Continuous Actor-Critic Reinforcement Learning for quadrupedal robot control

Tim Williamson tw964@bath.ac.uk
Ben Willshaw bw774@bath.ac.uk
Qun Yang qy458@bath.ac.uk
Linghao Zhou lz2038@bath.ac.uk

1 Problem Definition

For this task, we chose the problem of training a quadrupedal robot to stand and walk forward. We attempted to train two different OpenAI gym environments: Ant robot from the mujoco gym (Achiam, 2018) and a model of ANYbotic's ANYmal robot in gym jiminey, a third-party gym environment (Duburcq. 2022). The Ant robot is a simple 8 degrees of freedom robot, while ANYmal is more complex with 12 degrees of freedom.

For Ant, the state space consists of 27 variables, comprising the location, orientation, the linear and angular velocity of the torso, and the position and velocity of the joints (see appendices for details). The action space consists of 8 variables representing torque to be applied at each of the joints with a value in (-1, 1).

For ANYmal, the state space consists of 37 variables, comprising the location, orientation, linear and angular velocities of the torso, and the position and velocity of the joints (see appendices for details). The action space consists of 12 variables representing torque to be applied at each of the joints with a value in (-80, 80).

For both robots, the state is updated in discrete time steps, and the action provides the torque at each of the joints that are applied for the duration of the time step. The environment includes a rigid body physics engine that simulates the location and velocity of the various segments of the robot using these torques and the external forces of gravity and contact with the ground (modelled as a flat horizontal plane passing through the origin). Therefore the state is updated indirectly from the actions and these external factors.

For Ant, the reward function comprises four parts. There is a reward for keeping the Ant "healthy", which is defined as keeping the torso above the ground (Z dimension within the range [0.2, 1.0]; a reward for forwarding movement in the X dimension; a cost for high values in action; and a cost for high contact forces on the Ant.

For ANYmal, no effective reward function is provided with the environment, so we experimented with reward functions as part of the problem.

2 Background

For both the Ant and ANYmal robots, the action space and the state space are composed of continuous variables. Using the Ant robot as an example, the actions are continuous, taking values in the range [-1, 1], resulting in state values in the range (-inf, inf). It is, therefore, practically impossible to adopt tabular-based reinforcement learning (RL) methods to train a competent control agent, as it would require exploring and storing infinite permutations of state-action pairs in a table. A tabular solution could be designed by discretizing the states and actions, but this would lack the specificity required for the complex robot actions to achieve any level of control.

The Deep Q-learning (DQN) method is a prominent candidate for solving problems where the state space consists of continuous variables. DQN employs a deep neural net to approximate the

action-value function, taking in the state under observation as input. However, DQN is proposed as a value-based method, meaning that the policy update for DQN still relies mostly on utilising fixed pairs of state-to-action values. It is difficult for DQN to deal with problems where the action space is also continuous. Moreover, the DQN method cannot cope with stochastic policy learning because the actions are derived from fixed action values. Therefore, DQN is not adopted for this project.

To solve problems whose state spaces and action spaces are both continuous, policy-based methods are considered. Policy-based methods intend to learn the policy directly, approximated by the possibilities of actions taken in response to input states. Besides the apparent advantages that continuous actions and states are no obstacles, policy-based methods also exploit the inherent stochasticity so that the optimal actions are not entirely determined based on fixed values. This feature facilitates exploration and allows determinism to be achieved over time.

Among the policy-based methods, deep deterministic policy gradient (DDPG) is an actor-critic method designed specifically for environments with continuous action spaces. It uses a critic network to evaluate the value of state-action inputs (Q(s,a)), and uses these values to learn a policy function via an actor-network, which chooses the action which maximises the action-value. Finding this maximum is a trivial lookup when using tabular methods, but using function approximation requires sophisticated optimization of the critic network to accurately approximate Q-values for all state-action inputs (Achiam, 2018). At each time-step, gradient descent is performed on the critic network to minimise the mean-squared Bellman error (MSBE) (Ibid). This is calculated as the squared error between the action-value outputted by the network (current estimate) and a new Bellman target value (the immediate reward plus the discounted action-value for the best action in the next state). By performing gradient descent on the network parameters for this error, we are able to change the network's parameters in such a way as to move the output of the critic network for the respective state-action pair towards this new target value. Gradient ascent is used for the expected value of the sampled state in order for the actor-network to learn a deterministic policy which outputs the action which maximises the Q-value. With the policy being deterministic, actions are selected with some noise added in order to aid exploration.

