Modular Robot Snake Training Environment and Reinforcement Learning of Simple Locomotion

Tomas Musil

December 18, 2022

1 Background

Reinforcement learning (RL) has widely been used in creating control policies for robots for a wide variety of problems, such as walking of humanoid robots [1], spider-like robots [2] or quadruped robots [3]. In this project, I concerned myself with using a RL algorithm to teach locomotion of a modular snake robot.

The modular robot taught in simulation in this project is based on the hardware realization of a real modular snake robot. The robot system was constructed by me and my 3 other colleagues at the Czech Technical University and is made of very affordable parts, open-source design and simple communication and software stack. It is also a reason for me taking this course, where I intended to learn more about RL and apply it to creating a RL-based controller for this robot system.

Reinforcement learning is particularly of interest for robots for which it is hard to create a model by humans using mathematics.

2 Problem Definition

The goal of this project is to:

- Construct a custom gym environment which would allow effortless definition and spawning of different configuration of the modular robot which could be transferred to the real robot, and have multiple terrains on which to learn movement.
- Train moving in one type of terrain for multiple robot configurations to see how the number of degrees of freedom of observation and actions affect the training.
- Train on two types of terrain using two different RL algorithms. First a simple uneven terrain simulating for example the floor of a forest. Secondly on an environment with stairs.

3 Methods

3.1 Custom AI Gym Environment Construction

For the purpose of this project, I developed a new AI gym environment from scratch.

- **General description:** The environment uses the PyBullet (TODO) physics engine for its physics simulation. The new environment constructor takes as input the desired configuration of modules in the robot (specified by a string of tuples (module type, module orientation w.r.t. the previous module) and creates an URDF file which it uses to spawn the snake robot. Furthermore, the environment currently supports 2 different terrains on which the robot can learn to move. These terrains, which I created using the open-source software Blender and are shown in (TODO) are stairs, and a bumpy terrain which is randomly rotated around the z axis on every reset to simulate random bumpy terrain for the robot.
- **Observations:** The environment has $3 + n_{servo}$ continuous observations. These are an ensemble of roll, pitch, yaw of the robot (simulating measurements from an affordable IMU sensor, although without noise), and the current angle of every servo module in the robot.

- **Actions:** The environment has $n_{servo} + n_{drive}$ actions, with each servo having a position reference input (not velocities), and each drive module having a velocity reference input. The position and velocity control is done by pybullet for the robot joints.
- **Reward:** The reward differs for the 2 scenarios. For teaching to move forward (in x direction) on bumpy terrain, the robot gets a reward $r_{bumps} = 100 * dx - 50 * |dy|$ every timestep. For moving on stairs, the robot is motivated to lift its head to climb the stairs by adding a reward for moving in the z direction, so its reward is $r_{stairs} = 100 * dx - 50 * |dy| + 200 * dz$. I experimented with adding a penalty for rotation along the z axis to force the robot to move forward more quickly, but this proved to be counter productive.

3.2 Training Methods

To do this, I heavily utilized the open-source library Stable Baselines 3 [4]. First, I experimented with the algorithms from the library and found PPO [1] to work the best, A2C to work slightly for the bumpy terrain but not for stairs, and the other algorithms were not learning anything meaningful for the constructed environment. For training in both tasks, I decided to use the library TODO. Therefore, in this paper I showcase 2 experiments.

First experiment was comparing A2C and PPO on the same setup of the *bumpy terrain* scenario. I used the default hyperparameters and the train() function from stable-baselines3 and trained for 200 episodes of 3000 steps each.

Second experiment was simply learning to go up in the *stairs* scenario. I found that out of the algorithms with default hyperparameters, only PPO was able to learn to go up the stairs, and therefore only showcase PPO. I also used the train() function from stable-baselines3 and trained for 300 episodes of 3000 steps each.

