Learning dynamic behavior of a quadruped robot

Martin

MInf Project (Part 1) Report

Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

In this work I create reinforcement learning environment that simulates Anymal robotics platform. I then use Proximal policy optimization to show that it is possible to learn dynamic locomotion in the forward direction. However this motion relies on rapidly varied motion and does not exhibit gait pattern. I then try to reduce this problem by using linear interpolation, input filtering, torque cost and smoothness cost but I am unable to train an agent that would exhibit gait pattern.

In the last section I increase the complexity of the environment by adding obstacles that the agent has to step over. I then add ray-casting based visual input to the policy network. Lastly I demonstrate that agent is able to learn to step over some of the obstacles.

Acknowledgements

Special thanks to Dr Zhibin Li, supervisor of this project, Kai Yuan, his Phd student who provided most practical and specific advice on how to get working results and Wouter Wolfslag who provided initial guidance.

I would also like to express my gratitude to people of SLMC Group, with whom I have discussed some of the problems encoutnered.

Table of Contents

Chapter 1

Introduction

Compared to wheeled designs, legged robots are much more maneuverable because they can step over obstacles and cross over holes. Thus they can overcome complex environment such as cave systems[1] or forests[2].

Legged robots usually use either hydraulic or electric actuators, hydraulic systems have advantage of higher energy densities of conventional chemical fuel, while electric actuators are limited by current battery technology (Anymal platform lasts at least 2 hours [12]). On the other hand electric actuator offer more precision control.

In this work I have focused on Anymal platform[12] because the University of Edinburgh has two of these system available and even though this work is entirely based on simulation the future work might involve porting explored methods to the physical platform. Anymal platform is dog-like quadrupled robot with 12 joints and it can be equipped with 3D cameras and LIDAR.

Control of the Anymal platform is a complex task, there are 12 joints that can move continuously to any degree of rotation. This makes it highly complex space where only small subspace leads to stable behaviour. In the last decade there have been many breakthroughs in machine learning algorithms, particlarly AlexNet[16] demonstrated that it is possible to train very deep convolutional networks. Deep reinforcement learning (DeepRL) is machine learning discipline that uses deep neural networks and reinforcement learning to train agents to operate in complex environments using only reward signal.

There are significant challenges of using DeepRL for robotics control. Because the agent initially has no knowledge of the robots mechanics it requires training by trail and error. These errors might result in system damage and downtime. Moreover physical systems have to be supervised at all time with the personal ready to use emergency stop button. A solution is to use simulation for training and physical platform for testing

[1] Robotic Systems Lab: Graph-based Path Planner: ANYmal Quadruped Robot Exploring Gonzen Mine: https://www.youtube.com/watch?v=W9lgdmDg6UM

[2] SciNews: Boston Dynamics ATLAS robot walking in a forest. https://www.youtube.com/watch?v=M7nLQpWiy1o

the learned behavior. Apart from safety simulation also enables faster than real time training[13] and parallelization[9].

The drawback of simulation based learning is that there might be significant reality gap between the simulated environment and real environment. This can have many causes such as delays in processing time, imperfect mechanical modelling of the robot or different behaviour of the servomotors. [13] showed that it is possible to overcome the reality gap and deploy trained policy to physical platform. This project focuses only on training and testing in simulator.

The goal of this project is to simulate Anymal platform to learn walking behaviour in non-flat environment. The first part of this report discusses training an agent in a flat environment to establish baseline for the performance. The second part adds sensory input to the agent and extends the environment.

Chapter 2

Background

2.1 Reinforcement learning

Reinforcement learning is a branch of Machine learning that focuses on interaction of agent and environment; specifically learning mapping of observations of state to actions that agent executes in order to maximize cumulative reward[21]. The goal of this project is to develop an agent that is able to utilize visual information to navigate complex terrain. In order to be able to navigate the terrain, the agent has to be able to process visual information. Currently the best performing models at ImageNet [5] and similar datasets are based on deep learning. Therefore it makes sense to use deep learning techniques in reinforcement learning settings to tackle a problem of locomotion in complex terrain, deep reinforcement learning is a field that combines deep neural networks with reinforcement learning theory.

In this settings, the agent is controlled by policy π_θ, where θ are parameters of the network. In this project both action space and observation space is continuous and therefore a stochastic policy is used. Specifically action at time-step t is defined as $a_t \sim \pi_\theta(\cdot|s_t)$. Time steps t are organized into episodes of length T. Episodes are defined in terms of trajectories $\tau = (s_0, a_0, s_1, a_1, ...)$ describing states of the environment and actions that agent took. The return R is then defined as cumulative discounted reward over the trajectory. Reinforcement learning can thus be expressed as optimization problem[2].

Reinforcement learning algorithms are then used to optimize and improve the performance of the agent. In this project the goal was to use an existing algorithm and create environment and rewards to learn walking behaviour.

2.2 Reinforcement learning frameworks

There are number of publicly available frameworks and libraries that implement reinforcement learning algorithms. The advantages of using such libraries are clear; they are well tested, documented and optimized.

Most of these libraries work with OpenAI gym environment[3], this is standardized interface that provides unified access to environment state and actions. Because of the standardized interface we can change the RL algorithm without modifying the environment.

Firstly there is OpenAI baselines[6] a Python package that contains implementations of most commonly used RL algorithms. However this package is not well documented and the API is inconsistent across different algorithms it implements. Stable-Baselines[10] is fork of the Baselines package done by researches from ENSTA ParisTech. Stable baselines package is better documented and offers multiprocessing for some of the algorithms. Multiprocessing can improve the speed of training and thus is it an important feature.

However I have not been able to train any agents with stable baselines package. For all experiments in this project I have used Spinning Up package from OpenAI[2]. There are number of reasons why I have chosen this library, firstly I have been able to train agents using this library, secondly the output of the training processing is much clearer and lastly because they have recently added support for PyTorch[18] that I prefer over Tensorflow[1]. This package also has a multiprocessing support using the Intel MPI library, however this feature did not work for me because it frequently crashed.

Another RL library is RLlib[17] from Barkley BAIR lab that focuses on distributing the training across multiple computers. This library is well documented and offers potential gains in speed of training, unfortunately I have not been able to test it.

2.3 Simulation frameworks

Simulating the environment requires a 3d rigid body simulation. The most common choices are MuJoCo[22], PyBullet[4] and Gazebo[15]. I have decided to use PyBullet because it is open source and does not require license. Moreover it has been suggested by Wouter Wolfslag[1] as easier to use.