DDPG makes use of techniques employed in DQN, including experience replay and target networks. Instead of updating the networks' parameters based on the current transition (S, A, R, S'), this transition is stored in a replay buffer, and at each time step, a sample of randomly chosen previous experiences is used to update the networks' parameters. This is important when implementing deep reinforcement learning, as when updating a network's parameters for a single input, it affects the outputs obtained from all possible inputs, not just the current input. Therefore, updating the parameters based on the current transition would have the effect of prioritising transitions experienced later in training to a greater extent than earlier experiences. Instead, updates are performed on a mini-batch of randomly sampled past experiences.

Target networks are used for both the actor and critic networks to make training more stable. The target networks are used to compute the MSBE, which is used to update the moving networks. After updates are performed, a polyak update to the target networks is performed, moving the parameters of the target networks slightly towards the parameters of the regular network every time step. Without target networks, the networks would be unlikely to converge. This slow-moving target network allows stable MSBE minimization. The disadvantage of this method is that generating and updating 4 networks is computationally expensive, causing a long training time.

Twin delayed DDPG (TD3) proposed by Fujimoto, S [1] is considered as an improved version of DDPG. TD3 maintains a similar structure to DDPG, with a few modifications to address the issues where DDPG tends to exploit overestimated Q values, meaning that the policy learning breaks drastically. Based on the implementation guidance published on OpenAI Spinning Up [2], TD3 employs a method named "target policy smoothing", which adds noise to the target actions generated from Actor-network, making it more challenging to exploit the Q-function errors. Another method TD3 adopted is a twin-critic structure. Each policy update will be made using the smaller Q value estimated by the twin critics so that the Q errors are suppressed. Lastly, the policy and target critic update rates are reduced compared to that of DDPG. In this project, TD3 agents are implemented and trained for the Ant and ANYmal robots.

During the background literature research, existing results dealing with quadrupedal robotic control problems were discovered. In the research published by Hwangbo et al. [4], the control problem for ANYmal robot was simulated and solved using the Trusted Region Policy Optimization (TRPO)

method, proposed by Schulman (2015). TRPO is reported to guarantee a monotonic improvement for policy learning. Authors have reported 4 hours of training for locomotion and 11 hours of training for fall recovery on the customarily developed ANYmal model. Similarly, in Duburq et al. (2022) reported research on learning push-recovery policy on humanoid Atalante robot using proximal policy optimization (PPO) [6]. PPO is also an actor-critic style method, improved on TRPO. It was reported in [3] to be one of the most reliable policy-based RL methods, which limits the gradient and how much the policy is allowed to change to tackle instabilities during learning. However, due to the limited time and the knowledge gap, TRPO and PPO will be considered for possible future work.

3 Method

For the Ant robot, only TD3 is implemented. Both DDPG and TD3 agents have experimented on ANYmal robot. In the rest of this report, the TD3 method will be focused on, as TD3 agents have shown significantly better performance. As discussed in the Background section, TD3 method resembles DDPG but with a few improvements, namely noise clipping, twin-critic structure, and delayed policy/target critic update. The discussions for the Ant robot and ANYmal robot are presented in the following subsections. The neural networks implemented for TD3 agents are based on PyTorch.

3.1 TD3 method for Ant robot

To implement TD3 methods, an Actor and a Target Actor are created to learn the policy. Two Critic networks and their associated Target Critic networks are also created. During initialization, the weights and biases of the Actor and two Critics are copied to their Targeted counterparts.

To start the learning process, the Actor takes in the states S from the Ant robot, then outputs smoothed (i.e. adding clipped noises) estimated actions a, responding to the states. The Ant robot takes the actions and generates a corresponding state S' and reward for the actions. Learning then starts.

Target Actor first generates target action a' in response to S', the twin Target Critics then observe the a' and S' pair to evaluate the Q values. The smaller Q value is used to generate target Q values and update both the Critics to avoid policy exploits larger Q function errors. The Actor is updated using the gradient ascent technique which maximises the averaged Q values from the Critic produced from current state S and action a. Because of the delayed update rule, the Actor, Target Actor and Target Twin Critics are updated only when Critics are updated for a couple of times or more depending on parameter setting, using Polyak update strategy. Huber loss is adopted as the loss function to improve the model stability, although MSE loss is also implemented.

