QuadSwarm: A Modular Multi-Quadrotor Simulator for Deep Reinforcement Learning with Direct Thrust Control

Zhehui Huang, Sumeet Batra, Tao Chen, Rahul Krupani, Tushar Kumar
Artem Molchanov, Aleksei Petrenko, James A. Preiss, Zhaojing Yang, Gaurav S. Sukhatme

University of Southern California

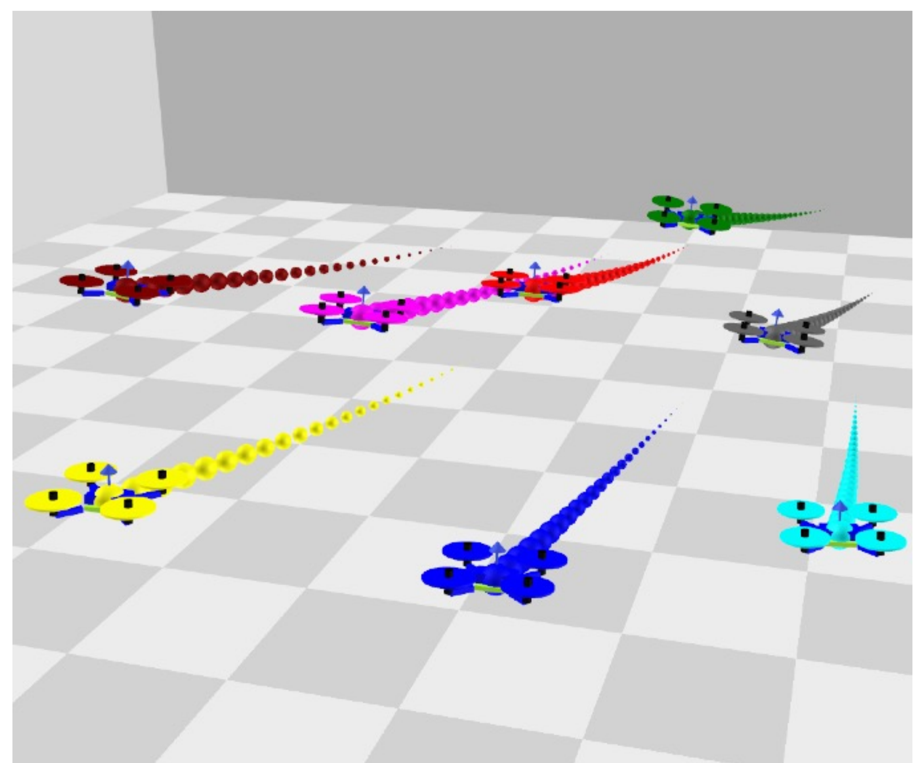
USC University of Southern California

ROBOTIC EMBEDDED SYSTEMS LABORATORY

Highlights

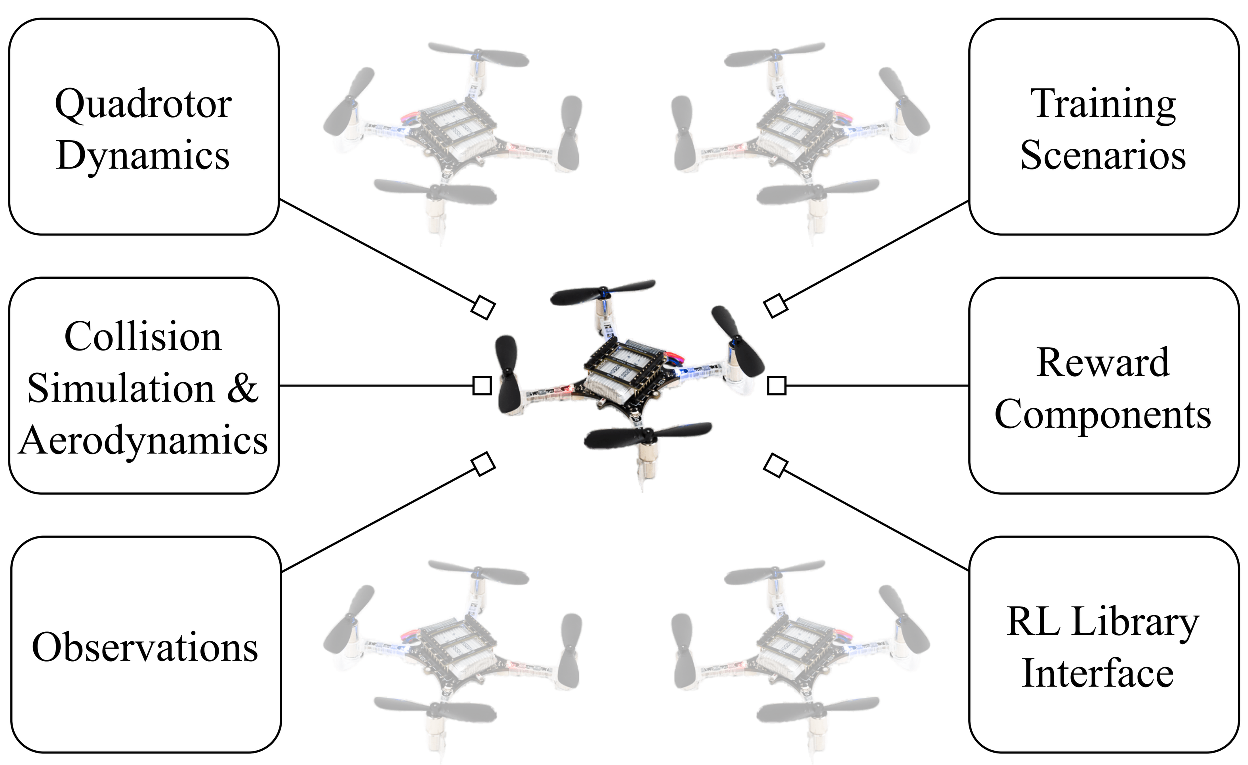
QuadSwarm: A fast and high-parallelizable simulator with a level acceptable for transferring policies learned in the simulator to reality.

