International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 07 | July - 2022 Impact Factor: 7.185 ISSN: 2582-3930

Robotic Automation Using PyBullet

Dhanesh Amin[1], Nihar Kadam[2], Shubhangi Mahadik[3]

[1] Student, MCA Bharti Vidyapeeth Navi Mumbai, India
[2] Student, MCA Bharti Vidyapeeth Navi Mumbai, India
[3] Asst. Professor, MCA Bharti Vidyapeeth

Abstract - Physics-based simulations of passive phenomena such as substances and liquids have become almost ubiquitous in the industry. However, the introduction of physically simulated characters was more modest. Modelling the movements of humans and animals remains a difficult problem, and there are currently few ways to simulate the different behaviours shown in the real world. Permanent challenges in this area include generalization and manageability. While manually designed controller-based methods have produced compelling results, the availability of human insights limits new skills and the ability to generalize to new situations. Humans can perform a variety of skills themselves, but it is difficult to clarify the internal strategies underlying such skills, and it is even more difficult to code them into a controller. Mobility is another obstacle that has hindered the adoption of simulated characters. Creating motion for a simulated character is still notorious for being difficult, and the current interface does not provide users with an effective means of deriving the desired movement from the simulated character. Reinforcement learning offers a promising approach to motion synthesis. This approach reduces the need for human insight by learning that agents perform different skills through trial and error.

***Key Words*:** Physics-based simulation, Reinforcement Learning, motion synthesis, Simulated characters

1. INTRODUCTION

Bullet is a physics engine that simulates collision detection and soft and rigid body dynamics. Bullet Physics Library is free opensource software and is subject to the terms of the zlib license. The main task of the physics engine is to perform collision detection, resolve collisions and other constraints, and provide updated world transformations for all objects. pybullet is an easy-to-use physics simulation Python module for robotics, games, visual effects, and machine learning. You can use pybullet to load articulated bodies from URDF, SDF, MJCF, and other file formats. pybullet provides forward dynamics simulation, inverse dynamics computation, forward and backward kinematics, collision detection, and ray crossing queries. The Bullet Physics SDK contains examples of Pybullet robots such as simulated Minotaur quadrupeds, humanoids that perform TensorFlow inferences, and KUKA arm grab objects. Multi-body, rigid, and deformable bodies with reduced coordinates are handled by the integrated LCP constraint solver, similar to the articulated island algorithm for this task. It uses the Articulated Body algorithm to create linear forward time dynamics and the Solver A matrix.

2.1 METHODOLOGY

The simulated robot described in the URDF file has a base and optionally a link connected by a joint. Each joint connects the parent link to the child link. The root of the hierarchy has a single root parent called the base. The base can be completely fixed with 0 degrees of freedom or completely free with 6 degrees of freedom. Each limb is connected to the parent's limb with a single joint, so the number of limbs is equal to the number of limbs. Regular links have a link index in the range [0..getNumJoints ()]. Since the base is not a regular" link", we will use Rule 1 as the link index. The articulated frame uses the rule that it is represented relative to the inertial frame of the parent center of gravity aligned with the principal inertial axis.[2]

Robot simulators are critical to academic research, education, and the development of safety-critical applications. Reinforcement learning environments (simple simulations combined with problem specifications in the form of reward functions) are also important for standardizing the development (and benchmarking) of learning algorithms. However, full-scale simulators usually lack portability and parallelism. Conversely, for toy-like problems, many reinforcements learning environments trade high sample throughput and realism. Public datasets have brought great benefits to deep learning and computer vision, but there is still a lack of software tools to develop control theory and reinforcement learning approaches at the same time and make fair comparisons. This article proposes an open source OpenAI-Gym-like environment for the movement of humanoid robots based on the Bullet physics engine. As far as we know, its multi-agent and vision-based reinforcement learning interface and support for realistic collision effects make it the first of its kind.

2.2 APPROACH

About PPO Algorithm

The policy gradient method is the basis of a recent breakthrough that uses deep neural networks to control everything from video games to 3D movement to Go. However, it is difficult to get good results from the policy gradient method. This is because it is sensitive to the choice of step size and is too small to progress desperately. If it is too large, the signal can be overwhelmed by noise and can cause catastrophic performance degradation. Also, sampling efficiency is very low, often requiring millions (or billions) of time steps to learn a simple task.[2]

Researchers have attempted to address these deficiencies with approaches such as TRPO and ACER by limiting or optimizing the scope of policy updates. These methods have their own trade-offs. ACER is much more complex than PPO and requires out-of-policy fixes and replay buffer code, but is slightly better than the Atari benchmark PPO. TRPOs are useful for ongoing control tasks, but also for algorithms that share parameters between policies and value functions, and for solving problems in Atari and other areas where visual input is important. Not easily compatible with additional losses.

PPO

With supervised learning, you can be confident that you can get good results with relatively few hyperparameter adjustments by simply implementing a cost function and then performing the steepest descent method. The road to success in reinforcement learning is not so clear. The algorithm has many moving parts that are difficult to debug and require considerable tweaking to get good results. PPO attempts to calculate updates that minimize cost functions at each step, ensuring that deviations from previous guidelines are relatively small, for ease of implementation, rehearsal complexity, and coordination. Balance ease.

