Aerial Gym – Isaac Gym Simulator for Aerial Robots

Mihir Kulkarni Theodor J. L. Forgaard Kostas Alexis

***Abstract*—Developing learning-based methods for navigation of aerial robots is an intensive data-driven process that requires highly parallelized simulation. The full utilization of such simulators is hindered by the lack of parallelized high-level control methods that imitate the real-world robot interface. Responding to this need, we develop the Aerial Gym simulator that can simulate millions of multirotor vehicles parallelly with nonlinear geometric controllers for the Special Euclidean Group SE(3) for attitude, velocity and position tracking. We also develop functionalities for managing a large number of obstacles in the environment, enabling rapid randomization for learning of navigation tasks. In addition, we also provide sample environments having robots with simulated cameras capable of capturing RGB, depth, segmentation and optical flow data in obstacle-rich environments. This simulator is a step towards developing a – currently missing – highly parallelized aerial robot simulation with geometric controllers at a large scale, while also providing a customizable obstacle randomization functionality for navigation tasks. We provide training scripts with compatible reinforcement learning frameworks to navigate the robot to a goal setpoint based on attitude and velocity command interfaces. Finally, we open source the simulator and aim to develop it further to speed up rendering using alternate kernel-based frameworks in order to parallelize ray-casting for depth images thus supporting a larger number of robots.**

***Index Terms*—Parallelized aerial robot simulation, Parallelized geometric control, Environments for reinforcement learning**

I. Introduction

In recent years, aerial robots have gained significant attention in various applications, ranging from search and rescue missions to automated site inspections. These environments can present complex geometries and may present challenges to the perception systems owing to darkness, airborne particles, geometric self-similarities, etc. Due to the highly complex nature of the problem, there is a renewed interest in utilizing learning-based navigation methods to operate in these settings. The development of such methods is an intensive data-driven process, necessitating highly parallelized simulation environments. Despite the existence of powerful simulators, the full potential of these tools remains untapped due to the absence of parallelized control methods that imitate the real-world robot interface and allow users to provide high-level commands to the robot.

We address this gap by harnessing the capabilities of NVIDIA Isaac Gym to design an aerial robot simulator that enables the parallel simulation of thousands of multirotor vehicles. We integrate nonlinear geometric controllers for the Special Euclidean Group $\mathsf{SE}(3)$ [1] to achieve accurate attitude and velocity tracking in the vehicle frame, or position tracking in the inertial frame, while taking advantage of GPU parallelization. We rely on the underlying physics interface to simulate forces, torques, and collisions, while modeling the robot as a rigid body.

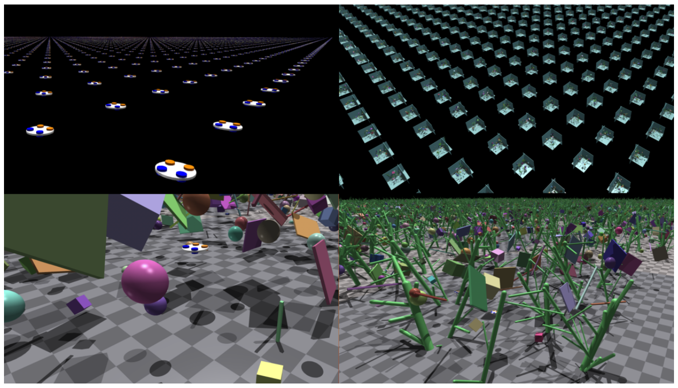
Fig. 1. Visualization of the Aerial Gym simulator with multiple simulated multirotor robots. Instances show – in clockwise order – the simulation of the robots in obstacle-free environments, a zoomed-out view of separated box-like environments, as well as cluttered environments for navigation consisting of randomly distributed obstacles of different shapes.

