Balancing Upkie with PPO

Romain Gambardella, Philippe Gratias-Quiquandon

November 2024

1 Introduction

1.1 Presentation of Upkie

Upkie [1] is a bipedal robot with two wheels, featuring sagittal symmetry. Developed by Inria, this robot stands out due to its simple structure, making it an ideal platform for research in robotics, control, and reinforcement learning.

Upkie consists of a torso with a total of 6 joints, 3 on each side, and 2 wheels that are in contact with the ground (see Fig. 1). One of the challenges with this robot is maintaining continuous balance: unlike a quadruped robot, Upkie will fall and touch the ground if it is not continuously rebalanced. This means that before giving it commands to perform tasks, it must already have robust balance control.

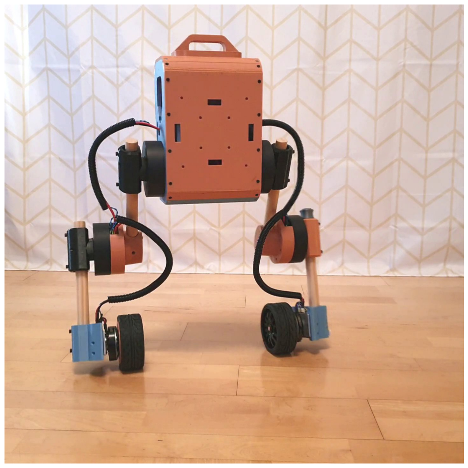


Figure 1: The Upkie robot, with 3 joints on each side, totaling 6 joints

1.2 Project Presentation

The goal of this project is to improve the robot's balance using reinforcement learning. We measure the robot's balance with an indicator called MSFOS, defined below. We use the PPO (Proximal Policy Optimization) algorithm [2] to control the robot's balance. The specific problem is to increase the force that can be applied for one second on the robot's sagittal plane before it falls. We will refer to this criterion as MSFOS, defined as follows for the rest of the project:

Definition 1.1. MSFOS is the maximum sagittal force (in N) applied to the Upkie robot between 1.5s and 2.5s of simulation, such that the robot remains standing until the 10th second of simulation, with a success probability greater than 2/3.

This MSFOS will be our main criterion throughout the project, which is why we have created a simulation environment that rigorously calculates the MSFOS using dichotomy. The algorithm used is presented below:

Algorithm 1 Calculate MSFOS using Dichotomy

```
Require: lower_bound, upper_bound, tolerance, num_trials
 1: while upper_bound - lower_bound > tolerance do
 2:     mid_force ← (lower_bound + upper_bound) / 2
 3:     success_probability ← EstimateSuccessProbability(mid_force, num_trials)
 4:     if success_probability > 2/3 then
 5:         lower_bound ← mid_force
 6:     else
 7:         upper_bound ← mid_force
 8:     end if
 9: end while
10: MSFOS ← (lower_bound + upper_bound) / 2
11: return MSFOS
```

We start by evaluating the performance of a simple linear return to have a comparison point with our successive improvements. Then, we evaluate a PPO policy trained to stay upright. We then try to improve our results using *Curriculum learning.* Additionally, we also tested some more advanced experiments on the *Upkie Servos* environment.

2 Results

We evaluate the MSFOS using the algorithm presented above.

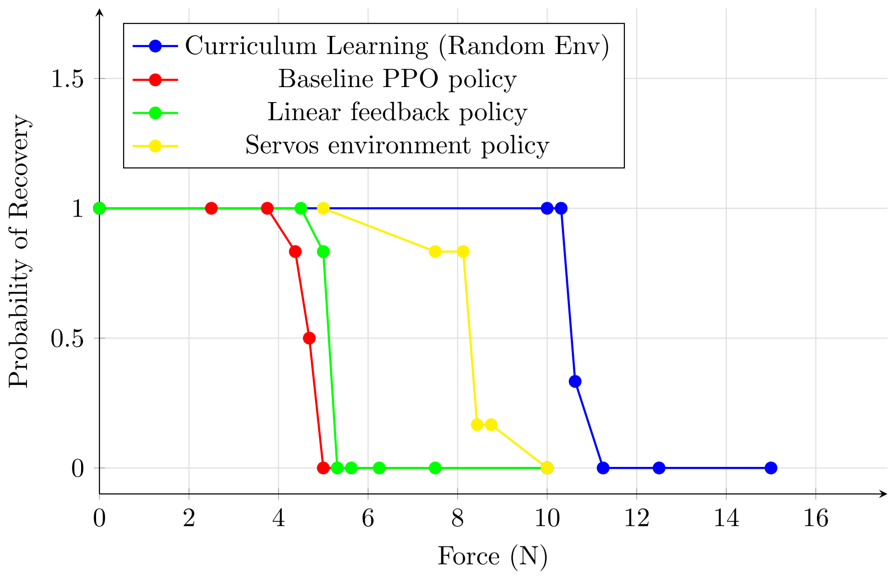


Figure 2: Probability of Balance Recovery with Respect to the Applied Force

Policy	**MSFOS (N)**
Baseline PPO Policy	4.3
Linear Feedback	4.5
Curriculum Learning	10.3
Servos environment	8.1