Earlier, we detailed a variant of PPO that uses an adaptive KL penalty to control policy changes at each iteration. The new variant uses a new objective function that is not normally found in other algorithms.

This goal simplifies the algorithm by implementing a method of performing confidence interval updates compatible with stochastic gradient descent and removing the KL penalty for requesting adaptive updates. In testing, this algorithm showed the best performance for continuous control tasks and was much easier to implement, but almost reached Atari's ACER performance.

In reinforcement learning terminology, a policy is a mapping from action space to state space. You can think of this as an instruction to take an action based on the state of the environment in which the RL agent is located. When we talk about agent ratings, we generally mean a rating policy feature to see how well an agent works according to a particular policy. The policy gradient method plays an important role here. If the agent" learns" and does not really know which action will give the best results in that state, it does so by calculating the gradient of the policy. It works like a neural network architecture and the output gradient is: H. Logs are created with the probability of action in a particular state related to the parameters of the environment, and policy changes are reflected based on the gradient.

While this proven method works well, the main drawback of these methods is their low sample efficiency and their high sensitivity to hyperparameter adjustments such as choices such as step size and learning rate. Unlike supervised learning, which has relatively few hyperparameter adjustments and guarantees a path to success or convergence, reinforcement learning is much more complex due to the various moving

parts to consider. PPO is a cost function that balances key factors such as ease of implementation, ease of adjustment, sample complexity, and sample efficiency, ensuring that deviations from previous guidelines are relative. The purpose is to calculate updates at each step that minimizes. low. PPO is actually a policy gradient method that also learns from online data. Ensure that the updated policy is not much different from the old policy and reduce training differences. The most common implementation of PPO is through an actor critique model that uses two deep neural networks. One handles actions (actors) and the other handles rewards (criticisms). The mathematical equation of PPO is shown below:

$L^{CLIP}(\theta) = E\hat{}t[\min(r_t(\theta)A\hat{}t, \text{clip}(r_t(\theta), 1\ \varepsilon, 1 + \varepsilon)A\hat{}t)]$

$L^{CLIP}() = E^t[\min(r_t()A^t, \text{clip}(rt(), 1, 1+)A^t)]$

θ is the policy parameter

$_tE^t$, denotes the empirical expectation over timesteps

r_trt, is the ratio of the probability under the new and old policies, respectively

$_tA^t$, is the estimated advantage at time tt

ε is a hyper parameter, usually0.1or0.2

[1][5]

The following important inferences can be drawn from the PPO equation:

This is a strategic gradient optimization algorithm. H. At each step, there is an update to an existing strategy that seeks to improve certain parameters. This ensures that the update is not too big. H. The old policy isn't much different from the new policy (this is basically done by" clip"). The update range is very narrow.

The profit function is the difference between the future discount amount of reward for a particular state and action and the value function of that policy.

Each update uses the severity sample ratio or the likelihood ratio under the old and new policies. is a hyperparameter that indicates the boundaries of the areas that are allowed to be updated. This is how the working PPO algorithm looks, in its entirety when implemented in Actor-Critic style:

What you can observe is that a small batch of observations is used for updates and then discarded to integrate a new batch of observations, also known as a" mini-batch". The updated policy is clipped to a small area to disallow large updates that can be irreparably harmful. In other words, PPO behaves exactly like any other policy gradient method in the sense that it calculates the output probability of the forward path based on various parameters and calculates the gradient to improve the backward path determination or probability. To do. This includes the use of important sample rations such as its predecessor TRPO. However, it also guarantees that the old and new policies are at least in a certain proximity (indicated by) and do not allow very large updates. It has become one of the most widely used policy optimization algorithms in reinforcement learning.

3. CONCLUSIONS

This article introduced OpenAI Gymlike, an open source multiquadcopter simulator written in Python and based on the Bullet Physics engine. The striking and innovative features of our proposal compared to similar existing tools are (i) a more modular and sophisticated implementation of physics, (ii) visual gym observation, and (iii). A multi-agent interface for reinforcement learning. Using the example of trajectory tracking and target velocity input, we showed how to use the Gympybullet drone for low- and high-level control. It also showed how to leverage work in separate workflows for single-agent RL and multi-agent RL, based on a state-of-the-art learning library. We believe that our work will help bridge the gap between reinforcement learning and control research, and help the community develop realistic MARL applications for aerial robotics.

ACKNOWLEDGEMENT

I have great pleasure to express my gratitude to all those who have contributed to and motivated me during my project work. I want to thank my teachers, and parents for motivating me. It's because of their constant support I was able to get a job in this wonderful company and become a part of this great project.

REFERENCES

1. https://openai.com/blog/openai-baselines-ppo/
2.https://github.com/bulletphysics/bullet3/blob/master/docs/Bullet_User_Manual.pdf (PyBullet)
3. https://arxiv.org/pdf/2103.02142v3.pdf (Quadcopter)
4. https://arxiv.org/pdf/1804.02717v3.pdf (DeepMimic)
5. https://openreview.net/pdf?id=r1etN1rtPB (PPO)