Moreover, we develop interfaces for efficiently managing a large set of objects in the environment, defined using Universal Robot Description Format (URDF) files. This facilitates rapid randomization of the environment for learning in cluttered settings. Our simulator provides sample environments containing robots equipped with simulated cameras capable of capturing RGB, depth, segmentation, and optical flow data at thousands of frames per second (a total of ~ 1800 fps for 2^{10} robots). These powerful tools and simulators mark a significant step towards the development of a highly parallelized aerial robot simulation environment with geometric controllers at a large scale, coupled with a customizable obstacle randomization functionality. We release our simulator as open-source software at https://github.com/ntnu-arl/aerial_gym_simulator and shall continue its development by leveraging alternative kernel-based frameworks to parallelize ray-casting for depth images, ultimately supporting simulation of an even larger number of robots. We will also release example scripts that show the robots interfaced with commonly used learning frameworks (e.g., cleanRL [2] or rl-games [3]) to learn to reach a target setpoint using attitude control in an obstacle-free environment.

In the remainder of this paper, Section II presents the related

All authors are affiliated with the Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

work, followed by the design of the proposed simulator for aerial robots in Section III. Sections IV and V detail the overall design, the parallelized geometric controller and the obstacle asset management functionality respectively. Section VI benchmarks the performance of the simulator, followed by conclusions in Section VII.

II. Related Work

Several simulators have been developed to simulate aerial robots for a variety of tasks such as navigation, mapping, and control. RotorS [4] is a Gazebo-based [5] simulator that provides a variety of multirotors with RGB-D sensors. Airsim [6] is a photo-realistic simulator built on Unreal Engine. The simulator supports both hardware- and software-in-the-loop simulations for a limited number of simulated robots. Flightmare [7] offers the functionality to simulate a large number of robots in parallel. However, the robot dynamics simulated by Flightmare are calculated on parallel threads on the CPU, limiting the number of robots that can be simulated. Flightmare uses the Unity rendering engine, allowing high-fidelity graphics simulation.

NVIDIA's Isaac Gym [8] provides GPU-accelerated highly parallelized simulation functionality for robot learning tasks. This simulator is being extensively used to simulate articulated and multi-linked robots [9]. Some simplified simulation environments for aerial robots have also been developed to work with it [8], however, the models either lack fidelity, or they only provide interfaces to command motor forces but ignore the effect of torque generated by the motor on the body. They also do not support any other higher-level reference tracking interfaces. Accordingly, to provide the simulation capability exploiting the GPU, our simulator is built upon the Isaac Gym simulator. Then, we further provide GPU-based geometric attitude and velocity controllers thus supporting a wider range of control inputs enabling the simulator's utility to a larger set of use cases and the capability to train for real-world deployments with potentially reduced sim-to-real gap.

III. Simulator Design

We build the proposed Aerial Gym simulator utilizing the tensor-based parallelization provided by NVIDIA Isaac Gym simulator [8]. We design our simulator with the appropriate interfaces to imitate standardized reinforcement learning environments [10] in order to facilitate easy extension of commonly-used learning-based algorithms for robot navigation. We design a generalized asset management class, that allows a user to load URDF files describing obstacles from a folder structure. We define an asset - in line with NVIDIA Isaac Gym - to represent a mesh entity in simulation that may contain links, joints, and other physical properties. In our case, assets are considered to be represented by URDF files, however, extending this to other supported formats is trivial. The selection and loading of these files happen in a randomized fashion per environment, where a predefined number of different obstacle meshes per custom-defined class

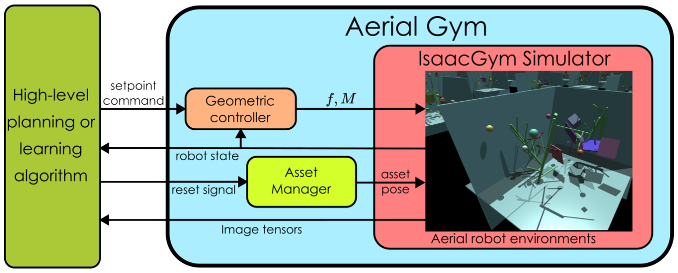

Fig. 2. Block diagram of the Aerial Gym simulator with the components to control the simulated robots and manipulate and randomize the simulated obstacles (also called assets) in multiple parallel environments.

