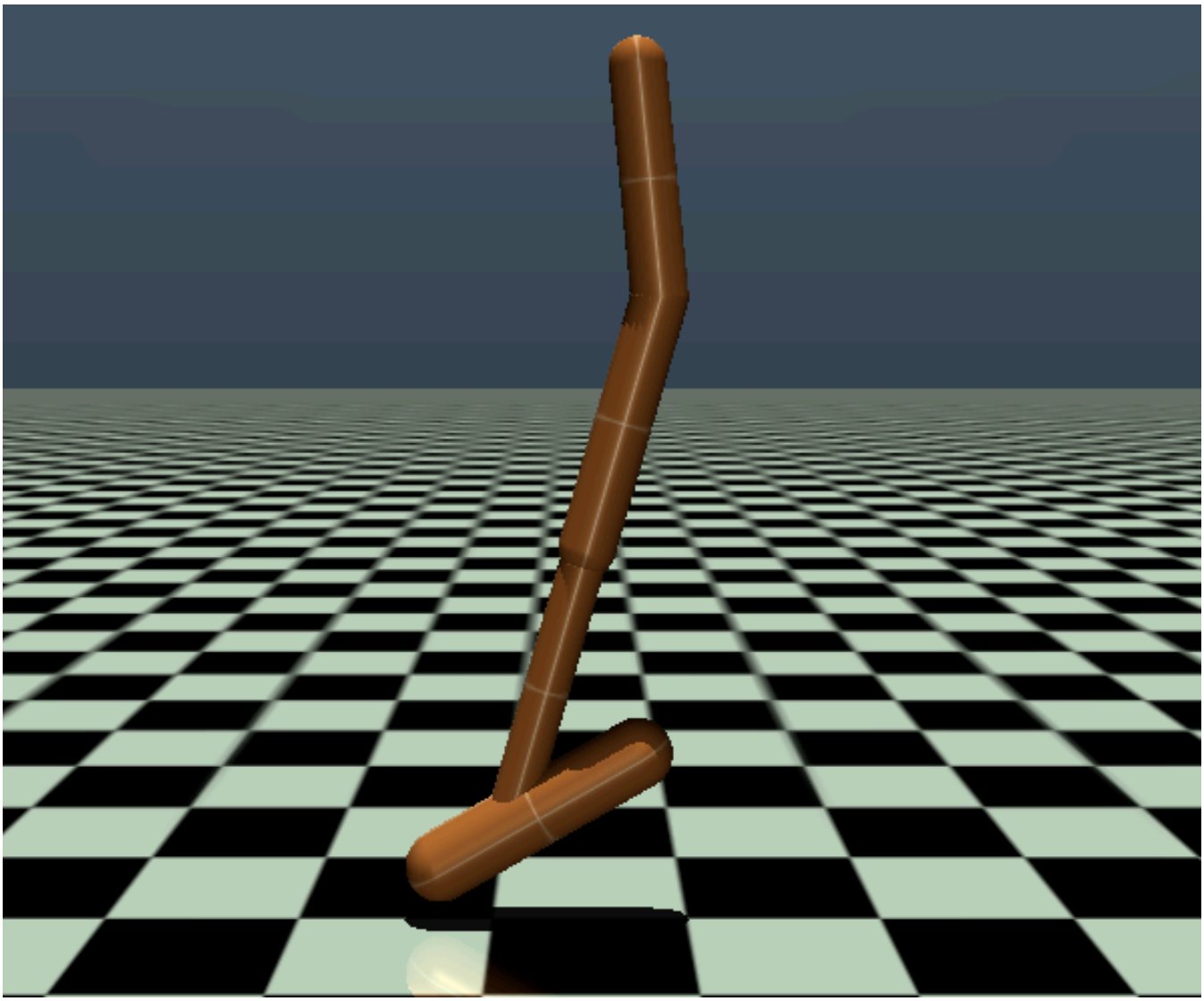
GTS

Gymnasium Training System



Bruno Nicoletti

August 2024

Computer Animation For Film and Television

My career has mostly been in digital visual effects for film and television, having both worked creating visual effects and also writing software used to create those visual effects. I even started a company in the 1990s, The Foundry, to make and sell VFX software.