Table 1: Summary of MSFOS Values for different policies

3 Project Progress

3.1 Control with Linear Angle Return

The observations of the environment consist of four variables:

$$o = \begin{bmatrix} \theta \\ p \\ \dot{\theta} \\ \dot{p} \end{bmatrix}$$

where θ is the angle of the robot relative to the vertical ground, p is the average position between its two wheels. The action acts on the desired speed $\dot{p}$.

We start with a linear return on the observations, with a gain:

$$G = \begin{bmatrix} K_\theta & K_p & K_{\dot{\theta}} & K_{\dot{p}} \end{bmatrix}$$

$$\begin{tikzcd} {} \arrow[r, "o(t)"] & \boxed{\text{Controller}} \arrow[r, "u(t) = G^T.o(t)"] & \boxed{\text{System}} \arrow[r, "y(t)"] & {} \end{tikzcd}$$

We start by returning only the observation of the angle θ. This gives us an MSFOS of 4 N.

Next, we change the gain to take into account the entire observation vector (this gives a PD controller). With $G = [20.0, 1.0, 1.0, 0]$, we achieve an MSFOS of $\boxed{4.5N}$.

3.2 Reinforcement Training: PPO

PPO [2] is a reinforcement learning algorithm that prevents the action function from updating too quickly, making the learning more robust.

A pre-trained policy allows for an initial test: the achieved MSFOS is $\boxed{4.3N}$.

We perform training from scratch with the provided code, and we find a similar MSFOS of $\boxed{4.5N}$.

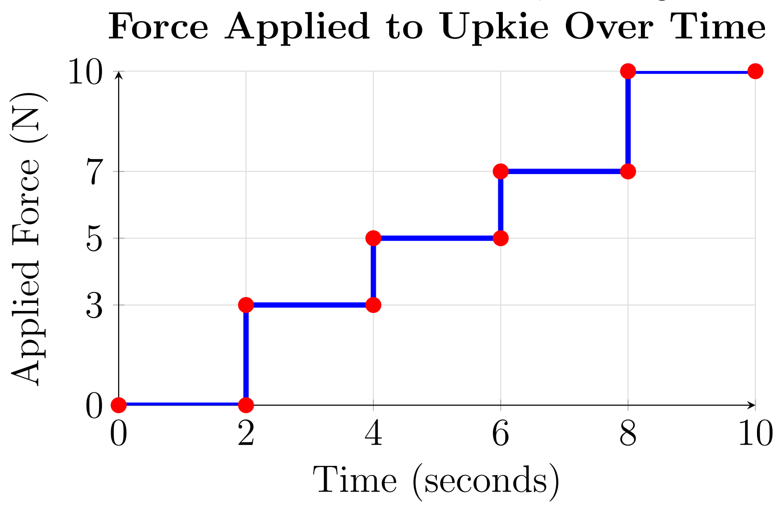
3.3 Curriculum Learning

We create a new environment (using a Wrapper) that exerts a force on the robot, increasing as the time the robot remains standing increases. The idea is that the robot learns to stand, then learns to stay upright when a force is applied, and we hope this will increase the MSFOS.

For this, we conducted several trials explained below:

3.3.1 Trial 1: Fixed Environment

We exert a constant force on the robot's torso, increasing over time.

