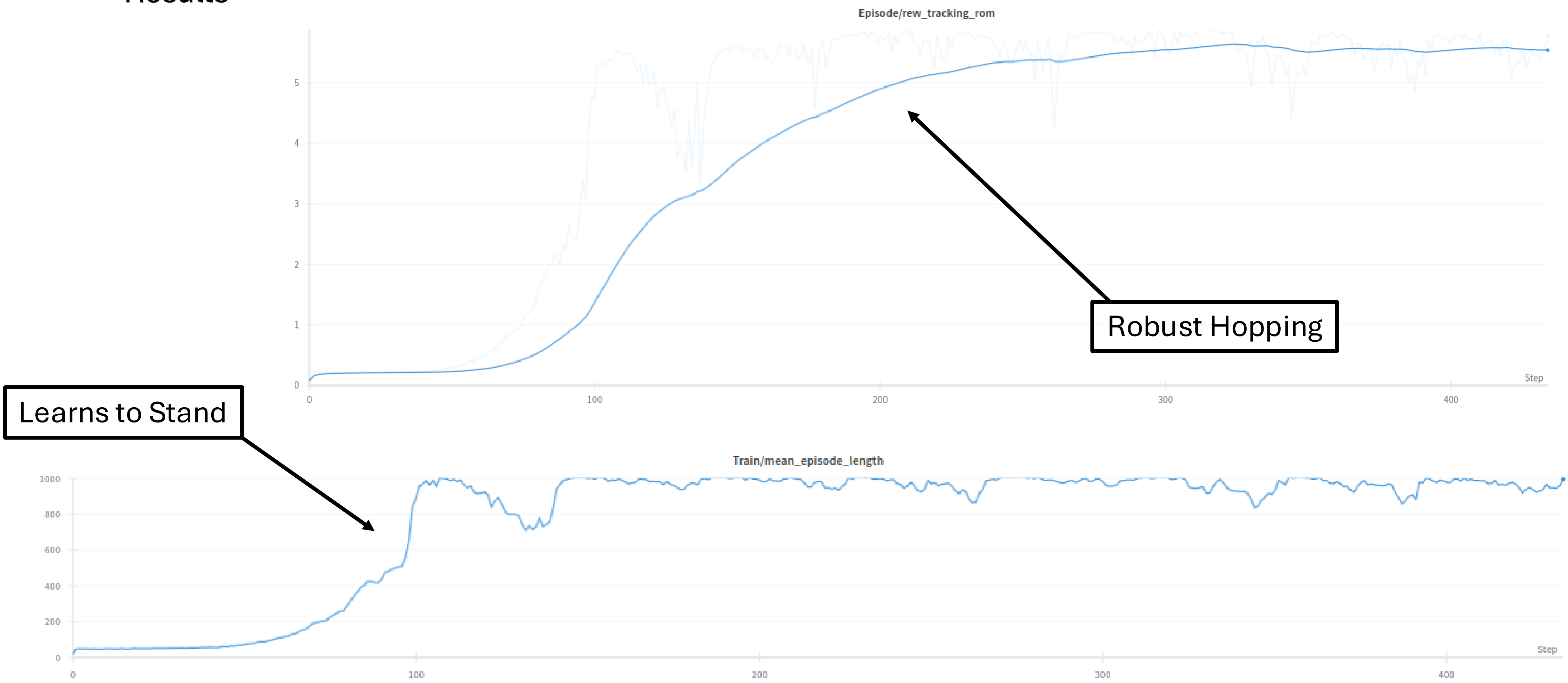


Original PD-Controlled Velocity Policy