[1] Postdoc working with project supervisor

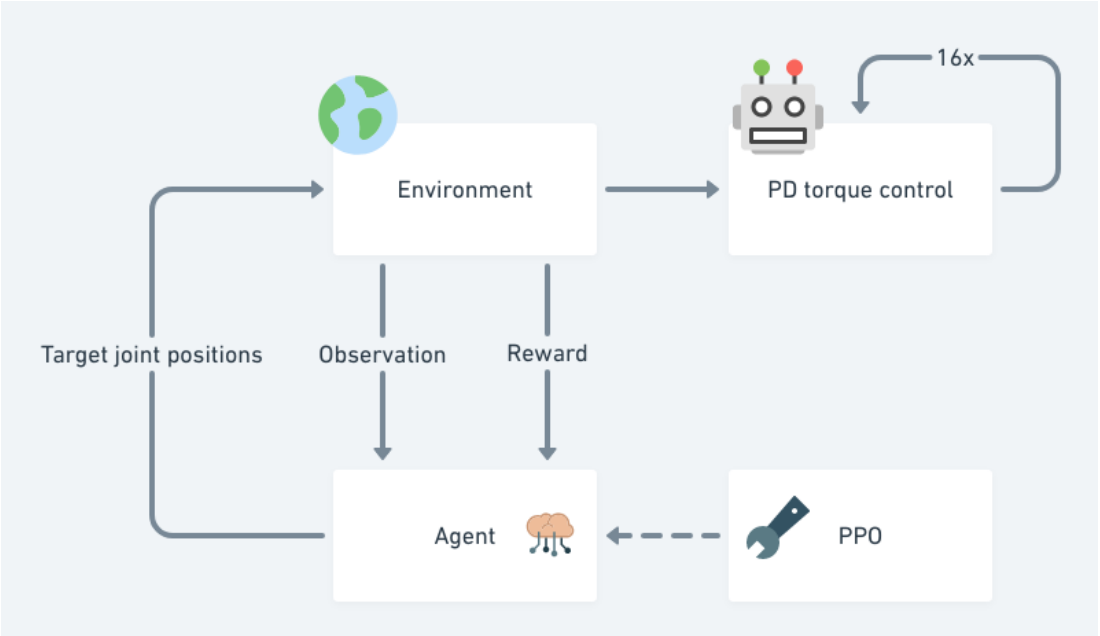

Figure 2.1: Interaction of environment, agent and PD control.

2.4 PD controller

A proportional–derivative controller (variation of proportional–integral–derivative controller) is used for torque control of the robot.

Specifically the torque is calculated by:

$$\tau(x) = K_p(x - p) + K_d(0 - v)$$

where

- x is target position of the joint
- p is current position of the joint
- v is the velocity of the joint
- k_p is proportional gain (stiffness constant)
- k_d is derivative gain (dumping constant)

Note that the target velocities are zero. This is not ideal for for swing motions. I experimented with outputting target positions from agent together with target velocities however I did not manage to get meaningful behaviour.

Chapter 3

Creating gym environment

As a starting point of this project I was provided OpenAI gym environment that integrated PyBullet and Anymal 3D files [1]. However this environment was undocumented and not working therefore I had to rewrite most of the code.

The goal of the project is to develop a reinforcement learning environment that enables agent to learn dynamic locomotion. As a start I have created the environment that rewarded standing.

3.1 Environment

The environment provides following observations about the state:

- joint position; angle of each one of the 12 joints of the robot
- base orientation; angle with reference to the environment
- base velocity; velocity vector in m/s
- base angular velocity

Not all of these are necessary for the standing task, but they will be useful for the walking task.

The environment is controlling time flow in PyBullet simulation, specifically each time the environment steps the simulation it executes $\frac{1}{400}$ seconds of action. PyBullet also offers real-time mode of control, but using step-based control has the advantage that the simulation can happen faster than real time and thus speed up the training process.

3.2 Cost function

There are a number of ways to reward standing, I have chosen one of the simplest reward - a mean squared error of desired joint angles and current position. This approach is less than ideal the agent is forced to learn specific posture instead of trying to find

[1] This code was provided by Wouter Wolfslag

the best stable stand. Moreover this reward is not normalized - it ranges from 0 to 2π. However it was very simple to implement.

$$R = \frac{1}{12} \sum_{n=1}^{12} \sqrt{p_n - t_n}$$

where p is current angle of join n and t is target position of joint n.

3.3 Action

The output of the network - action space, is interepreted as position targets for the joins. PD controller is used to compute the torques for each joint. In this experiment the PD constants were set as $P = 65; D = 0.3$. In this experiment there was only one PD action executed for each step.

3.4 Agent

A fully connected network with two hidden layers of 64 ReLU units was used. This network size was used because it is the default for the OpenAI Baselines package.

3.5 Training procedure

The policy was trained for 10,000 epochs where each epoch had 1,000 time-steps. Thus each episode had 2.5 seconds of experience and overall the agent experienced 6.94 hours of time.

3.6 Results

The agent was able to learn to stand for the length of an episode after two hours of training, this result has been confirmed both by inspecting the average reward per episode as well as inspecting the trained model visually. This result confirmed that the agent was able to learn to counter gravitational forces and control simulated body of Anymal robot.

The standing experiment confirmed that the Reinforcement learning loop has been set up correctly and thus enabled me to move to the next part - learning movement.

Chapter 4

Forward motion on flat surface

In previous chapter I have estabilished that the agent is able to train with-in the environment, the goal of this chapter was to find agent and environment configuration that enables simple movement of Anymal robot on a flat surface.

This chapter is a list of consecutive experiments that enabled the forward motion, most of these experiments failed or did not improve the behaviour of previous experiments.

4.1 Reward of absolute distance traveled

The motivation for this experiment was to reward the agent for absolute distance it traveled from the origin.

$$R(s) = \sqrt{x^2 + y^2}$$

This experiment failed to learn. There are multiple problems with this reward, firstly it does not reward action but position, this means that if the agent takes bad action at later state it is rewarded more than good action at initial state.

The second problem with this reward function is that it is not normalized. Stable-baselines "Tips and Tricks" [10] recommends to use normalized rewards and action space for Gaussian policies such as PPO.

4.2 Normalized velocity reward

The maximum velocity of Anymal is 1.6m/s[13]. Threfore I have tried linear reward function clipped between -1 and 1.

$$y = max(-1, min(1, \frac{1}{1.6}x))$$

this function receives highest reward of 1 at velocity of $1.6m/s$. This addresses some of the issues with previous reward.

However similarly to previous experiments, I was not able to achieve any meaningful behaviour.

4.3 Early stopping

Each episode lasts at most 20s. It is stopped early if any part but the feet touches the ground. I have tried training agent without early stopping and discouraging bad behaviour with negative rewards, however the agent failed to learn motion forward.

Previously I have used a reward function that returned negative rewards for states with low base height. This kind of reward led to agent learning to trigger early stopping to stop accumulating negative return. For this reason I only use positive rewards.

4.4 Radial basis reward

After consulting the reward with Kai Yuan[1], I implemented the reward using radial basis functions. This was also inspired by a report on the same project[19].