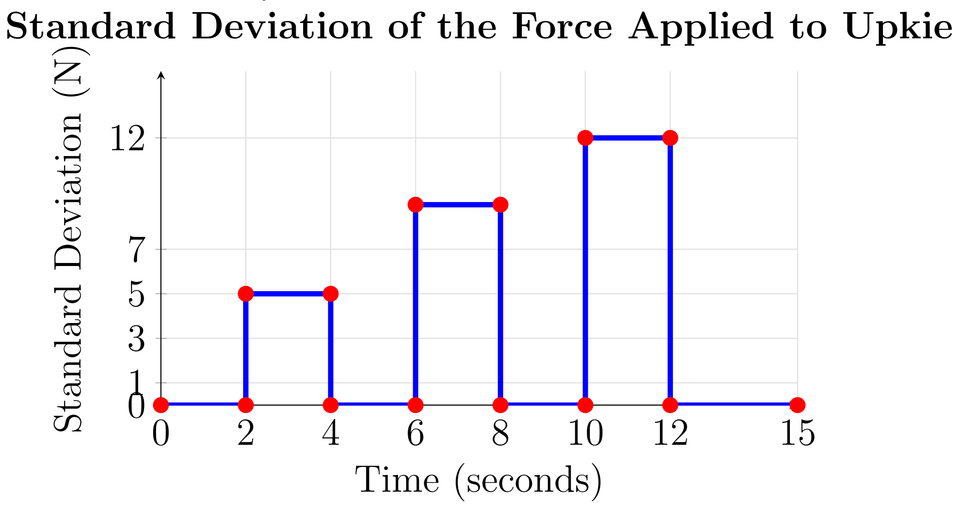


The results were not very good: the robot specializes for this form of force and does not resist for the previous MSFOS. We decided to change the constant forces to Gaussian forces where we gradually increase the standard deviation. We hoped this would result in a more robust policy.

The trial with increasing standard deviation also gave poor results because the problem was that we did not give the robot time to rebalance.

3.3.2 Trial 2: Random Environment

Then, we tried making the applied forces random: each force applied in an interval follows a Gaussian distribution with a slightly positive mean and a standard deviation fixed by interval.



The robot balances better: it achieves 10.3 N of MSFOS (see Fig. 2).

3.4 Change of Environment

We switch to the environment UpkieServos, which changes the observations and the actions that can be applied to the robot. Previously, we acted on the robot's speed to rebalance it, i.e., $\dot{p}$.

In the new environment, we independently control each servomotor. Thus, the action space is much more complex and the exercise is more difficult. Here are the few difficulties we encountered:

- First, we had to normalize each observation in the observation space. Indeed, the observation space is now a bunch of parameters (velocity, position, torque, etc.) for each servomotor and each of these parameters have their own min and max boundaries. We normalize them by their max to have observations and actions in $[-1, 1]$. Conversely, we have to multiply the action the policy gives us by the normalization factor.

- In our first experiments, the robot just kneeled to the ground as in Fig. 3, so we changed the reward by setting it to -10 if the height of the body is under a threshold and otherwise, setting it to the square of the body's height.

- Then, the policy just failed to converge and the loss oscillated wildly (or collapsed at some point). We hypothesized that this was caused by too big updates of the policy with respect to the value function. After (a lot of) experiments, we found that increasing the steps between policy updates and diminishing the PPO clipping parameter helped convergence a lot.

- Finally, we observed a big performance gap between the training and testing performances. We found this was due to the delays in the environment that were not present in the testing environment.

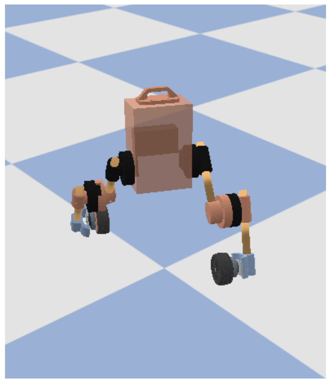


Figure 3: Upkie trained on the Upkie Servos environment

We also changed the `settings.gin` file to decrease the clip range and the learning rate, otherwise the policies did not converge. We trained in the Upkie Servos environment without forces.

We have got interesting results as the MSFOS computed with this policy was equal to 8.1 N (see Fig. 2) after only 420,000 steps of training. Indeed, we did not impose it to be symmetrical with respect to the sagittal plane, then it can have a much better balance with one wheel in the front and one wheel in the back.

It however needs to be noted that the trained policy is very sensitive to delays in the actuation of the motors: removing the delays added in the training to the test bench completely breaks the robot. In the same spirit, running the simulation on my old laptop induces delays in the control loop (compared to a simulation running on the server that was used to compute the MSFOS), which really hurts the performance (the robot isn't able to keep standing).

4 Conclusion

We saw that applying reinforcement learning to the balance of Upkie greatly increased the MSFOS, taking it from 4.5 N to 10.3 N. Concerning the Upkie Servos environment, the added degrees of freedom allowed the policy to achieve an 8.1 N MSFOS without curriculum learning, which is a great improvement compared to the base policy in the Upkie Ground Velocity environment. However, finding the right hyperparameters to reach convergence of the PPO method is a huge challenge which makes the training difficult.

References

[1] Stéphane Caron **andothers**. *Upkie open source wheeled biped robot.* **version** 6.0.0. 2024. URL: `https://github.com/upkie/upkie`.

[2] John Schulman. "Proximal Policy Optimization Algorithms". **in**(): URL: `http://arxiv.org/abs/1707.06347`.