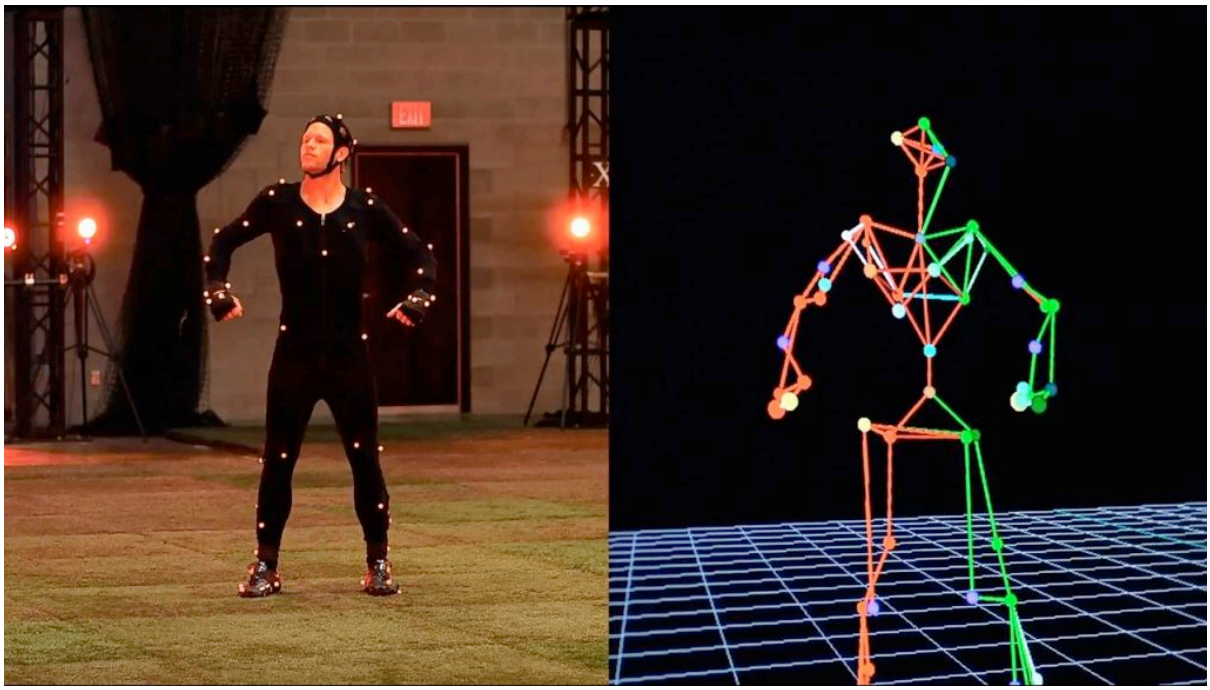
Generating convincing imagery is only part of VFX, making things move realistically is also important to keep the audience engaged. This is especially true for digital humans and creatures.

Digital Characters And Creatures

Autodesk Maya Animation Software

For animation purposes, a digital character or creatue is defined as a skeleton, made up of various bones, which are connected by joints. Each joint is at some angle, and a given set of angles makes up a pose. To animate a digital character, the various joints need to be made to move in a convincing way over time. Normally this is done using animaton software

such as Autodesk's Maya, where an animator specifies poses at keyframes and software deals with making it move in between frames. Another option is motion capture, where the movement of a humun or animal is recorded and applied to a model in a computer.



Motion Capture

Animation and motion capture are both time consuming and require great skill. They are perfect for digital main characters and important scenes where artistic nuance is required. However, they don't scale to large crowds or battle scenes with many thousands of synthetic characters, nor are needed for back ground digital characters/creatures who are not the centre of the action.