Massively Parallelized RL Trajectory Tracking

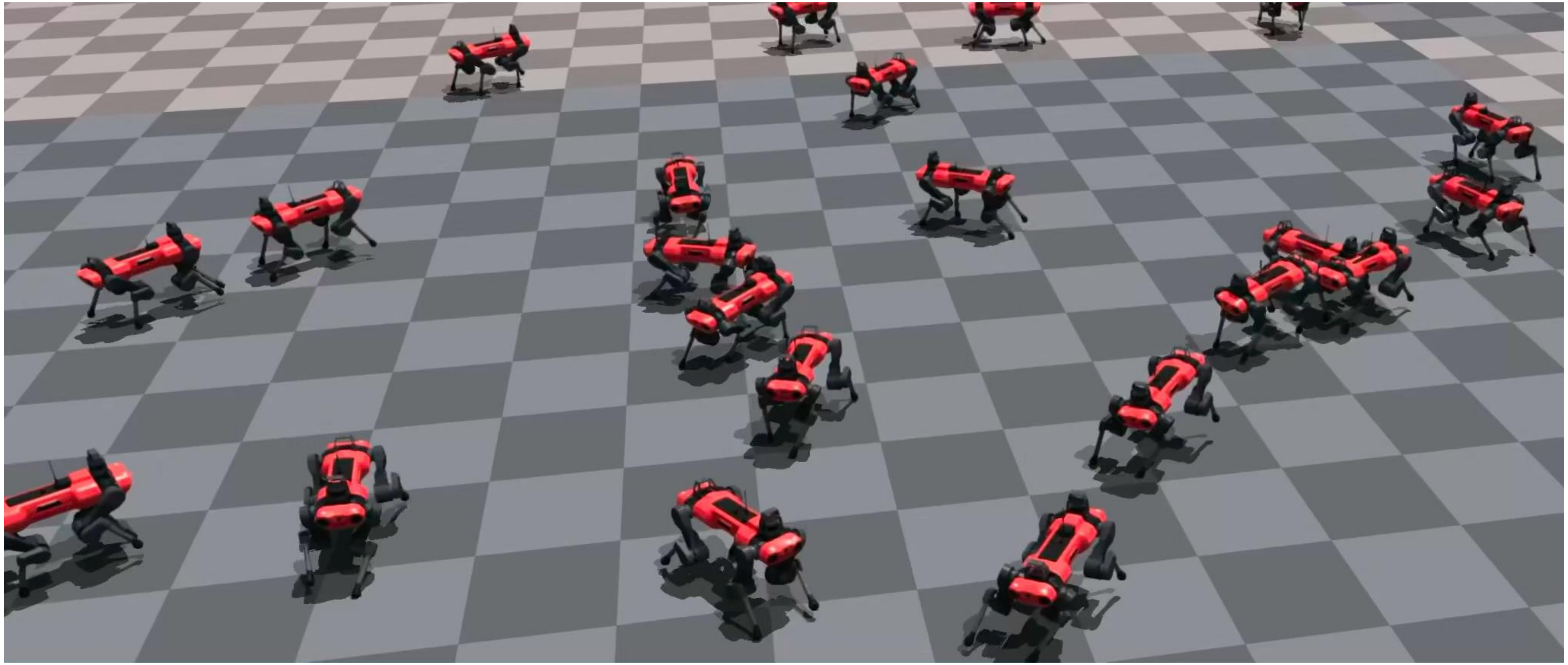
- Results



Demonstration