of obstacles are picked randomly to be included in a simulation environment. This allows various environments to have sufficient diversity and prevent the reuse of identical obstacle meshes across environments. The robots simulated in obstacle-rich environments can also be equipped with camera sensors capable of capturing RGB, depth, segmentation, and optical flow images. The position and orientation of the camera sensors with respect to the robot are configurable and may be randomized. We aggregate images into a consolidated tensor and provide direct access to this tensor. In addition, we adapt nonlinear geometric controllers for aerial robots and – importantly – parallelize the controllers to be run on the GPU. This allows for the capability of providing high-level input commands to the robots, relying on interfaces that are available on commonly used flight controllers for multirotor aerial vehicles [11]. The structure of the simulator is shown as a block diagram in Figure 2, where the interaction between various modules is highlighted. A high-level planner or a learning-based framework can access the robot state which includes the position, orientation, linear velocity, and angular velocity of the robot. Access is also provided to the image tensors from the simulated sensors onboard the robot. Data from these sensors can be utilized by user-provided high-level planning or learning methods for navigation tasks or simulated data collection. In addition, access to the state of each obstacle is made available to the user as privileged information for training learning-based methods. As an additional contribution, we provide example scripts to procedurally generate URDF models of simplified multi-linked tree-like objects that can directly be added to the simulator for training learning-based methods to navigate forest-like environments. These procedurally generated trees have configurable length, diameters and branching factors, allowing the users to randomize the generated meshes to learn collision avoidance in diverse sets of environments.

IV. Parallelized Geometric Controller on $\mathsf{SE}(3)$

We develop the parallelized controller on $\mathsf{SE}(3)$ based on the work of [1]. The inertial reference frame is denoted as $\mathcal{E}$ with basis vectors $\{\vec{e}_1, \vec{e}_2, \vec{e}_3\}$ and a body-fixed frame $\mathcal{B}$ with basis vectors $\{\vec{b}_1, \vec{b}_2, \vec{b}_3\}$. We additionally define a vehicle frame $\mathcal{V}$ with the basis vectors $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$ that is yaw-aligned with the body-fixed frame, and having the $x - y$ plane parallel to the inertial frame. We define

$m \in \mathbb{R}$	the total mass
$g \in \mathbb{R}^3$	the gravity vector
$\psi \in \mathbb{R}$	the current yaw angle of the robot
$\phi_d \in \mathbb{R}$	the reference roll angle
$\theta_d \in \mathbb{R}$	the reference pitch angle
$\dot{\psi}_d \in \mathbb{R}$	the reference yaw rate
$J \in \mathbb{R}^{3\times3}$	the inertia matrix with respect to the body-fixed frame
$R \in \mathsf{SO}(3)$	the rotation matrix from the body-fixed frame to the inertial frame
$R_d \in \mathsf{SO}(3)$	the rotation matrix from desired body-fixed frame to the inertial frame
$\Omega \in \mathbb{R}^3$	the angular velocity in the body-fixed frame
$\Omega_d \in \mathbb{R}^3$	the desired angular velocity in the desired body-fixed frame
$v \in \mathbb{R}^3$	the velocity vector of the center of mass in the vehicle frame
$v_d \in \mathbb{R}^3$	the desired velocity vector of the center of mass in the vehicle frame
$f \in \mathbb{R}$	the total thrust magnitude along the $-\vec{b}_3$ axis
$M \in \mathbb{R}^3$	the total moment vector in the body-fixed frame.

The position of the aerial robot is defined by the location of the center of mass and the attitude is expressed with respect to the inertial frame. Due to the underactuated nature of quadrotors (and most other multirotor systems) asymptotic output tracking of both attitude and position is not possible. Two flight modes are detailed in this section. We consider the control of the robot using either a) the attitude-controlled flight mode or b) velocity-controlled flight mode, and provide adapted controllers for the same.