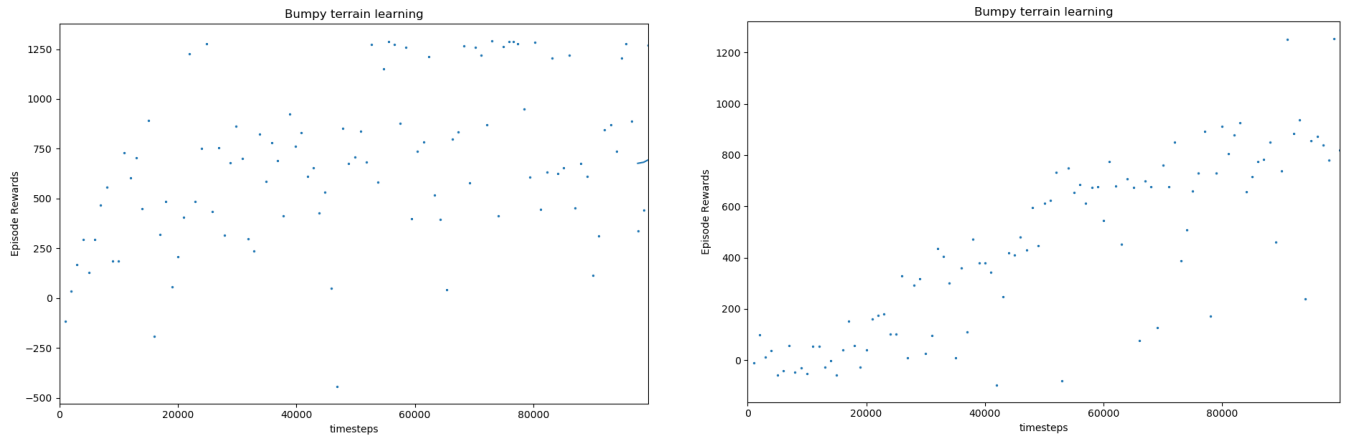
4 Results

The results of learning on *bumpy terrain* can be seen in TODO. PPO learned a policy much faster than A2C and TODO.

The results of learning on *stairs* using PPO can be seen in TODO.

Furthermore, I note these findings from the experiments I ran:

- TODO
- TODO



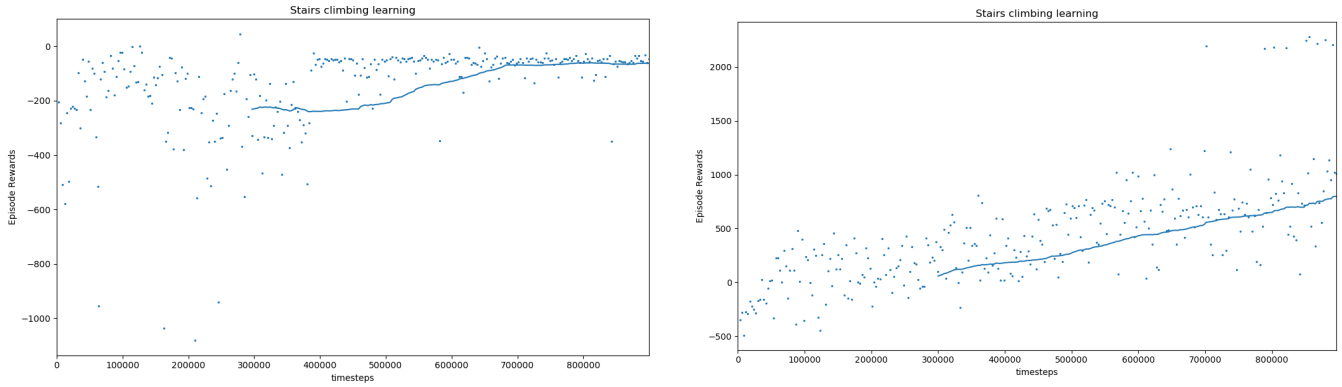
(a) A2C (b) PPO

Figure 1: Put your caption here

[1] VIDEO?

5 Discussion

Mention - happy that I was able to make it learn on the stairs and on the bumps. Fun would be putting it on real HW and testing its movement and trying to bridge the reality gap (different masses, powers of motors, delays, springiness,



(a) A2C

(b) PPO

Figure 2: Put your caption here

...) which would take a lot of effort, maybe for one full masters thesis.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 07 2017.

[2] Nicolas Manfred Otto Heess, TB Dhruva, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyun Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *ArXiv*, abs/1707.02286, 2017.

[3] N. Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *ArXiv*, abs/2109.11978, 2021.

[4] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.