Replay memory is implemented so that the agent will be learning not only based on most recent experiences but also on past experiences to decorrelate the training data. The replay memory can provide batch data sampling once sufficient experiences are stored.

The reward function for Ant robot is predefined to be a combination of three factors. Firstly, for every time step the Ant survives (i.e. the centre of the Ant is within [0.2, 1]), 1 score is rewarded. Secondly, the difference between the X position of the ant between two consecutive time steps is counted as a reward (if positive) or penalty (if negative). Lastly, smooth control is rewarded, while large actions are penalised by a weighted sum of squared actions. The final reward is a sum of the three factors. The reward function aims to encourage the robot to survive and move forward in X direction as much as possible.

The training routine is coded such that the undiscounted reward for each episode during training is saved in a list, as well as the averaged reward for the past 10 episodes. All Actors and Critics are saved only when there is an increase in the episodic reward.

3.2 TD3 method for ANYmal robot

For ANYmal robot control, the TD3 architecture and the concept remain unchanged. Two Actors and two Critics, as well as their corresponding target counterparts, are initialised. Huber loss and memory replay are also implemented.

Several modifications are made to adapt TD3 agent to work on ANYmal robot. Policy smoothing is changed so that it generates a noise vector of size 12 instead of 8, as ANYmal has 12 action space

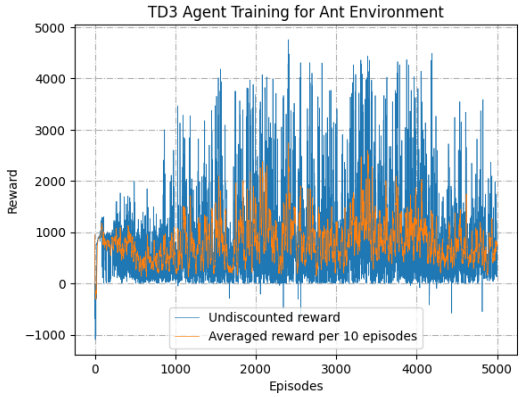

Figure 1: Reward of TD3 agent in Ant environment over 500 training epochs.

variables. A customised reward function is coded since the Jiminy environment does not provide a ready reward definition. Similar to the reward function defined in Ant, the directional and smooth action rewards are calculated and summed. The survival reward is still 1 per time step. However, a -10 death penalty is also added when the ANYmal robot falls to encourage its survivability. A customised wrapper class is also implemented to ensure that the output format from ANYmal environment is consistent with that of Ant environment, and no further changes are required to use TD3 method. For ANYmal robot, the training contains many more episodes to facilitate the robot to learn to stand, which was not an issue with Ant environment. Smaller batch size is selected to have finer experience learning and help with convergence. Conservative learning rates are selected for both Actors and Critics to combat the instability when learning such a complex robot model.

The secret tricks of TD3 that make it superior to vanilla DDPG have been discussed in section 2. To reiterate, it employs not only the methods used in DDPG, such as experience replay, fixed target networks and Huber loss, but it also adopted policy smoothing, delayed target networks update and Twin-Critics structure to manage the Q-function error exploitation issue with DDPG.

4 Results

4.1 TD3 agent performance for Ant robot

The TD3 agent for the Ant robot was trained for 5000 episodes. The undiscounted reward for each episode is plotted in Figure 1. A 10 moving average filtered reward history is also plotted. This result suggests that the TD3 agent learnt quickly to achieve a reward above 0, and the maximum reward learnt was approximately 4800. The learning remains fluctuated after approximately 500 episodes.

To better observe the ability of TD3 agent, Figure 1 is zoomed to the range of the first 400 episodes. It is evident from Figure 2 below, that the agent learnt quickly in the first 100 episodes, resulting in the reward increasing from below 0 to above 1000.

To further demonstrate the performance of the TD3 agent, a performance comparison was made using the trained TD3 agent against applying randomly sampled actions to Ant robot, for 200 episodes separately. The results are illustrated in Figure 3 below. It is evident that with randomly sampled actions, the Ant robot rarely achieved a reward above 0. The mean reward for TD3 agent on Ant control is 1309.8, whereas random actions give a poor average reward of -57.7 over 200 episodes. The maximum achieved reward for the TD3 agent is 4654.7 and for random actions, 36.8 on average.