- ✓ Supports Crazyflie 2.x
- ✓ Demonstrated sim2real transferability for single and multi-quadrotor teams
- ✓ Supports Per-rotor thrust control
- ✓ Fast single-threaded throughput and scales with additional compute
- ✓ A diverse collection of learning scenarios
- ✓ 100% written in Python, and sped up with Numba



Paper

System Overview



Quadrotor Dynamics

$$\ddot{x} = g + \frac{\mathbf{R}f}{m} \qquad \dot{\mathbf{R}} = \omega_{\times}\mathbf{R}$$

$$\dot{\omega} = \mathbf{I}^{-1}(\tau - \omega \times (\mathbf{I}\cdot\omega)) \qquad \tau = \tau_p + \tau_{th}$$

Motor Lag

$$\hat{u}^{(t)} = \sqrt{\hat{f}^{(t)}} \qquad \hat{u}_f^{(t)} = \alpha_{lag}(\hat{u}^{(t)} - \hat{u}_f^{(t-1)}) + \hat{u}_f^{(t-1)}$$

Final Thrust

$$f = f_{max}\cdot(\hat{u}_f)^2 + \epsilon_f$$

Collision Simulation & Aerodynamics

Quadrotor & Quadrotor

$$n_{col} = \frac{x_1 - x_2}{\|x_1 - x_2\|_2} \qquad \tilde{v} = (v_2 \cdot n_{col} - v_1 \cdot n_{col})\cdot n_{col}$$

$$v_1 \leftarrow \alpha_1(v_1 + \tilde{v} + \epsilon_{v1}) \qquad v_2 \leftarrow \alpha_2(v_2 - \tilde{v} + \epsilon_{v2})$$

$$\omega_1 \leftarrow \omega_1 + \epsilon_{\omega 1} \qquad \omega_2 \leftarrow \omega_2 + \epsilon_{\omega 2}$$

Quadrotor & Walls / Ceiling

Similar to quadrotor & quadrotor collision model, except the collision updates are only applied to the quadrotor.

Quadrotor & Ground

$$f_{xy} \leftarrow \max(f_{xy} - \mu(mg - f_z), 0) \qquad \|v\|_2 = 0$$

$$f_{xy} \leftarrow f_{xy} - \mu(mg - f_z) \qquad \|v\|_2 > 0$$

Downwash

Apply when the relative positions of quadrotors are within certain range

$$\ddot{x} = k_1(k_2\delta_{pos} + b_1) + \epsilon_d \qquad \dot{\omega} = \epsilon_{\omega d}$$

Observations

$$[\delta_{xi}, v_i, R_i, \omega_i, [\tilde{x_{i1}}, \tilde{v_{i1}}, ..., \tilde{x_{iK}}, \tilde{v_{iK}}]]$$

Observation Noise

$\epsilon_x = U(0, 5e^{-3})$ $\epsilon_v = U(0, 1e^{-2})$ $\epsilon_\omega = U(0, 1.75e^{-4})$

Training Scenarios

Static formations
Dynamic formations
1) Dynamic goals 2) Swap goals
3) Shrink & Expand 4) Swarm-vs-Swarm
Evader Pursuit
1) 3D Lissajous curve 2) Bezier curve