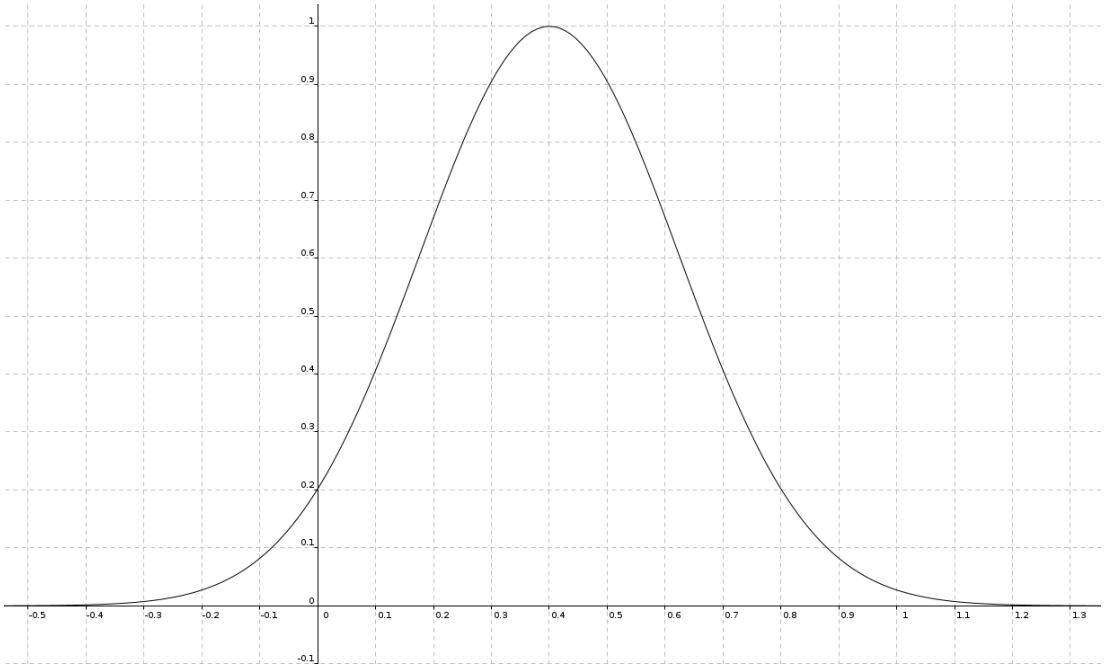

Figure 4.1: Radial basis reward for velocity, $R(v_x) = e^{(-10(v_x-0.4)^2)}$

$$R(v_x) = e^{-10(v_x-v_t)^2}$$

Simply changing the reward was not enough to improve the behaviour. The second problem with my environment was PD loop, specifically the environment was running at 400Hz, 1 PD loop for every agent action. The problem is that because of discount

[1] PhD student of project supervisor

factor of $\gamma = 0.99$. This parameter quantifies how much importance should be given to future rewards. With the PD controller running at 400HZ a reward 1s in future will be discounted by $0.99^{400} \approx 0.02$ - this is not ideal.

Increasing number of PD loops for each step from 1 to 16 preserves the 400HZ frequency of the controller and improves the discounted return 1 second in future - with 25Hz the discount factor is $0.99^{25} \approx 0.78$. Thus the agent is considering states that are in immediate feature.

This change also affected the amount of training time per epoch - previously the agent would experience 2.5 seconds of training time for epoch, but with more 16 loops per each agent step this increased to 40 seconds.

Using this reward function I was able to train the agent to move forward with target velocity of $v_t = 0.4$ m/s.

4.5 Results

The agent was able to achieve the maximum reward over 20s episode in 10 trials. Visually inspecting the agent[2] reveals that it learned to use its back legs stabilize itself while using the front legs to push itself. The movements are very small and there is no gait-like pattern. However this position is stable and the agent does not fall on the ground.

This experiment showed that the agent is able to learn forward motion however inperfect it is. In the next chapter I have focused on improving the performance and developing better ways of evaluating learned behaviour.

[2]Video of trained agent https://www.youtube.com/watch?v=jaPIAxIAUk4, note that the video ends with the fall of the agent, this has been caused by increasing the time of the episode for the evaluation, during training the episode ended before the fall.

Chapter 5

Improving the forward motion

The previous chapter established that the agent is able to learn to move forward within the simulated environment. However this movement is less than ideal, this chapter focuses on improving the motion.

5.1 Improving the experiment procedure

5.1.1 Evaluating performance of the agents

In order to compare the performance and potential improvements I had to develop logging and graphing functionality for trained agents. PyBullet offers logging and plotting utilities however these don't allow for logging of custom quantities such as rewards and combining the graphs together.

Therefore I have created following plots utilities:

- Tracking plot - this is essentially a 2D view of where each end-effector touched the ground. This is useful for observing the trajectory robot took and how each of its leg behave.
- Contact plot that shows when enf-effectors touches the ground, this helps to show whether the agent uses each leg equally or whether it relies on some of the legs to stabilize itself.
- PD controller plot that shows relation between output of the policy and how it was interpreted using PD controller.
- Reward graph - plotting different rewards received at each step.
- Graphs for all observable quantities from the PyBullet - these include angular velocities, reaction forces and applied torques.
- Plots of the unfiltered and filtered observation about joint positions. The filtering is explained in later section in this chapter.

5.1.2 Increasing reproducibility of the experiments

Adding new features quickly increased code complexity; in particular in order to be able to run the older experiments I had to preserve the code that was used to train the models.

To an extent Git versioning system does provide this functionality - one can always revert to the commit that contains the trained model. This approach however does not enable to easily fix errors in code that does not influence the training process such as logging and plotting. In order to see how particular experiment compared to previous result it is sometimes needed to regenerate the plots.

Therefore I have create my own solution where each experiment is defined by a folder containing JSON configuration file and trained models. The configuration file contains following information:

- Initialization arguments for the environment. These are mostly flags and constants to enable new behaviour. To add a new feature to the environment I would then add a flag of the feature that is turned off by default.
- Configuration for the training procedure such as number of epochs and steps per epoch.
- Policy configuration such as network type and size.

This experiment setup is easy to inspect and change. All configuration is versioned in Git repository.

5.2 Improving the agent

5.2.1 RL Algorithm selection

Motivation behind this experiment was to find the best performing reinforcement learning algorithm, previously I have established that training with PPO[20] works, but I have not tried alternatives. Because OpenAI Gym uses standardized interface it did not require significant effort to set-up these experiments.

I had following requirements for reinforcement learning algorithms:

- It needs to work with continuous input and output space.
- It should work with Tensorflow or PyTorch models because in following experiments the input space will include visual data.
- The training should take less than 10 hours on my computer - Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz, with 64 GB of RAM and NVIDIA GeForce RTX 2080 GPU.
- It needs to be implemented by well maintained python package; either Spinning up [2], OpenAI baselines [1] or Stable-Baselines package[10].

[1] https://github.com/openai/baselines

OpenAI Spinning up [2] documentation has bench-marking for several MuJoCo environments[2], the most similar to this project is ”Ant” because it involves movement of agent in 3D environment. Results from the benchmark show that the best performance was achieved using Twin Delayed Deep Deterministic policy gradient algorithm (TD3)[7] and Soft Actor-Critic (SAC)[8] algorithms.

In this benchmark PPO[20] performed significantly worse then TD3 or SAC, however as authors note[2]:

> The Spinning Up implementations of VPG, TRPO, and PPO are overall a bit weaker than the best reported results for these algorithms. This is due to the absence of some standard tricks (such as observation normalization and normalized value regression targets) [...]

The TD3[7] also compared various algorithms (page 7), interestingly compared to Spinning Up benchmark, SAC performed significantly worse in Ant-v1 environment. However in SAC[8], SAC outperformed TD3. It is unclear where the discrepancy between these results comes from.

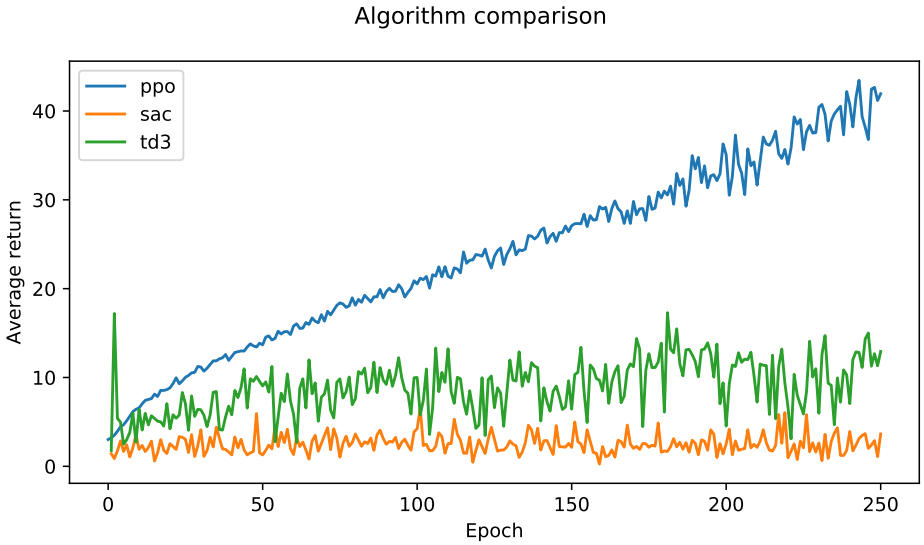

Figure 5.1: Average return per epoch, for PPO this refers to training, whereas for SAC and TD3 it refers to average test episode return. It is is clear that PPO outperforms both of these algorithms.