A. *Attitude-Controlled Flight Mode*

To imitate widely available real-world control interfaces, we consider ϕ_d, θ_d, $\dot{\psi}_d$ and f as command inputs to the attitude controller and calculate the desired body-fixed frame orientation, R_d, and the desired angular velocity, Ω_d, as below:

$$R_d = R_z(\psi)R_y(\theta_d)R_x(\phi_d) \tag{1}$$

$$\Omega_d = \begin{pmatrix} 1 & 0 & -\sin\theta_d \\ 0 & \cos\phi_d & \sin\phi_d\cos\theta_d \\ 0 & -\sin\phi_d & \cos\phi_d\cos\theta_d \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \dot{\psi}_d \end{pmatrix} \tag{2}$$

where $\mathbf{R}_j(\eta)$ $(j = x, y, z)$ denotes the rotation matrix for a rotation around the j-axis by η degrees. The attitude tracking error $e_R \in \mathbb{R}$ is expressed as:

$$e_R = \frac{1}{2}(R_d^T R - R^T R_d)^\vee, \tag{3}$$

where the *vee map* $^\vee : \mathfrak{so}(3) \to \mathbb{R}^3$ maps skew-symmetric matrices in $\mathbb{R}^3$. The angular velocity error $e_\Omega \in \mathbb{R}^3$ is expressed as

$$e_\Omega = \Omega - R^T R_d \Omega_d. \tag{4}$$

The nonlinear controller for the attitude-controlled flight mode is defined as

$$M = -k_R e_R - k_\Omega e_\Omega + \Omega \times J\Omega, \tag{5}$$

where k_R and k_Ω are diagonal matrices with positive entries. The final term from the corresponding equation in [1] is dropped, similar to the implementation in [4].

B. *Velocity-Controlled Flight Mode*

Similarly, a nonlinear controller for the velocity-controlled flight mode is utilized. An arbitrary velocity tracking command ${}^{\mathcal{V}}v_d \in \mathbb{R}^3$ is given, with the desired yaw-rate $\dot{\psi}_d$. The commanded acceleration vector can be calculated from the velocity tracking error as:

$$a_d = k_v(v_d - v), \tag{6}$$

where k_v is a diagonal matrix with positive entries. From this, the total thrust and the commanded tilt angles can be deduced as:

$$f = (a_d + mg) \cdot {}^{\mathcal{V}}\vec{b_3} \tag{7}$$

$$\phi_d = \operatorname{atan2}(-a_{d,y}, \sqrt{a_{d,x}^2 + a_{d,z}^2}) \tag{8}$$

$$\theta_d = \operatorname{atan2}(a_{d,x}, a_{d,z}), \tag{9}$$

where $a_{d,j}$ $(j = x, y, z)$ is the $j-$component of a_d, and ${}^{\mathcal{V}}\vec{b_3}$ is the transformed coordinates of $\vec{b_3}$ in $\mathcal{V}$. The reference tilt angles and reference yaw rate can then be tracked by the low-level attitude controller described above. The force f and torque M are applied to the simulated rigid-body robot in Isaac Gym using the underlying physics engine.

V. Asset Manager

To robustly train navigation policies using exteroceptive sensor data, environments containing obstacles and general clutter need to be created. Since the Isaac Gym simulator [8] provides tensor-level access to the position, orientation, and linear/angular velocities of each object in the simulator, we exploit this feature to build classes to easily modify each environment to randomize the obstacles. The simulator allows us to separately consider each individual environment, where collisions can be separated and masked between various entities. However, modifying each obstacle in the environment can be a cumbersome process. To avoid this, we develop a tensor-based asset manager to easily configure, randomize and manipulate the objects that are created in the simulator. Various objects are categorized into user-defined classes (e.g., thin obstacles, large trees, geometric shapes, etc.) based on their use case. Multiple URDF files for each class are stored in a folder structure named after the name of the associated class of objects. A configuration file describes the number of obstacles from each class to be created in each environment. The asset manager randomly picks (with replacement) the required number of

