Learning optimal control policies for quadruped locomotion

Srinath Tankasala

***Abstract*—Classical control algorithms for quadruped locomotion have been well explored in literature. However, designing and tuning such controllers can be time consuming and requires a detailed kinematic and dynamics model of the robot.**

More recent work has been focused on developing model free agents that can learn to walk with enough training. In this project, training of a locomotion policy on a quadrupedal robot is explored. We explore the usage of multiple model free reinforcement learning policies and rate them based on sample efficiency and robustness of the final agent. While we successfully train the quadruped to walk directly using the action space, training it to walk on uneven terrain is prohibitively expensive and may require some form of environmental feedback like perception.

Firstly, we import the Laikago robot into the Mujoco environment as part of the system setup. Secondly, OpenAI's stable-baselines implementation of Actor-Critic methods is used to train the different agent policies. A good amount of effort went into shaping the reward to obtain a satisfactory walking gait from the quadruped robot. Finally, a comparison of the different model free training methods is done to see how they perform relative to each other in terms of robustness to perturbations and uneven terrain.

I. Introduction

Doing autonomous surveys/inspections of oil rigs and other plants is of great interest in the industry. It is currently done by humans and is a routine task that can be automated. For such applications, a robot must be able to navigate indoor and outdoor environments. A wheeled robot is incapable of climbing stairs and can have non-holonomic constraints, meaning that the actions (wheel speeds) cannot take it to all possible desired states. For example, the robot can have a turn radius constraint and also it cannot move sideways/laterally or yaw 180. This is why Quadrupeds are a universal choice for these types of surveys. They're agile and their ground locomotion is nearly unconstrained and can traverse any type of terrain while doing an inspection.

Dynamic locomotion of quadruped and biped robots is a challenging topic as there are multiple problems that need to be addressed for the controller to work well. A controller for a quadruped needs to be able to handle the aerial phases of the leg motion, short ground contact times and high speed swings to locomote successfully. Thus the locomotion is often charatecterized as gaits (trotting, galloping, pronking, bounding, etc.) of animals that can be reproduced on a robot.

Instead reinforcement learning (RL) algorithms can be used to actuate such robots through pure exploration and this a well studied area in RL. Here the user doesn't need to have any knowledge of the robot model or that of the environment. In this project we explore applying such RL techniques to learn a control policy that enables the robot to walk and navigate in different environments.

The RL agents used is model free and so it is exploration intensive. Given this is a continuous domain problem, there exist many deep RL agents that can learn to convert the input observations from the robot sensors into motor actions for the robot. It's assumed that the dynamics of the motor controller is very fast and so it achieves the given action/torque instantaneously.

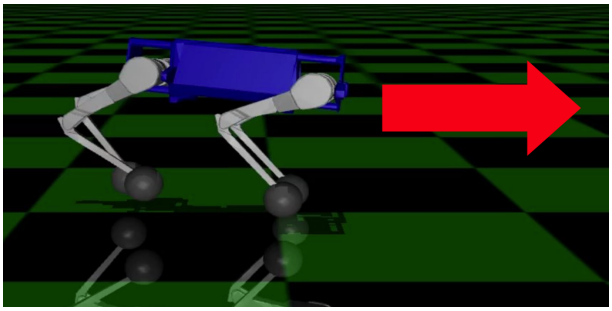

Fig. 1. Final trained policy learnt to move forward

After training the robot for a long time horizon, it was able to learn how to walk, figure I. In terms of sample efficiency, the SAC was found to do better than other policies. For brevity, we only show the comparison between SAC (off-policy) and TRPO (on-policy). Based on the rewards provided, the bounding gait emerged as the robots gait. To learn different gaits like trotting, pronking, etc., will require reward shaping and this is not explored here. This is why imitation learning methods are sometimes used as making such behaviours emerge manually may be difficult. It may require some form of Imitation learning which is beyond the scope of this project.

II. Related Work

A. Classical locomotion control

Several control strategies have been developed in literature that address them. The Boston Dynamics Wildcat [3] and the MIT Cheetah project are good examples of this. While classical controllers exist to control such robots, the engineer needs to have the full kinematic model to enable these classical controllers to work. This makes them difficult to scale up or the controller design is too system specific making it hard to transfer to another type of actuation (hydraulic) or design.