To select optimal algorithm I have decided to compare TD3[7], SAC[8] and PPO[20]. Average episode returns are plotted at figure 5.1. All experiments were run with default hyper-parameters from OpenAI Spinning Up framework. It is not clear why both SAC and TD3 performed worse than reported results in OpenAI benchmark. Furthermore average PPO training took 11.5s per episode while SAC took 52.6s and TD3 50.4s,

In conclusion PPO clearly outperforms both SAC and TD3 and is much faster to train.For proper comparison it would be necessary to find better hyper-parameters for

[2]https://spinningup.openai.com/en/latest/spinningup/bench.html

SAC and TD3 algorithms, this is however unnecessary for this project because PPO trains good enough.

5.2.2 Network size

The motivation behind this experiment was to find the best performing network size and architecture. This experiment was quite limited in scope, I have only explored network sizes: I did not explore changing activation unit or exploring different network design.

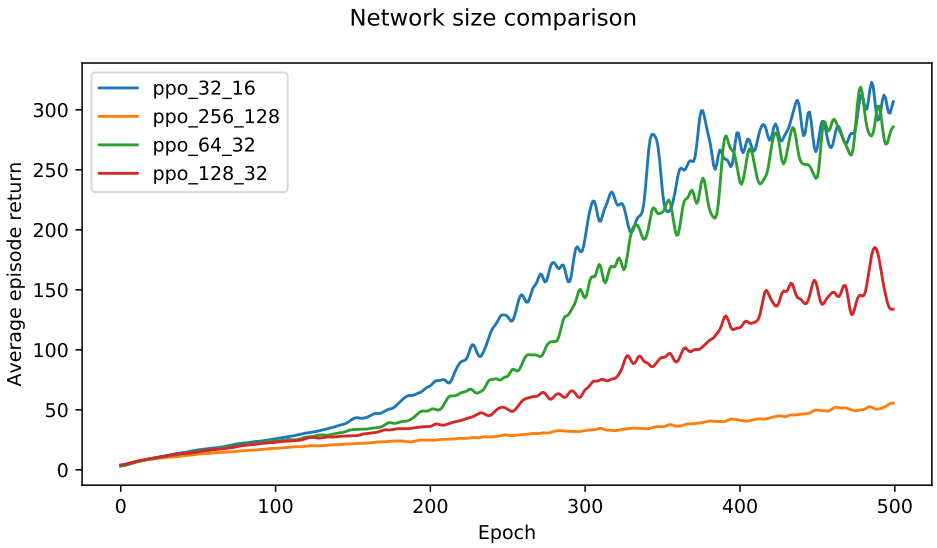

Figure 5.2: Comparison of different number of hidden units. All networks used ReLU activation function. Note this graph has been smoothed.

From figure 5.2 it is clear that smaller networks outperform bigger networks. This was unexpected result to me, I have expected the larger networks to be able to perform at least as good as a smaller networks. It is possible that the larger networks need more episodes to train.

5.3 Increasing smoothness of the movement

5.3.1 Linear interpolation of actions

The learned policies managed to move forward but the movement was not very fluid, the transitions from one state to another were abrupt. One approach to reduce this is interpolate action with previous actions.

This is very simple interpolation method, we calculate each target value for PD controller as follows:

$$\tau_i = \tau(\frac{x' - x}{16} i)$$

where

- x is target position of the joint given by agent
- x' is previous target position
- i is the index of the current PD step ranging from 1 to 16

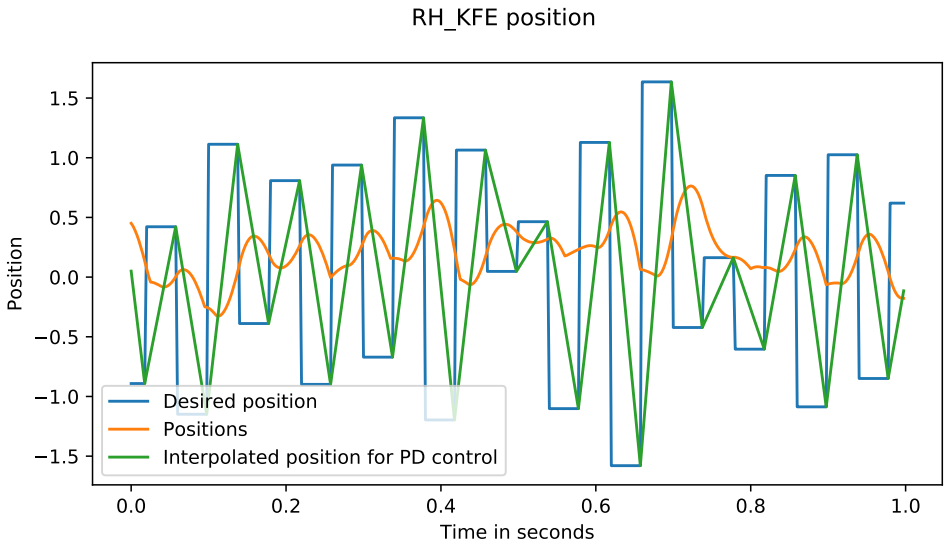

Figure 5.3: This graph shows three different positions of the agent, ”Desired positions” refer to positions as outputted by the policy network, ”Positions” refer to actual ground truth positions at each timestep. Finally interpolated positions refer to interpolated positions that were used as an input for PD controller.

Figure 5.4 shows an example of learned behaviour. The policy learned to work within linear interpolation, as a means to control the interpolated controller the outputs of the policy are far apart from one state to another and relying on interpolation to ”smooth” the action between the extremes. From the Figure it is also clear that PD controller is not working correctly. One explanation is that P (proportional) constant used in this experiment - 125 is too small and therefore agent does not react quick enough. Other possible explanation is that PD controller does not work well with moving targets.

Overall linear interpolation did not measurably improve learned behaviour.

5.3.2 Torque cost

The motivation behind torque cost is to reduce the jittery movement of the robot. By constraining the torque the policy has to work with the angular momentum of the joints and therefore the actions should be smoother and less jittery.

In this experiment I have added radial basis function that penalizes sum of torques used throughout each PD loop. I have inspected previously trained policies and calculated that average sum of PD torque over 100 runs is around 600Nm. Therefore I have decided that the agent reward should be centered around value of 300.