4.2 TD3 agent performance for ANYmal robot

Learning an optimal policy for the ANYmal robot is a significantly more challenging task compared with the work done for the Ant robot. The learning continued for 200,000 episodes, as shown in Figure 4 below. The learning was considerably slower than TD3 agent for Ant, and the slope of undiscounted reward for ANYmal increased gradually until training for approximately 140,000

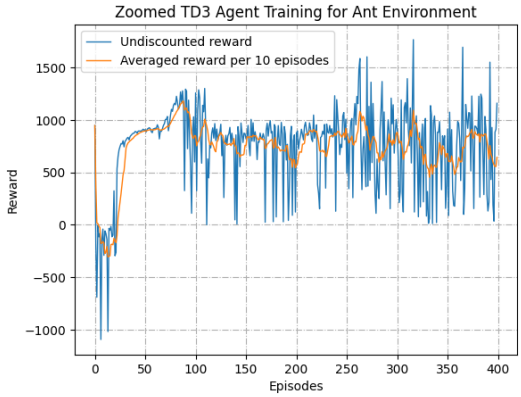

Figure 2: Zoomed reward of TD3 agent in Ant environment over 500 training epochs.

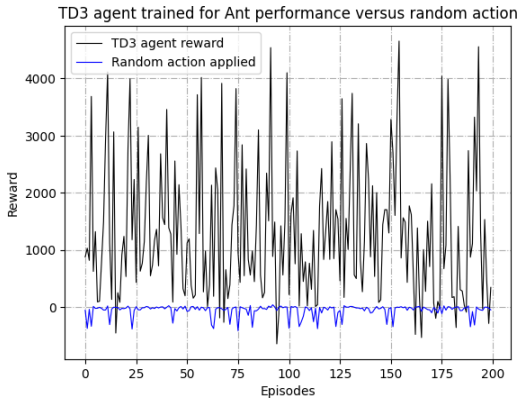

Figure 3: Reward of TD3 agent in Ant environment compared to a random agent.

episodes. Most of the time spent was for the Actor to learn a policy that can generate episodic reward above 0, meaning for the ANYmal robot to barely survive for no more than 15 20 time steps, considering the death penalty is -10. Additionally, even for the averaged reward, the fluctuation is more dominant than that for Ant TD3 agent training, suggesting an unstable learning process.

The challenges for a stable convergence come from the battling between maximising the reward for survival and maximising the reward to move. Unlike the Ant robot, whose survival is not an imminent issue even when random actions are applied, the ANYmal robot would collapse instantly if inappropriate actions are applied. So when survival is maximised, moving will be hindered, and vice versa. The fluctuation of reward is caused by the TD3 agent trying to find optimal policies to maximise both rewards in a complex environment.

However, the TD3 agent does an excellent job of making ANYmal robot survive. Using the agent trained after 200,000 episodes, it can be achieved where the ANYmal robot stands for 20 seconds until reaching the maximum calculation time steps. A baseline comparison is made to showcase the performance of TD3 agents for ANYmal and randomly sampled actions for 200 episodes. The results are illustrated below.

Table 1: Results for TD3 agent in ANYmal environment

Metrics Averaged	TD3 Agent	Random actions
Total reward	131.5	-57886.0
Survival time (s)	19.5	0.45
Travel distance	0.31	0.00033

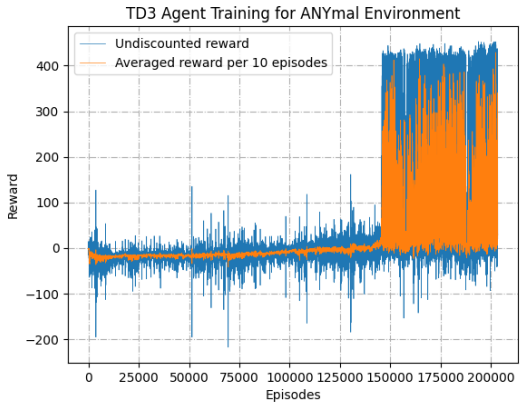

Figure 4: Reward of TD3 agent in ANYmal environment over 200,000 training epochs.

