Modular Robot Snake Training Environment and Reinforcement Learning of Simple Locomotion

Tomas Musil

December 19, 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 in various terrains.

The modular robot taught in simulation in this project is based on the hardware realization of a real modular snake robot, shown in Figure 1. The robot system was constructed by me and my 3 other colleagues at the Czech Technical University and features very affordable parts, open-source design and simple communication and software stack.

RL is particularly of interest for complicated robots for which it is hard to create a controller by hand. As such, I believe RL to be well suited for developing controllers for this particular robotic system, since it is physically capable of for example climbing up stairs, but I see no way of designing a controller for that without RL. Furthermore, it would be difficult to construct a controller every time the user wants to change the robot configuration, and having automatic controller generation could be very useful.

Figure 1: The real hardware of the modular robot system for which I constructed the simulation in this project, in configuration with a head module followed by a wheel module, two servo modules to allow bending up and down and steering, and one more wheel module

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.

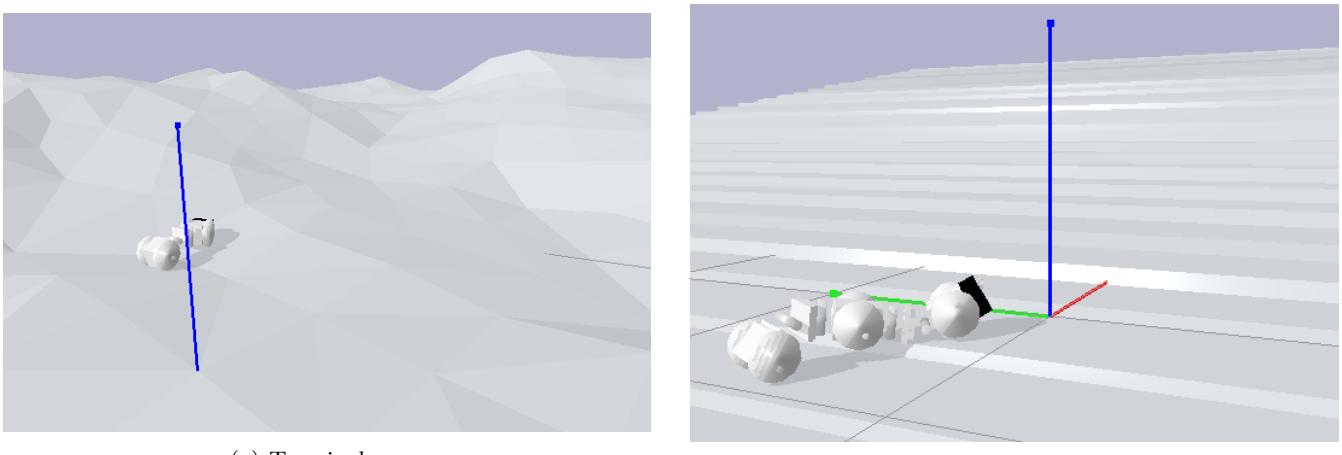
- Train on two types of terrain using two different RL algorithms. First a simple uneven terrain simulating for example the floor of a forest. Secondly on an environment with stairs.

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 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 Figure 2 are *stairs*, and *bumps* 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. Additionally, in both terrains, the agent gets a large fixed reward if it reaches the end of the terrain in the x direction.
- **Reality gap:** The masses and moments of inertia of robot links, max speed of wheels and max force and speed of the servo modules were just approximately estimated. Also the real-world servos feature a spring for SEA actuation, which is not modeled in the simulation servos. Therefore, this simulation is more of a proof of concept and would need additional work to be deployable 1:1 on the real hardware.



(a) Terrain *bumps*

(b) Terrain *stairs*

Figure 2: The two learning scenarios. A configuration with 2 wheel modules and 2 servos is used for the *bumps* terrain and a longer configuration with 3 wheel modules and 4 servos is used for the *stairs* terrain. The *bumps* terrain is also randomly rotated around the vertical axis at every reset.

3.2 Training Methods

For training the robot on the 2 terrain types, I used the open-source library stable-baselines3, which allows easy implementation of RL algorithms such as A2C, PPO, DDPG etc. I experimented for some time with various algorithms, and found PPO and A2C to work best for this environment, which is why I chose them for the demonstration experiments. Both are using a Multi-Layer Perceptron (MLP) policy in all cases.

The first experiment I designed 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 100 episodes of 3000 steps each.

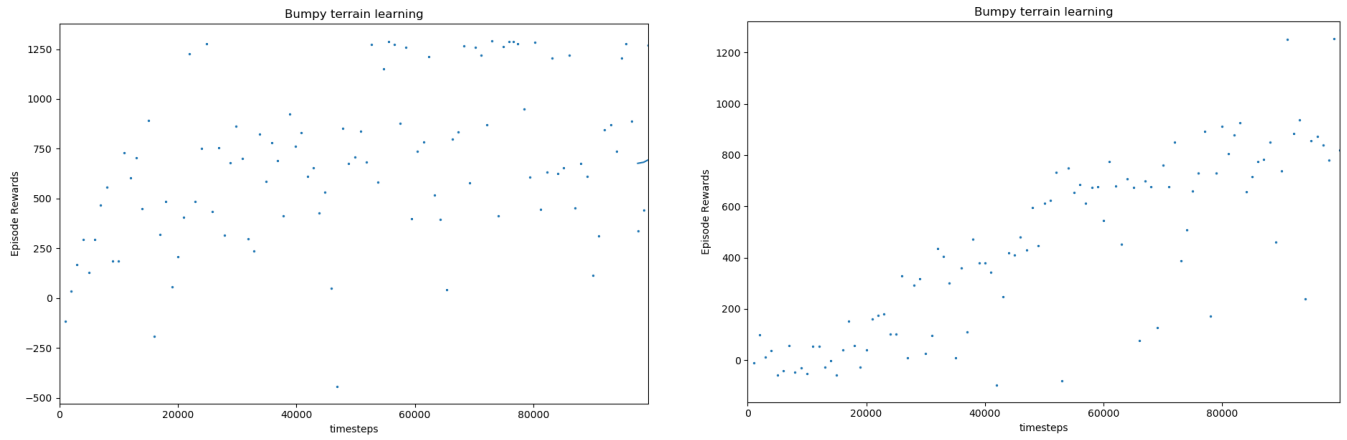
Second experiment was learning to go up in the *stairs* scenario. I also used the train() function from stable-baselines3 and trained for 300 episodes of 3000 steps each.