Retraining the agent with torque cost did not improve the before-mentioned issues. The agent did not learn to maximize the reward, it is able to occasionally obtain it but

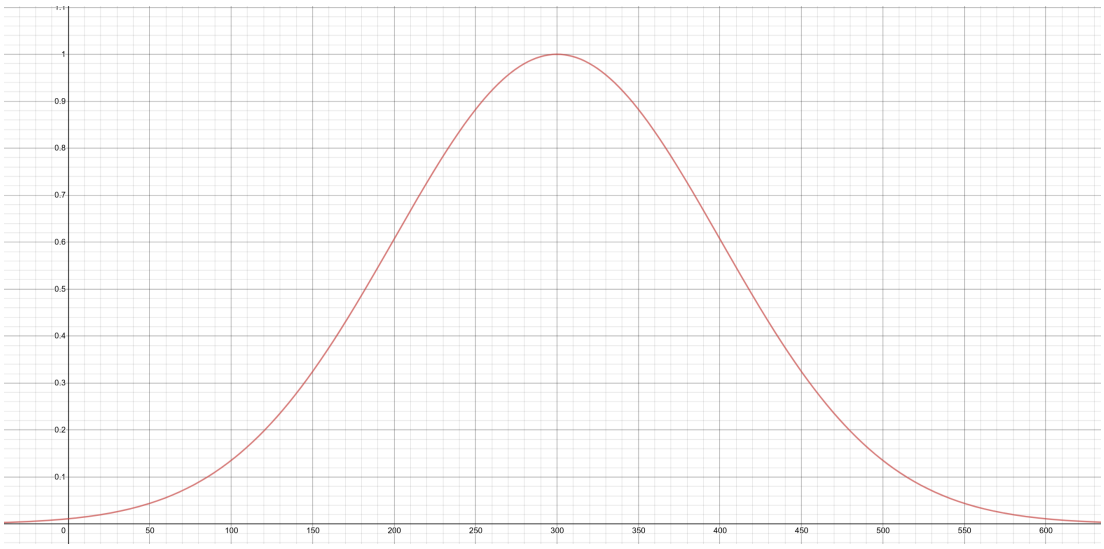

Figure 5.4: Torque reward of $e^{-0.00005(x-300)^2}$, where x is the sum of torque executed over the 16 actions of PD loop.

it does not get score this reward regularly unlike other rewards. It is possible that more thorough search for target value is necessary.

5.3.3 Smoothness cost

The motivation behind this experiment was similar to torque cost - decrease sudden changes in actions from one action, state pair to another. Smoothness cost[14] has also been used in similar setting [13]. It is defined as norm of a difference between previous action and current action $||x_{t-1} - x_t||$.

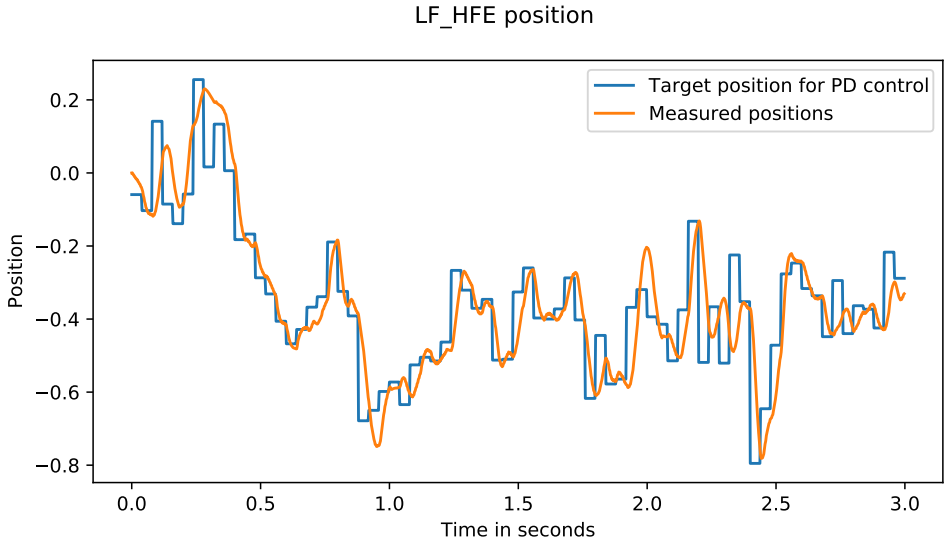

Figure 5.5: Example of PD control of policy trained with smoothness cost. The action seems less abrupt fron one state to another.

As demonstrated by figure 5.5 the actions give by the policy seem to less abrupt from

one state to another. Overall there is visually a slight increase in smoothness of the movement.

5.3.4 Butterworth filter

In previous two experiments I have focused on reward shaping to improve the movement of the agent. Motivation behind this experiment is similar. The hypothesis of this experiment that abruptly changing observation cause the policy to have abruptly changing output. Therefore smoothing the input should result in smoother output. Butterworth filter [3] of order 1 with cut-off frequency of 10 was used to test this hypothesis.

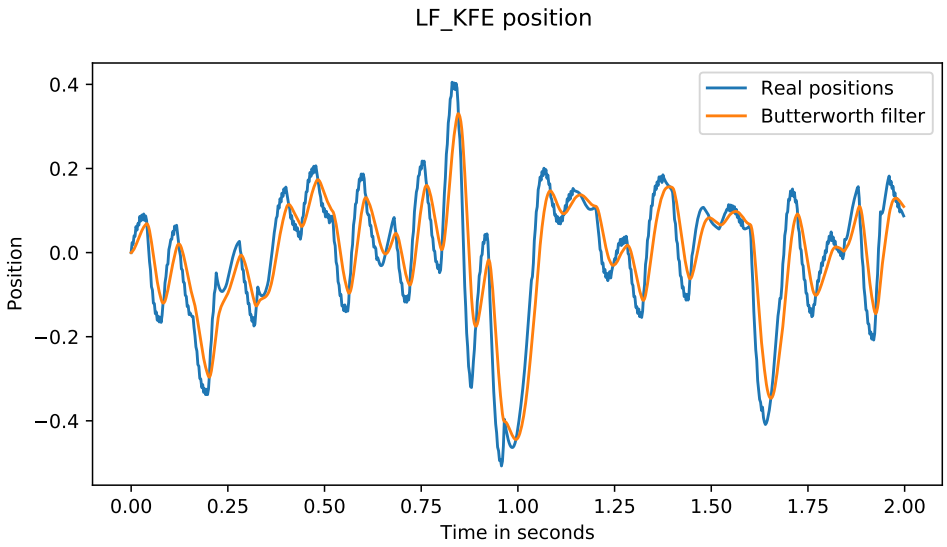

Figure 5.6: Example of filtered observation using provided Butterworth filter. The input is slighlty delayed. We can see that the filtered output is much smoother and is following the same trends as the input. However some of the peaks are lower than in real data. It is unclear how important this is.

From figure 5.6 we can see that the input filtering works well and the observation for the policy are indeed smoother. However as the figure 5.7 illustrates the resulting movement is not smoother. Visual inspection of trained behaviour also confirmed that the filtering have not improved the behaviour.

5.4 Side reward

Using forward reward on itself proved enough to train the agent to move forward, however the direction of the robot was not straight, it turned about 10' to the left. To encourage moving in straight line I added side reward of $e^{(-v_y^2)}$. This reward encourages the agent to minimize y velocity.