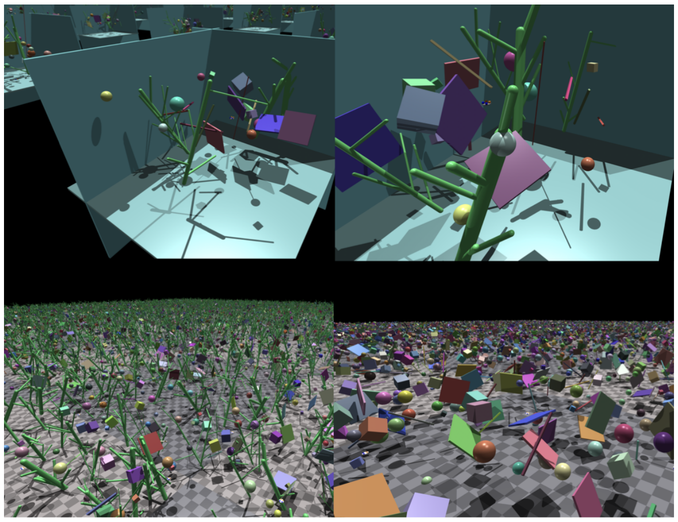
Fig. 3. Examples of different environments configured. The top row shows bounded environments, with the front, top, and left walls removed for visualization. The bottom row shows different configurations for multiple environments without walls.

URDF files separately for each environment. This allows the user to populate a directory with a large number of models, of obstacles with various types, shapes, and sizes, ensuring that there is randomization across various environments. This prevents learning methods from overfitting to specific types of obstacles. The asset manager is also utilized when the environments are reset as it samples a random position and orientation for each obstacle in the environment that is to be reset. The minimum and maximum bounds for the position can be set as a fraction of the environment bounds, while the orientation bounds can be set as numerical values corresponding to euler angles. The randomization in specific dimensions can be prevented by assigning a constant value to specific dimensions for chosen asset types. This allows placing obstacles at predefined locations across environments. While we provide only randomization for position and orientation for static obstacles, extending it to dynamic obstacles is trivial. The asset manager uses the NVIDIA Isaac Gym API and provides an easier interface to easily add segmentation labels to different classes of obstacles. Some examples of randomized environments generated with the help of the asset manager are shown in Figure 3.

VI. Benchmarks and Evaluation

To benchmark the performance of the simulator we simulate environments (without obstacles) with 2^{17} robots and command a set attitude or velocity. The parallelized simulation, including that of the implemented controllers, allows the Aerial Gym simulator to simulate an aggregate of over 3.8×10^6 steps per second, with each step corresponding to 10 ms of simulated time. Effectively, the data generated for the robot simulation from the simulator is equivalent to obtaining a speedup of 3.8×10^4 as compared to a real-time run on a single robot. However, the addition of camera sensors restricts the number of robots that can be used. The Isaac Gym API also does not allow direct access to a consolidated camera tensor leading to an overhead to read each image separately. We simulate 2^{10} robots with a depth camera with a resolution of 270×480 pixels and can render a total of up to $1,800$ frames per second. Each of these tests was performed on a workstation with $2\times$ NVIDIA RTX 3090 GPUs and an AMD Ryzen Threadripper PRO 3975WX CPU with 32-Cores.

VII. Conclusion

In this work, we presented the Aerial Gym simulator for aerial robots, capable of parallelly simulating thousands of flying robots. We adapted a nonlinear geometric controller to work with parallelly simulated robots to provide high-level interfaces that allow the imitation of real-world control interfaces. Asset management functionality is packaged to allow randomization of obstacles across environments and the creation of custom user-defined obstacle classes for effective handling of each kind of obstacle. Finally, we open-source our simulator to enable the aerial robotics community to leverage the benefit of parallelized simulation frameworks. Future developments on this simulator includes the utilization of alternate kernel-based ray-casting frameworks allowing a rapid speedup in rendering depth images with a higher number of robots. The software for the simulator is made available at https://github.com/ntnu-arl/aerial_gym_simulator.

References

[1] T. Lee, M. Leok, and N. H. McClamroch, "Control of complex maneuvers for a quadrotor uav using geometric methods on se(3)," 2011.

[2] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[3] D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning," https://github.com/Denys88/rl_games, May 2021.

[4] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-923

[5] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[6] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[7] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Conference on Robot Learning*, 2020.

[8] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[9] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," 2022.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[11] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.