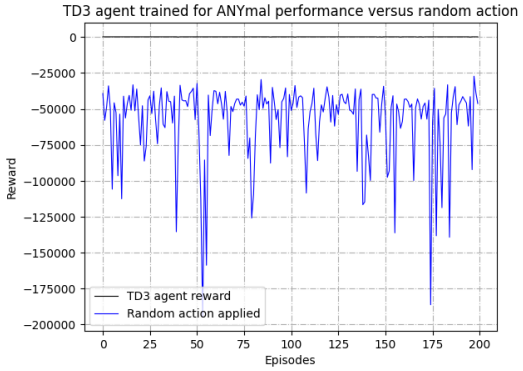

Figure 5: Reward of TD3 agent in ANYmal environment vs random agent.

The results reported in the table above are further illustrated in Figure 5 below. The overall performance with regard to the episodic reward of the agent is compared to randomly sampled actions. The random actions performance was vastly outperformed by the TD3 agent.

Due to the difference in magnitude, the TD3 agent performance was then plotted in Figure 6 alone, showing an average reward of 131.5. In most of the episodes, the ANYmal robot managed to survive for nearly 20 seconds, whereas immediate collapse was unavoidable for the random-actions-controlled ANYmal robot.

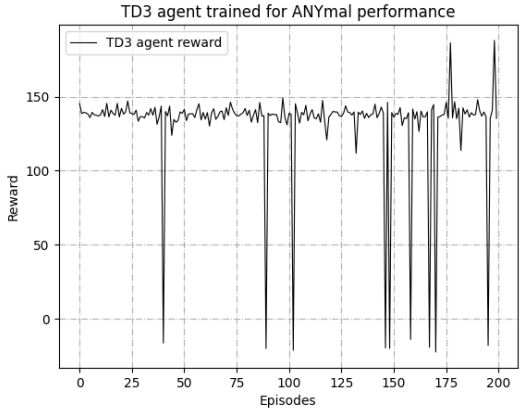

Figure 6: Reward of TD3 agent in Ant environment over 500 training epochs.

5 Discussion

We found that TD3 is a suitable algorithm for solving the Ant robot problem. The agent learnt quickly. It achieved increasingly positive rewards and progressed further throughout the episodes. The resulting agent performance has also shown convincing results; the Ant robot is able to quickly move to the edge of the environment that contains it. The TD3 agent showed less competence in controlling the ANYmal robot. As the results in the previous section suggest, the TD3 method struggled to learn a policy that generates positive reward until more than 150,000 episodes had elapsed. The trained TD3 agent is able to successfully control the ANYmal robot, enabling it to survive as long as 20 seconds, with obvious yet hesitant attempts to move forward slightly.

6 Future Work

The problem we chose is a very hard one, and there is a great deal of work we could do to improve our solution; improving locomotion in quadruped robots is still an area of active research. Given more time, there is, therefore, a wide range of different directions we could take this project.

For the algorithms we implemented, we have explored only a fraction of the possible hyper-parameter space, and a more exhaustive search of this space may yield improved training and results. We can also try a range of other loss functions. In particular, we have had to make trade-offs between prioritising survival, posture, forward movement and costs for other movements. Better trade-offs may exist that would improve training speed and results.

We could experiment further with different neural network architectures. Higher numbers of neurons per hidden layer and higher numbers of hidden layers may improve the algorithm's ability to learn at the cost of increased computation time and the risk of overfitting.

Given more time, we would be able to train our algorithms for a greater number of time steps, which may yield better policies.

Beyond the algorithms we used, we could explore both further extensions of DDPG, and more sophisticated algorithms such as Trust Region Policy Optimisation (TPRO) and Proximal Policy Optimisation (PPO).

For extensions of DDPG, we created an implementation of the Soft Actor Critic (SAC) algorithm but were unable to achieve improved results over TD3. We also experimented with prioritised mini-batches, rather than just drawing batches randomly from the memory buffer. Given further time, more processing power availability and experimentation, this might yield improved policies.

As mentioned above, other authors have had success with algorithms such as TPRO and PPO, which constrain policy updates to ensure that updates do not lead to catastrophic collapses in policy performance and ensure a continuously improving policy (Duburcq et al., 2022; Hwangbo et al., 2019). This could be an alternative path we could take on this project if given more time.