Altogether my reward function is defined as:

[3] provided by Kai Yuan

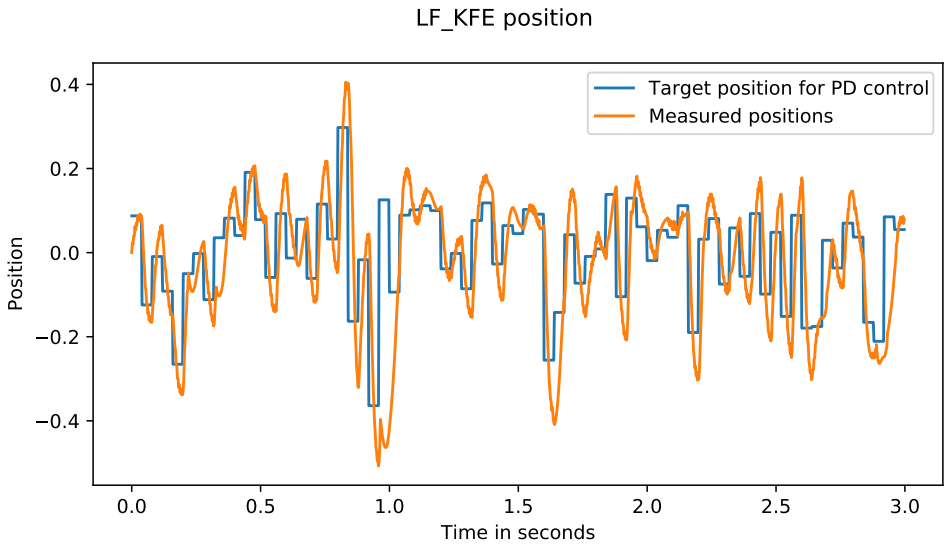

Figure 5.7: Example of policy actions used for PD control, note that this is same episode and time as figure 5.6. The positions don't seem to follow any trajectory and seem to be changing abruptly.

$$R(v) = 0.8e^{-10(v_x-0.4)^2} + 0.2e^{(-v_y^2)}$$

This reward managed to reduce the agent turning behaviour, in 10 trails the agent ended only about 2 degrees of the desired heading.

5.5 Uneven terrain

Throughout the experiments I have worked with perfectly flat ground. In this environment it might be risky for agent to lift the feed above the ground because it is more unstable position that has higher chance of early termination.

Hees 2017 et al[9] write:

> we believe that training agents in richer environments and on a broader spectrum of tasks than is commonly done today is likely to improve the quality and robustness of the learned behaviors – and also the ease with which they can be learned. In that sense, choosing a seemingly more complex environment may actually make learning easier.

In this experiment I have replaced the flat ground with triangular mesh with varying height. I have also randomized starting position so that agent does not memorize the terrain. The agent had no observation about the terrain.

From the video[4] it is clear that agent learned to walk on the uneven terrain, it also managed to learn to overcome fall-like situations. Previously the policy learned to use its right rear leg to safe-guard itself against fall, this has improved on the uneven

[4]https://www.youtube.com/watch?v=uNoX9mdFy88

terrain - the policy uses the legs more proportionally. However agent still uses very small steps to push itself forward, it did not seem to develop gait-like pattern.

5.6 Incremental action

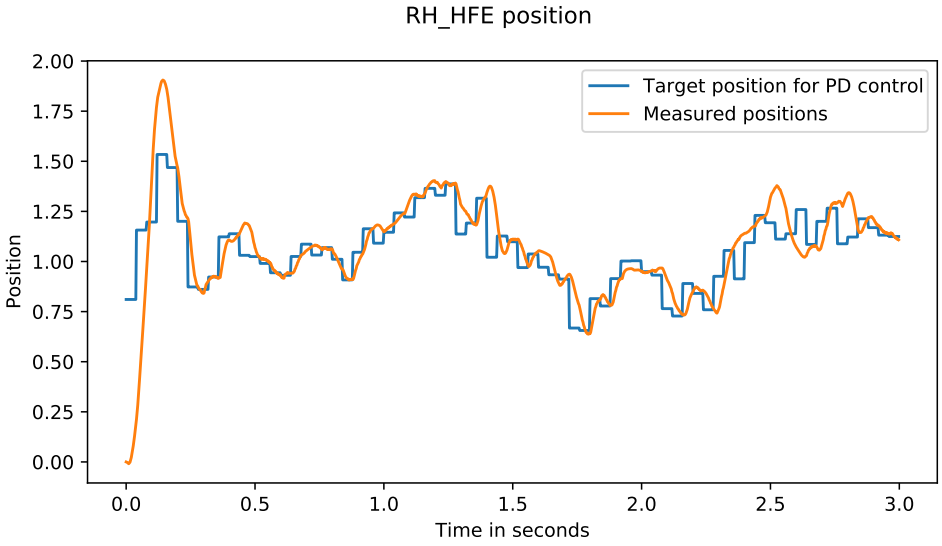

Figure 5.8: Example of PD targets given by incremental action. The targets are much smoother and there is a less variation fron one state to another.

The environment interprets the action provided by agent as a target joint positions that are then used as input for the PD control. But as was seen in previous experiments the action from one state to another can vary greatly. In this experiment the action given by the policy is interpreted as an update of the target position rather than position. The hypothesis is that it is simpler to learn to output small updates rather than the positions itself.

The target positions for the PD control is thus calculated as $p_t = p_{t-1} + a$, where a is the action given by policy.

Overall this approach worked better than others, the step size of each step increased. However as noted illustrated by figure 5.9 the policy still relies on using rear legs for stabilization and does not exhibit gait-like pattern.

5.7 Results

The goal of this chapter was to improve the learned behaviour from previous chapter and increase stability and step size of the agent. While none of the experiments above resolved these issues there were small improvements.

Firstly because I have created plotting utilities for trained policies I was able to resolve issues with PD controller. I have also increased reproducibility of the experiments by separating configuration of the experiments from the code. This change made it easier to compare experiments under the same settings.

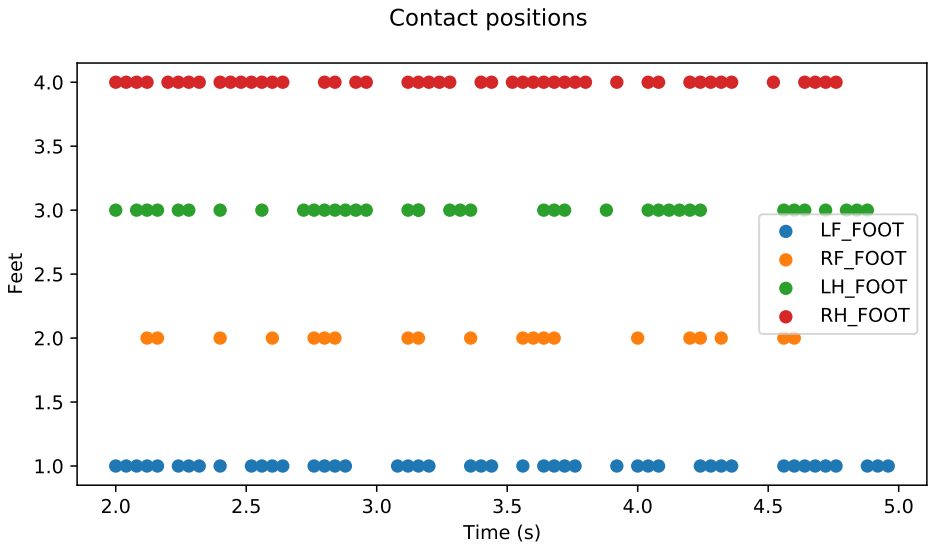

Figure 5.9: Graph of feet contacts during an example episode of incremental action experiment. The learned policy relies on the rear legs for stabilization as can be seen by higher intensity of floor contacts by rear feet. Occasionally the policy makes larger step by both front legs.

I have empirically verified that PPO[20] outperforms both SAC[8] and TD3[7]. This was unexpected as other benchmarks showed that PPO performs worse in different experiments. Furthermore I have empirically found that smaller networks tend to perform better at this task and generally train faster.

The majority of my effort went into increasing smoothness and stability of the movement. To that end I have tried linear interpolation of action, torque cost and Butteworth none of which proved effective. Adding smoothness cost directly to the reward calculation proved to be the most effective and simple way to reduce the abrupt action from one state to another.