- Single Simulation, Sim-to-Sim

IsaacSim

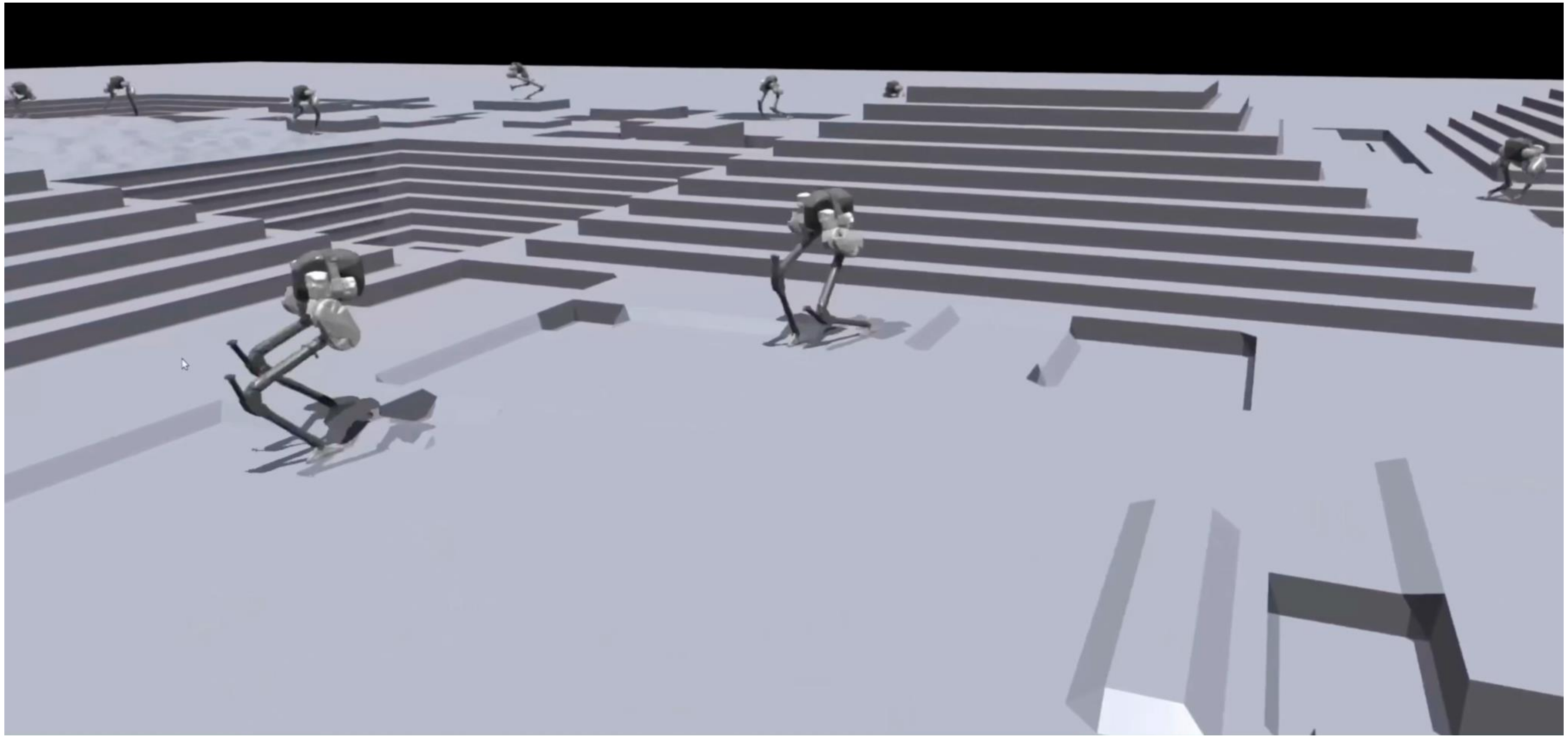


(Anymal C)