4 Results

The results of learning on *bumpy terrain* can be seen in Figure 3. In this experiment, PPO was learning more slowly than A2C, but A2C was somewhat inconsistent in the episodes, while PPO was steadily increasing the episode reward. Both methods learned fast compared to the *stairs* terrain, which is understandable, since this task is substantially easier.

The results of learning on *stairs* using PPO can be seen in Figure 4. In this experiment, PPO was able to gradually learn a stair climbing motion. A2C was able to not fall off the back side of the terrain, and slowly got better at not going backwards, but was unable unable to get over even the first step, which corresponds to the plaetau in the graph. A2C, in essence, learned not to fall down the stairs, but could not climb forward, whereas PPO managed to climb the stairs and even get to the end of the stairs (this corresponds to the high reward outliers near the end).

I have also published a video of the PPO agent after 300 episodes here: https://www.youtube.com/watch?v=A939OQfjrN8



(a) A2C (b) PPO

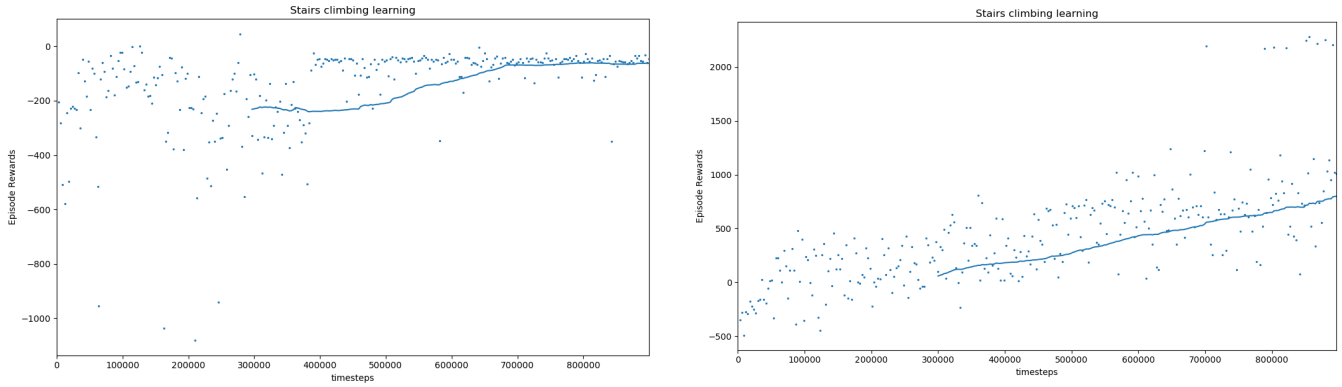
Figure 3: Results from learning PPO and A2C models on the *bumps* terrain for 100 episodes.

5 Discussion

I believe it is a great success that I managed to construct a working gym environment and implement the PPO algorithm for it and succesfully make the model learn a motion that can climb the stairs.

An interesting thing is why PPO learned more slowly than A2C in this example. From observing the two learning processess in the PyBullet GUI, it seemed to me that A2C does not produce such noisy action signals as PPO, meaning that PPO in the stable-baselines3 likely uses some action noise, while A2C does not. Due to time constraints, I could not have investigated this further.

As can be seen in the video in the Results section showing the PPO model scale stairs, the robot in that case produces a sort of "oscillatory" motion where it throws its body from side to side. I was worried if a simple MLP-policy would be sufficient to learn stair climbing and if it did not rather need a recurrent policy, as the robot would need to develop



(a) A2C

(b) PPO

Figure 4: Results from learning PPO and A2C models on the *stairs* terrain for 300 episodes.

some "gait" which would need some recurrence. However, it seems to me, that thanks to the complexity of the body of the robot, and since all joints of the servos are controlled AND observed, and multiple physics steps are happening between each environment step, the robot was able to form this kind of gait without any recurrence in the model itself. My explanation for this is that the MLP-policy model likely learned a gait such as (body bent left => bend body right and up, body bent right => bend body left and down). This seems fascinating, that the learning solved the recurrence problem in this way.

Future work would certainly be interesting. Firstly, I would like to work more closely with the real hardware and try running the learned policy on a real robot, and then seeing the discrepancies and working on bridging the reality gap as much as possible. Additionally, I would like to, in the future, add some form of touch/force perception in a physically realizable manner, for example using tiny bumpers/buttons in the front of the robot or on the servo module, or also measuring the current and velocity of the wheels to infer if they are touching ground or not. I believe that would make the robot learn to climb stairs/rocks even better.

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.