I have also showed that adding side reward successfully improved the direction of the movement while being very easy to implement.

Experimenting with more complex environment - using triangular mesh with varying height improved the movement of the agent. The relationship between environment complexity and learned behaviour should be investigated further.

Lastly changing the policy to output updates of the desired position instead of the values themselves showed promise, in particular some of the learned movement exhibited large swing-like behaviour that suggest that agent based on this mechanism should be able to learn larger steps.

Chapter 6

Learning walking behaviour with vision

In previous chapters I have demonstrated that the policy is able to learn motion forward. While I was unable to achieve gait-like behaviour in none of the experiments it was still worth-while to explore if more complex environment would improve the learned behaviour.

6.1 Adding obstacles to the environment

In order to increase complexity of environment five rectangular steps with increasing height. The height of the lowest step is 5cm and the highest one is 10cm. Progressively increasing obstacle size has also been done by Hees 2017 et al[9], one motivation is to allow agent learn the behaviour on simple problems before encountering more challenging problems. Other motivation is to filter bad policies - in order to score the reward agent has to move forward and therefore the obstacles can be viewed as filtering mechanism.

The obstacles are placed far enough from each other so that the body of the robot would in between them.

6.2 Visual input

PyBullet simulator has a built-in camera that allows to obtain image color image from any location, it also allows to control field of view. It also provides depth map for each image. However using this method proved to be too slow during the training. Training of single epoch without visual input takes approximately 10 seconds, after adding the image data using the built-in camera this increased to over 400 seconds. This would make some of the previous experiments prohibitively long to train.

Another option is to use PyBullet ray-casting function. This function takes an input of rays defined by their origin and destination and returns their hit-ratio. This enables

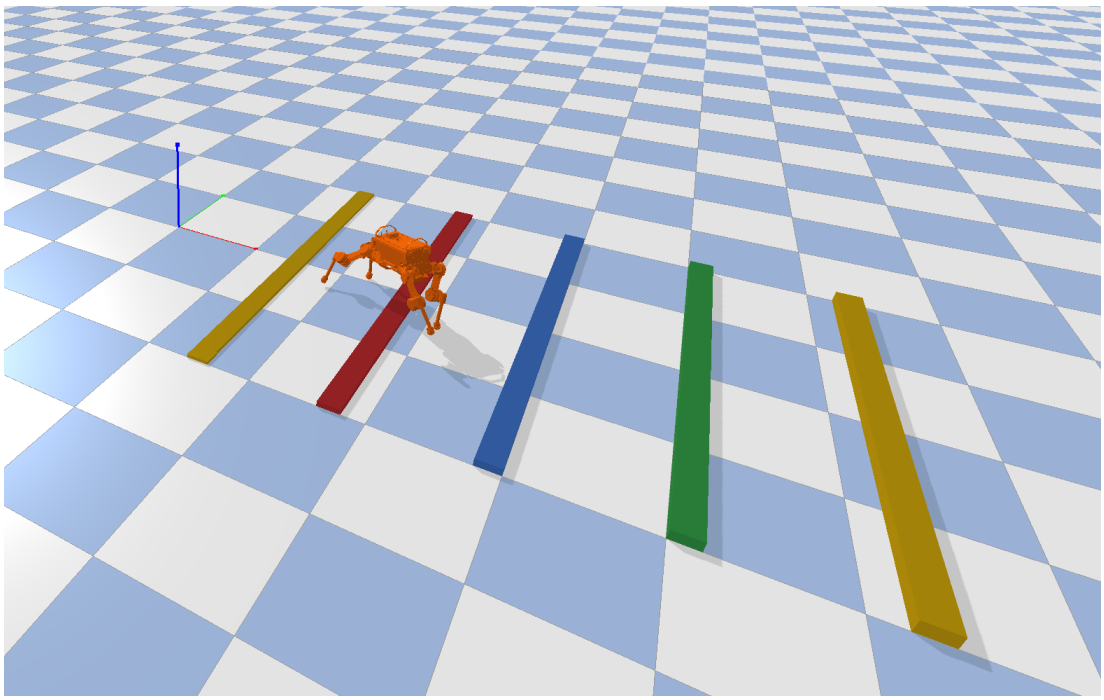

Figure 6.1: Illustration of obstacles in the environment. Notice that they are incresingly higher. This is done to stimulate the learning process.

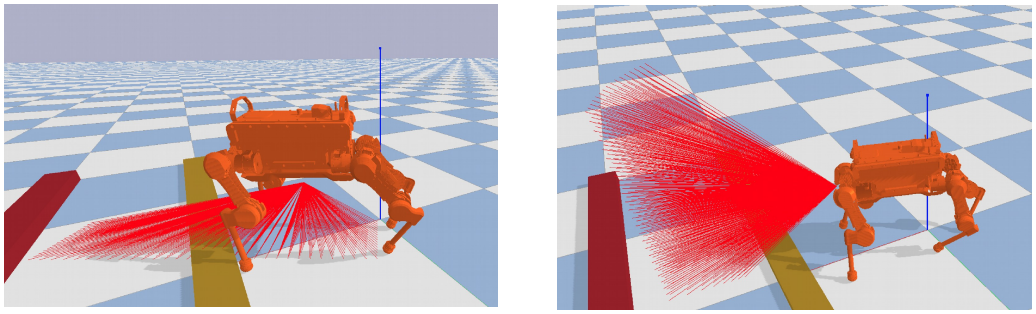

Figure 6.2: Visualization of two configuration of ray-casting camera. Note that this is just an example, I did not use as many rays in any of the experiments. Initially I have developed the configuration on the right, this was motivated by field of vision of four legged animals. However inspecting the ray-casting visualisation it is clear that this configuration is not particularly useful for objects in very near distance. It is thus far more useful to monitor the area bellow the main body as it enables agent to see the obstacles it is stepping over. This is important because the policy only has information about current state - it has no memory and thus would not be able to remember that it is stepping over obstacle.

to reconstruct depth map of the environment. Using ray-casting was much faster than using built-in camera, the average epoch took only about 13s to train this method.

6.2.1 Representation of the ray-casting data

I have explored two different approaches to representing the data from ray casting. The first option is to use hit-ratio of the rays. However this depends on the way rays are formed. If the rays destination is calculated as subset of spherical surface than for example the information about ground will not have linear representation. It is not clear it is then possible to use convolutional neural networks on this type of data. It is also possible to construct ray destination as a rectangular grid under the agent body, the z-coordinates for the hit are then all equal.

Another option is to use the z-coordinates of the hits. This way we can obtain height map of the environment. This representation is more similar to 3D-cloud representation obtained by LIDAR sensor.

I have implemented all of these approaches the only approach that experimentally yielded results was the rectangular grid under the body of the agent.

6.3 Experimenting with convolutional neural networks

In the last decade we have seen that deep convolutional networks tend to work much better at image recognition than other models[16]. Using convolutional neural networks to process the visual input of the agent is thus a promising approach.

Using CNN for this task was not straight forward because the observation space of the environment contains two types of data, firstly visual information that needs to be processed by CNN and then state observation such as joint angles that need to be processed separately.

I have created a network that uses convolutional layers to processes the visual input, the output of the convolutional layers is then concatenate with state information and fed into fully connected neural network.

However this approach did not work better than simple fully connected ReLU network. This is possibly because the environment is very simple and does not contain complex shapes. Another explanation is that the method to obtain visual information - ray casting was not dense enough to enable the network to develop edge detection of the obstacles.