Demonstration

- Single Simulation, Sim-to-Sim

IsaacSim

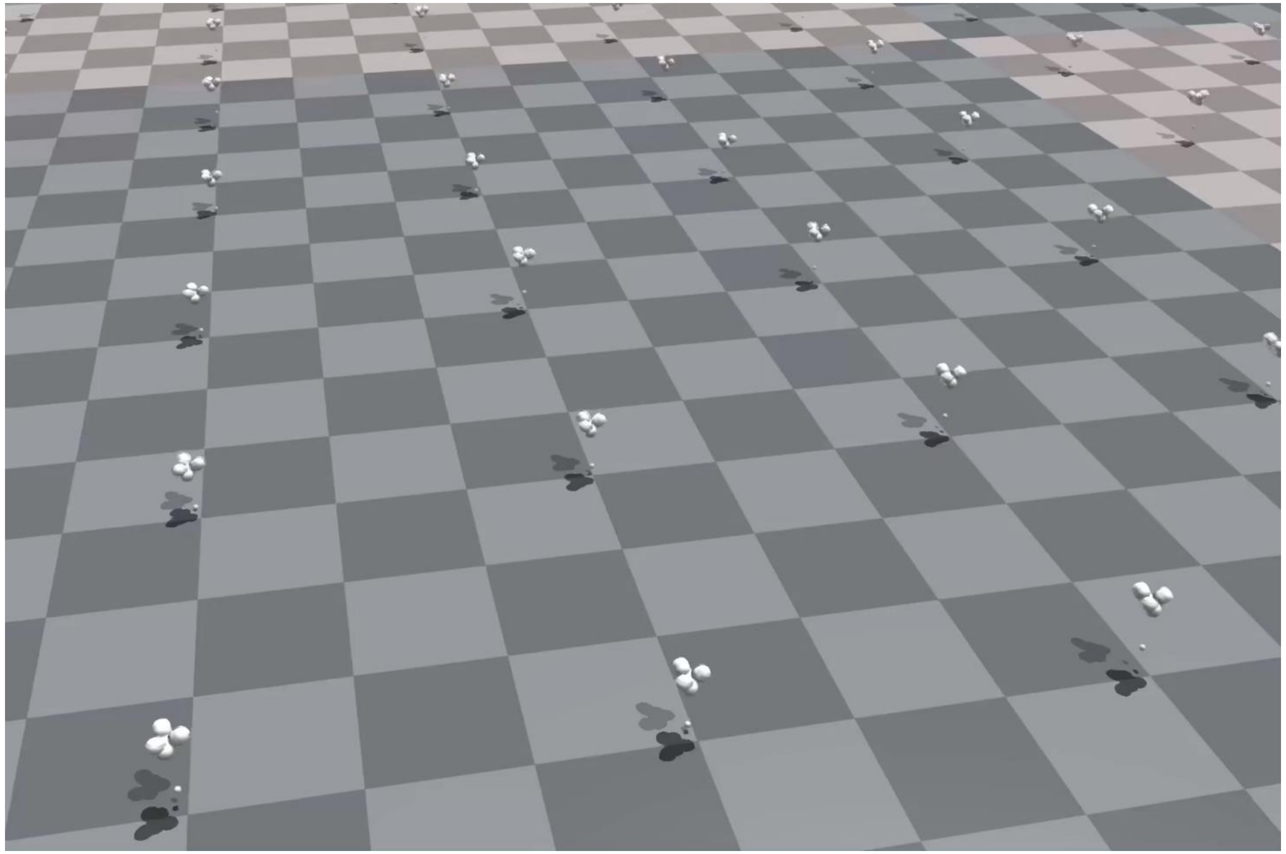


(Cassie)

Demonstration

- Single Simulation, Sim-to-Sim

IsaacSim



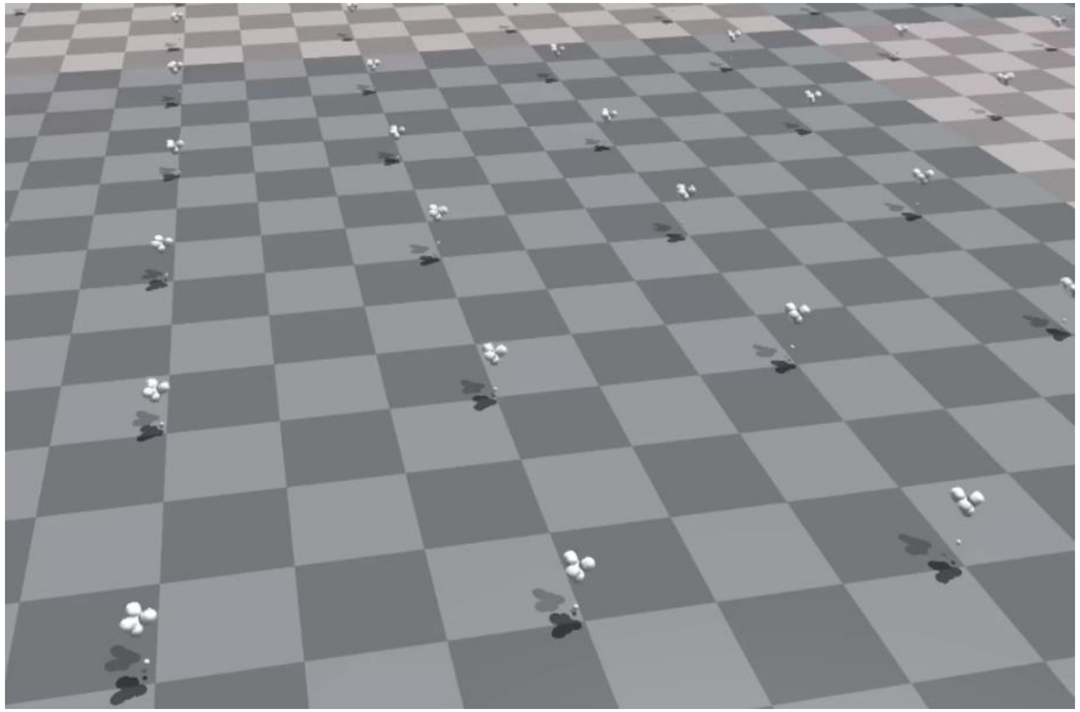
(Hopper)