7 Personal Experience

We felt that this was a very challenging problem. We found the move from tabular methods to complex policy gradient methods challenging, both conceptually and to implement. Our problem choice was very challenging and not easily solvable with the simpler policy gradient methods, and we were forced to investigate deeply in the literature for more sophisticated methods. Our methods took a long time to run to achieve any progress and had a lot of hyper-parameters to fine-tune, meaning we were pressed for time. We also had difficulties since the gym environment was large and complicated and not comprehensively documented, meaning we had to figure a lot out ourselves. Also, we encountered a minor bug which would interrupt our training runs. I reached out to the package author on Github, and he responded almost immediately with a fix for the bug and was very helpful with our questions. We were pleased with the final results. Considering how challenging this problem was, we feel that being able to keep the ANYmal robot standing was a good achievement.

References

Achiam, J. Deep Deterministic Policy Gradient — Spinning Up documentation [Online], 2018. Available from: https://spinningup.openai.com/en/latest/algorithms/ddpg.html [Accessed 20 August 2022].

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W. 2016. Openai gym. ArXiv Preprint ArXiv:1606.01540.

Duburcq, A., Schramm, F., Boéris, G., Bredeche, N. and Chevaleyre, Y., 2022. Reactive Stepping for Humanoid Robots using Reinforcement Learning: Application to Standing Push Recovery on the Exoskeleton Atalante. arXiv preprint arXiv:2203.01148.

Duburq, A. 2022. Gym Jiminy [computer program], Available from: https://duburcqa.github.io/jiminy/ [Accessed 21 August 2022]

Fujimoto, S., Hoof, H. and Meger, D., 2018, July. Addressing function approximation error in actor-critic methods. In International conference on machine learning (pp. 1587-1596). PMLR.

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V. and Hutter, M., 2019. Learning agile and dynamic motor skills for legged robots. Science Robotics, 4(26), p.eaau5872.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015, June. Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Twin Delayed DDPG - Spinning Up documentation [Online]: n.d. Available from: https://spinningup.openai.com/en/latest/algorithms/td3.html [Accessed 20 August 2022].

Appendices

Appendix A: State space for ANT

The state consists of a single vector, with values as follows (in this order):

- Body Z position
- Body X Rotation/orientation
- Body Y Rotation/orientation
- Body Z Rotation/orientation
- Body W Rotation/orientation
- Left Front limb hip joint angle
- Left Front limb knee joint angle
- Right Front limb hip joint angle
- Right Front limb knee joint angle
- Left Hind limb hip joint angle
- Left Hind limb knee joint angle
- Right Hind limb hip joint angle
- Right Hind limb knee joint angle
- Body X linear velocity
- Body Y linear velocity
- Body Z linear velocity
- Body X angular velocity
- Body Y angular velocity
- Body Z angular velocity
- Left Front limb hip joint angle
- Left Front limb knee joint angle
- Right Front limb hip joint angular velocity
- Right Front limb knee joint angular velocity
- Left Hind limb hip joint angular velocity
- Left Hind limb knee joint angular velocity
- Right Hind limb hip joint angular velocity
- Right Hind limb knee joint angular velocity

Appendix B: State space for ANYmal

The return state consists of two vectors: Q and V

Q: position

This field contains the position of the body an 3D space and the positions of each of the joints in their own space (i.e. the joint angles)

The values in the vector are as follows (in this order):

- Body X position
- Body Y position

- Body Z position
- Body X Rotation/orientation
- Body Y Rotation/orientation
- Body Z Rotation/orientation
- Body W Rotation/orientation
- Left Front limb hip abductor/adductor joint angle
- Left Front limb hip flex/extend joint angle
- Left Front limb knee flex/extend joint angle
- Left Hind limb hip abductor/adductor joint angle
- Left Hind limb hip flex/extend joint angle
- Left Hind limb knee flex/extend joint angle
- Right Front limb hip abductor/adductor joint angle
- Right Front limb hip flex/extend joint angle
- Right Front limb knee flex/extend joint angle
- Right Hind limb hip abductor/adductor joint angle
- Right Hind limb hip flex/extend joint angle
- Right Hind limb knee flex/extend joint angle

V: velocity

This field contains the velocity of the body an 3D space and the velocities of each of the joints in their own space (i.e. the joint angles).