6.4 Exploitation of environment weaknesses

During the training it became clear that the environment reward and setup allowed agent to learn behaviour that scored high reward while not exhibiting intended behaviour. In one such instance the agent learned to perform swing like motion that achieved high velocity reward but did not actually move forward[1]. This behaviour was prevented by withholding any reward unless agent moves forward - specifically that base position increases compared to previous states in current episode.

[1] Video of learned swing motion: https://www.youtube.com/watch?v=3IDIsS-QcnQ

In another instance the policy learned to walk around the steps [2]. This behaviour was prevented by early termination of the episode whenever agent base is detected to be more than 1.5 metres away from $y = 0$ trajectory.

In both of these examples the cheating behaviour was conditioned on stepping over the first step. It is possible that the agent does not have enough information about the back of the body and thus it is more challenging to control rear legs.

6.5 Experimenting with Incremental action

6.5.1 Motivation

In previous experiments I have used positional control where actions given by policies were interpreted as position targets for the PD controller. In Section 5.6 I have explored using different type of control that sometimes exhibited larger steps. Previous experiments showed that the policies that have not used incremental actions were using small steps and thus were unable to step over the obstacles.

6.5.2 Methods

Fully connected network of three hidden ReLU layers of sizes 256, 128 and 64 is used. This network is larger than in previous experiments because the agent has larger input space and has to navigate more complex environment. Rest of the environment and setup is identical to before-mentioned experiments.

6.5.3 Results

In some cases the agent is able to overcome up to 10cm high obstacles [3]. In none of the episodes the agent managed to fully finished the episode, all episodes ended with early termination.

The agent seems to be relying on visual input for aligning itself, often it corrects its heading before the obstacle. It is unclear why this is happening as it it has angle information in the state dimension. The motion of the agent seem very unstable and unreliable.

[2] Video of agent walking around the steps:https://www.youtube.com/watch?v=lp2_LQOe14Y
[3] Video of agent stepping over the steps: https://www.youtube.com/watch?v=YQlgzkUqV6Y

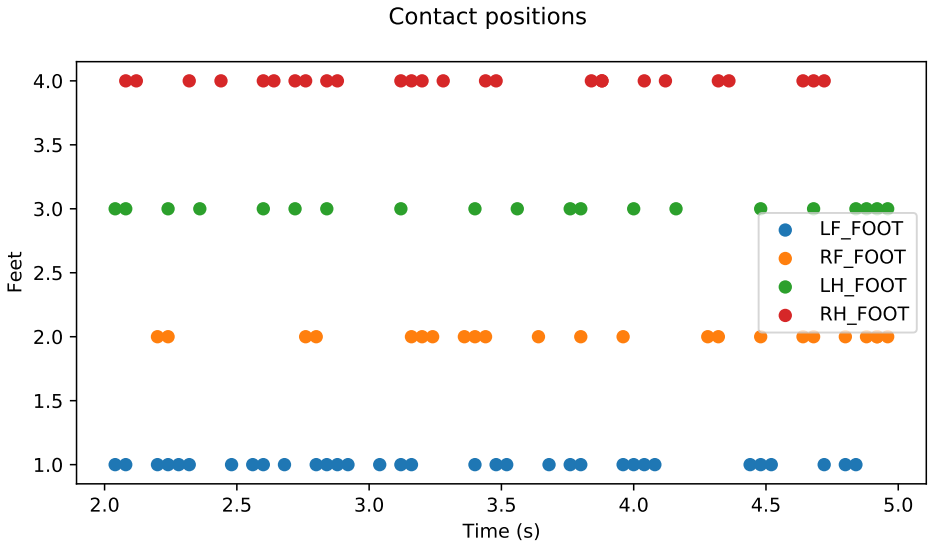

Figure 6.3: Feet position illustrating learned behaviour. Compared to previous policies the agent uses its legs more equally and overall the step size increased. However it still uses some of the legs for support more than others, specifically left front leg and rear right legs are used more than the other two legs.

Chapter 7

Conclusions

The goal of this part of the project was to develop reinforcement learning system that simulates Anymal robotics platform and simultaneously find an agent configuration that demonstrates forward locomotion.

Chapter 3 demonstrates simplest possible version of the simulated environment and demonstrates that an agent is able to learn with-in this environment using Proximal policy optimization. In the following chapter I have modified the environment to demonstrate that forward locomotion is possible. Thus the goal of locomotion has been achieved, however the learned behaviour is far from ideal, the motion relies on small steps that vary significantly from one state to another.

Chapter 4 focuses on improving results from chapter 3. I have explored different network sizes, unexpectedly small networks performed better and learned faster than larger networks. Bench-marking different reinforcement learning algorithms demonstrated that PPO[20] in this particular task learns faster than both TD3[7] and SAC[8], this is contrary to previously reported bench-marks, however I have not performed hyper-parameter tuning for these algorithms and therefore the results are inconclusive.

With the goal of achieving gait pattern I have explored using torque cost, linear interpolation, smoothness cost and Butterworth filter. Linear interpolation increased the variance of actions from one state to another, with policy relying on interpolation to average these differences. Add a torque cost factor to reward function did not increase fluidity of movement, overall policies don't score high on this reward and thus it is possible that this cost require more extensive fine-tuning. A different approach to reduce abruptly changing actions was to use Butterworth filter to smooth input signal. However this approach did not yield any improvement. Adding explicit smoothness cost to reward was the only approach that worked however the improvement was not significant.

I have also investigated whether more complex environment (with triangular mesh instead of flat ground) improves the behaviour of learned policy. I did find improvements of the learned forward motion, specifically the agent learned to increase the height of the steps in order not to fall.

In the last chapter I have explored extending the input space with sensory input. Specifically I added ray-casting based visual information and extended the environment by adding obstacles into agents direct path. I have then demonstrated that policy is able to learn to walk over some of these obstacles. However the motion of the agent is highly unstable and gimmicky. Nevertheless this experimented verified that this approach is viable.

In conclusion I did not achieve gait in none of the experiments. However the agents demonstrated complex behaviour and thus I believe I have partially completed basic goals of this project.

Chapter 8

Future work

It is clear that learned behaviour demonstrated in this report is less than satisfactory. The first task for next year is to improve the motion of the agent, specifically the goal is to achieve a gait pattern for the forward locomotion. One way to achieve this is to find working torque cost.

The most interesting question that arose from this project was the role of the complexity of environment on the learned policy. Gibson environment[23] created complex PyBullet environments from 3D scans of real world indoor spaces. One of the goals is then to try to achieve locomotion with Anymal system in one of the Gibson environments.

Throughout this project I have also struggled with training process, specifically the time required to train the model. Reducing this time with parallelization would enable more complex experiments. RLLib[17] is an reinforcement learning library designed specifically with these goals and thus I intend to try to use instead of Spinning up[2].

Lastly I would like to explore more complex neural networks. In this project I have only used very simple networks that utilized information about current state. Hwangbo et al[13] used observation from previous states, this approach is worth exploring. Another possibility is to use recurrent neural networks or more specifically Long Short-Term Memory Networks[11] as they are designed to for processing time-series data.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[7] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

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

[10] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov,

Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger. Anymal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44, Oct 2016.

[13] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4:eaau5872, 01 2019.

[14] Ming Jin and Javad Lavaei. Stability-certified reinforcement learning: A control-theoretic perspective, 2018.

[15] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan, Sep 2004.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[17] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Branislav Pilňan. Exploring dynamic locomotion ofa quadruped robot: a study ofreinforcement learning for theanymal robot.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[22] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct 2012.

[23] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.