Support formations: circle, grid, sphere, cylinder, cube

Reward Components

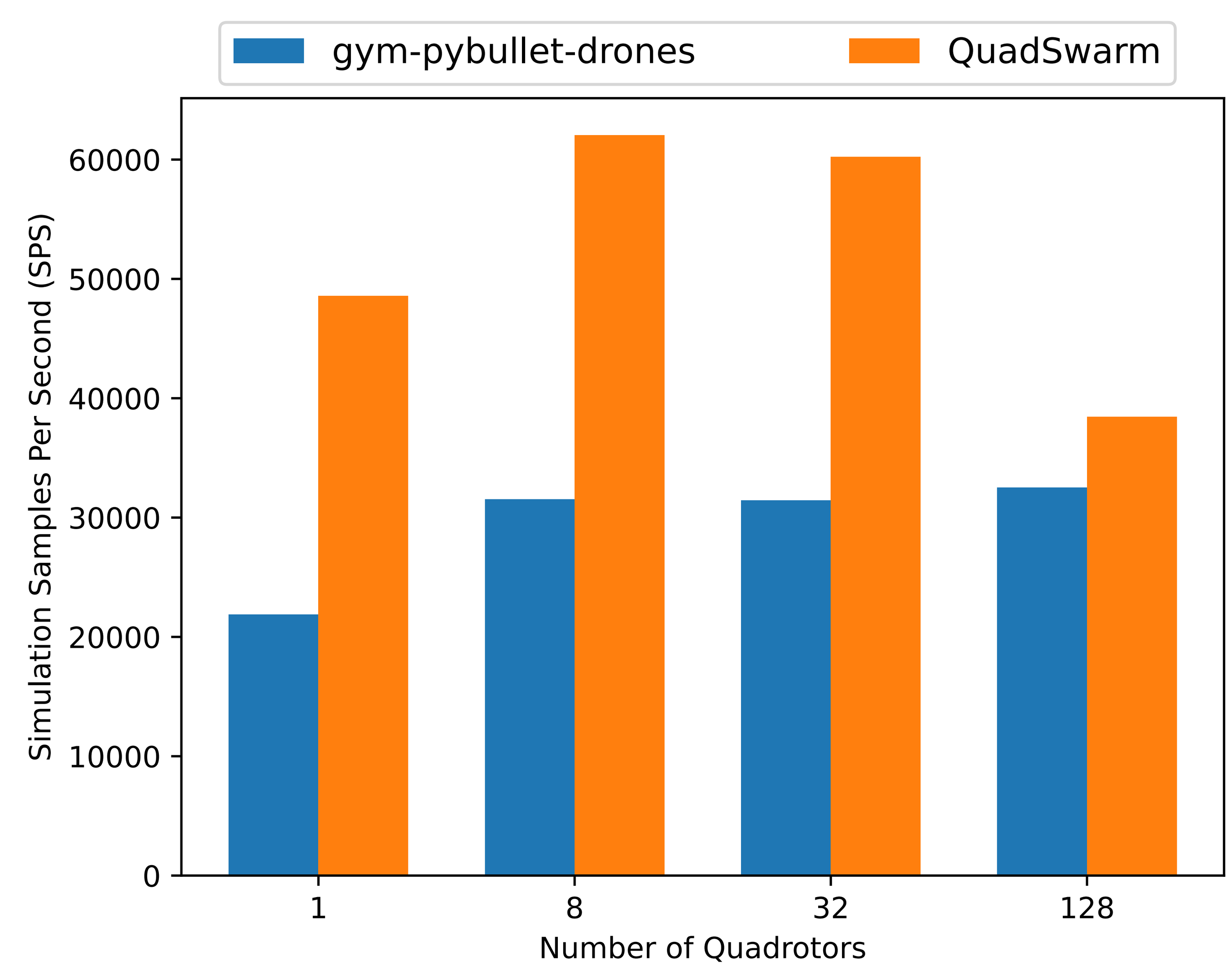
Support reward based on:
Distance to the goal 2) Linear velocity 3) Angular velocity
4) Actions 5) Change of actions 6) Rotation
7) Interaction with walls, ceiling, and ground 8) Interaction with other quadrotors

RL Library Interface

Integrated with Sample-Factory [1].
Supports PPO (single-agent) and IPPO (multi-agents)

Simulation Speed

To balance speed, readability, and flexibility:
1) Use Python to implement the minimum requirements of physics simulation and rendering
2) Use Numba to speed up physics simulations
3) Decouple rendering from physics simulations



Examples

QuadSwarm is used as the main simulation platform in two projects that demonstrated the transfer of learned control policies on single and multiple quadrotors.
For a single quadrotor [2], we showed how to learn a policy to stabilize multiple different quadrotors with domain randomization.
For multiple quadrotors [3], we showed how to learn a policy to control up to 128 quadrotors to approach their goals while avoiding collisions in diverse scenarios.

Parameter Table	
x	Position
g	Gravity vector
R	Rotation matrix
f	Total thrust vector
m	Mass
v	Linear velocity
ω; ω×	Angular velocity; Skew matrix of the ω
I	Inertia matrix
τ, τ_p, τ_{th}	Torque: total, along z-axis, produced by motor trusts
$\hat{u}$, $\hat{u_f}$	Rotor angular velocity: normalized, filtered
ϵ, α, k	fixed value
δ_{pos}, δ_{xi}	Relative position between quadrotors; Relative position to the goal
$\tilde{x_{iK}}$	Relative position between the quadrotor and its Kth nearest neighbor

References

[1] A. Petrenko, Z. Huang, T. Kumar, G. S. Sukhatme and V. Koltun, "Sample Factory: Egocentric 3D Control from Pixels at 100000 FPS with Asynchronous Reinforcement Learning," ICML 2020
[2] A. Molchanov, T. Chen, W. H¨onig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," IROS 2019
[3] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, "Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning, CoRL 2022.