- Body X linear velocity
- Body Y linear velocity
- Body Z linear velocity
- Body X angular velocity
- Body Y angular velocity
- Body Z angular velocity
- Left Front limb hip abductor/adductor joint velocity
- Left Front limb hip flex/extend joint velocity
- Left Front limb knee flex/extend joint velocity
- Left Hind limb hip abductor/adductor joint velocity
- Left Hind limb hip flex/extend joint velocity
- Left Hind limb knee flex/extend joint velocity
- Right Front limb hip abductor/adductor joint velocity
- Right Front limb hip flex/extend joint velocity
- Right Front limb knee flex/extend joint velocity
- Right Hind limb hip abductor/adductor joint velocity
- Right Hind limb hip flex/extend joint velocity
- Right Hind limb knee flex/extend joint velocity

Appendix C: Ant Environment Training Architecture and Hyper-Parameters

The parameters of the Actor-Critics model and the training parameters are selected with trial and error. The training parameters are presented in the table below:

Table 2: Training hyper-parameter values for Ant

Parameters	Value
Epoch / Episodes	5000
Batch size	128
Replay memory size	5000
Start Policy Learning	1000
Policy Update Frequency / Delay	3
Actors Net Learning Rate	0.0001
Critics Net Learning Rate	0.0001
Gamma (discount factor)	0.95
Polyak update rate	0.05

Table 3: The Actor-Critic network is built using PyTorch. Table describes the network architecture for the Actor and Target Actor.

Layers	Size
Input	27
First hidden layer	256
Second hidden layer	128
Output	8

Table 4: Architecture for the critics.

Layers	Size
Input	35
First hidden layer	256
Second hidden layer	128
Output	8

Appendix D: ANYmal Environment Training Architecture and Hyper-Parameters

Table 5: The training hyper-parameters for ANYmal.

Parameters	Value
Epoch / Episodes	500,000
Batch Size	64
Replay Memory Size	5000
Start Policy Learning	1000
Policy Update Frequency / Delay	3
Actors Net Learning Rate	0.00001
Critics Net Learning Rate	0.00001
Gamma (discount factor)	0.95
Polyak update rule	0.05

Table 6: The Actor-Critic network is built using PyTorch. Table describes the network architecture for the Actor and Target Actor.

Layers	Size
Input	37
First hidden layer	256
Second hidden layer	128
Output	12

Table 7: Critics architecture

Layers	Size
Input	49
First hidden layer	256
Second hidden layer	128
Output	12

Appendix E: Benchmarking of trained TD3 agent for survival and travel distance on ANYmal

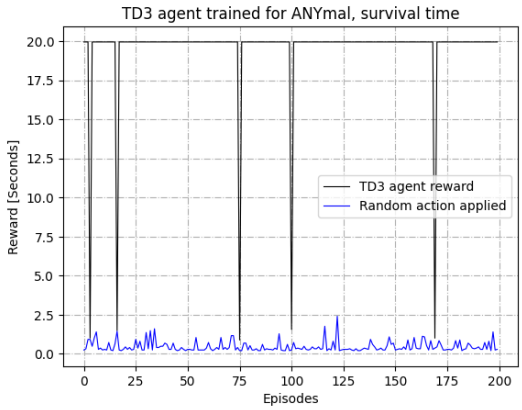

Figure 7: Survival time of TD3 agent in ANYmal environment over 200 training episodes.

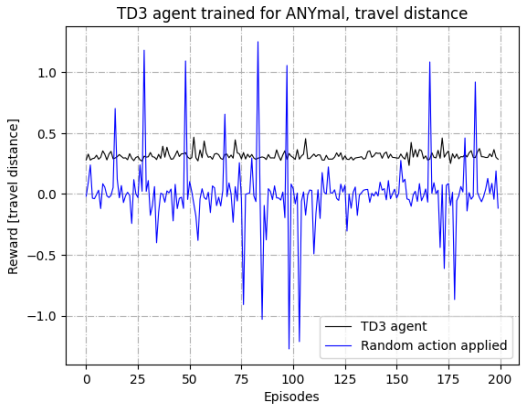

Figure 8: Travel distance reward of TD3 agent in ANYmal environment over 200 training episodes.