Reinforcement Learning For Computer Animation

Reinforcement learning is a way of training a neural network to achieve a goal, via a set of rewards and penalties. It can be a way to generate sufficiently realistic motion for use in computer animation and video games

by having a neural network animate a digital character. The same technique is used to control robots in the real world.

The main way to train a neural network to generate convincing motion is via a physics simulation. Rather than animate poses, the neural network specifies the forces that need to be applied to each part of the figure, so emulating what muscle do on an animal. The physics simulation will the do some maths and figure out how those forces affected the figure and move it appropriately. The neural network monitors that and choose new forces to apply, repeating this process over time. If the network gets it right, the figure animates nicely, if not, it usually falls.

Because it's a physical simulation, this can generate very realistic motion, so much so that robots are trained in such simulations before they are allowed to run around in the real world.

Reinforcement learning is the technique used to train such neural networks, via a system of rewards and penalties. The neural network is encouraged to learn what actions to take so that it that maximise rewards over time.

GTS

For my project I've developed GTS, or the Gymnasium Training System, which I've used to train several neural networks to animate various figures from the Gymnasium simulation package. This is a profoundly rough prototype of a system you could use to to train a neural network for animation or even robotics, though much better systems actually exist already.

It consists of 3 main python programs...

- train.py : which trains a neural network to control a figure,
- play.py : which will show the neural network controlling a figure,

- paramSearch.py : which searches for the best options to train a neural network.

Gymnasium comes with several pre-defined physical models, what it calls 'environments'. I have used three of those in my project.

- inverted pendulum : the equivalent of balancing an upright pencil on the end of your finger,
- hopper : a single leg that is being trained to move forward and not fall over,
- walker : a pair of legs that are being trained to move forward and not fall over.

Results

I've managed to train several neural networks on the pendulum, hopper and walker with various degrees of success using several techniques. Some of the results, while not succeeding, are quite funny. The neural networks are saved in 'checkpoint' files and can by run in realtime via a simulator, to see one run use the play.py program. (But first you need to follow the installation instructions in README.md). For example....

```
play.py hopper/hopperA2C.checkpoint
```

Ideally I should haved record some videos, but I ran out of time!

The Inverted Pendulum

This was the most succesful, I managed to train it with all three algorithms quite easily. They all stabilise rapidly and look like they aren't moving, but are infact jiggling very slightly. Play any of the models in the the folder 'inverted'.

The Hopper

The hopper was harder to train, but I got there. It had a bad habit of not wanting to move, because the system rewards not falling over as well as moving. So the system would often just stand still, maximising rewards for staying up right by avoiding the risk of tripping if it attempted a hop. Lots of tweaking the right set of training parameters finally got there. The hopper in "hopper/hopperA2C.checkpoint" is the result, and it happily hops away.

The Walker

The walker was even harder to train, as it had twice the number of joints as the hopper. I ran out of time to get it to move in a pleasing manner, and it had exactly the same problem as the hopper, with not wanting to move. Ironocally the most successful result for the walker had it hopping off into the distance on one leg. This is in the file "walker/walkerA2C.checkpoint", which can be played.

Comedy Value

Quite a few of the failed training schemes yielded neural networks that made me laugh when the failed. For example the file 'walker/walkerPPOBad.checkpoint' looks very much like a drunken stumble, and the file 'hopper/worming.checkpoint' worms forward once it trips over. If you are going to play these back, you need to pass the "-k" parameter, to keep the simulation playing after if falls over. eg...

```
play.py -k hopper/worming.checkpoint
```

I wish I'd kept the one of the walker doing the splits.

Reflection

Training neural networks to animate a character or create is complex and nuanced, but can achieve successful results. I've only scratched the surface

with my system, but it does show that reinforcement learning is applicable to computer animation (and infact already has been).