Demonstration

- Single Simulation, Sim-to-Sim

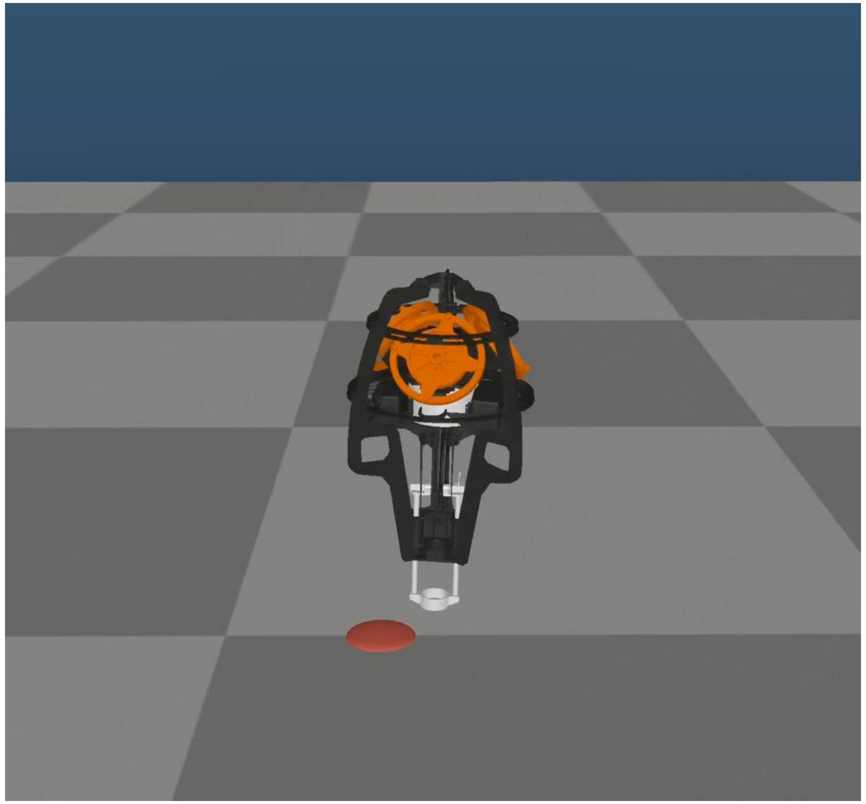
Sim-to-Sim

IsaacSim



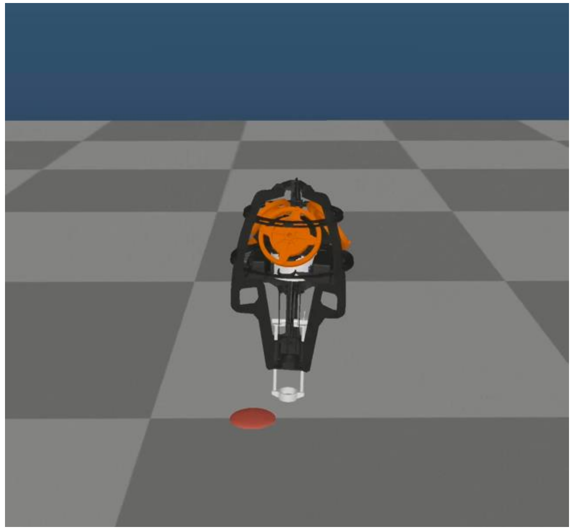
(Hopper)

Mujoco



Demonstration

- Sim-to-Real



Sim-to-Real

(Hopper)