In literature Whole Body Control (WBC) is considered one of the most robust control strategies for locomotion. WBC

is a tracking controller, where the robots centre of mass (CoM) is made to follow a known gait trajectory. It does this by manipulating ground contact forces at the legs, and this is called the contact/impulse phase. There is no established method to calculating these ground contact forces given the CoM mass trajectory. Some have been proposed in [9] and [5] but even they also have issues dealing with short contact phases, i.e. at high speeds. [6] uses a hybrid approach for this, where an MPC method is used to infer the reaction forces. Since MPC is computationally intensive, the contact model used in [6] made several simplifications to run in real time, and this simplification doesn't hold at high speeds. Hence the Cheetah achieves a max speed of 3.7 m/s (Froude number 6) which is one of the best in the surveyed literature.

Once the reaction forces are known at each leg, the kinematics model is used to calculate the required motor torque commands. This is the impulse control phase of the leg. To calculate the torque command when the leg is in the swing phase, the foot trajectories need to be inferred as well. This is not straightforward and several planners exist for this in literature for this such as the one used in [11].

Overall given the above considerations, classical control methods have several limitations. Using a controller with lots of hierarchy requires the lower level controllers to run at very fast rates. This can make the setup expensive. For example, in the Cheetah controller, the motor control runs at 40 kHz. Secondly, it's cumbersome to tune such a controller hierarchy on the robot, and special tests need to be performed for it.

The major advantage of the classical approach is that it provides performance guarantees for the controller. So given *any* robot state, one could exactly compute the action that would be taken by the controller. Thus the user can do some tuning to ensure the performance stays in acceptable limits. For instance, it would be easy to ensure the controller remains compliant in all states with this approach.

B. Learning based locomotion

There has been siginificant work in recent literature on using RL based methods to obtain locomotion policies for legged robots. There exist two RL approaches to learning locomotion. The first approach is to use an imitation learning method to make the robot locomote like an equivalent animal [2], [10]. Such methods can use behavioural cloning approaches to learn the locomotion policy. Though it is sample efficient, it requires collecting a lot of data to achieve good performance. Such policies are also more robust and have good transfer performance. It can also be used to learn different gaits as done in [10]. In Peng et al. [10], they successfully translate the motions from a dog onto a quadruped (Laikago). They use a sim-to-real approach where the agent is trained on expert data gathered from the dog and then a domain transfer is applied to make the robot successfully perform in the real world.

The second is to learn from direct exploration of the environment with a model free agent, as done in [4], [7]. This is very sample inefficient and requires lots of exploration. Hence, it doesn't require any data collection process but the final policy is not robust. Many approaches like Meta-Learning [12] and Bayesian optimization [1] have been shown to increase the robustness of such agents. The learned policy also doesn't offer performance guarantees. So if it encounters an unseen state not in the sampled dataset, there is no guarantee on the action the agent will take. Thus the agent needs some interaction in the new environment and should collect online data to adapt well.

III. Data

A. Quadruped robot model:

The Laikago robot from Unitree Robotics is used in this work. All of its specifications are open sourced and it is widely used in literature. The specifications of the robot for this work were taken from the Laikago github repo. Also, since this robot hardware is available in the UT research labs, it opens the possibility of applying the results learnt here onto the quadruped hardware.

B. Simulation environment

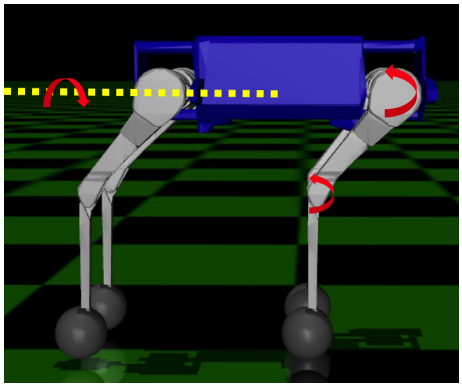

Fig. 2. Laikago joint rotation axes

The openAI MuJoCo simulation environment is used in this project. MuJoCo achieves state of the art performance in capturing contact dynamics. It is easy to use and a popular opensource tool for simulating quadruped locomotion. We use Mujoco-py package [8] to interact with the simulator and obtain experimental data.

It is very easy to mount sensors on the robot model in Mujoco. It can be done by making simple additions to the model XML file. Using the appropriate sensor description from the XML API documentation we mount contact sensors and IMU's. Perception based sensors were not used for this project. However, they will be essential for better environmental feedback, especially for complicated tasks. There are a wide range of actuators available in Mujoco. For this project we use the motor torque actuator at each joint.

Each leg of the robot has a hip, thigh and calf joint giving rise to 3 DoF. So the robot has 12 actions. For specifying the full position of the robot, we need 3 variables to specify CoM (Centre of Mass) position and 4 variables to specify its quaternion orientation.

The simulation step size is set to 0.002s and we simulate for 2000 time steps, for a total of 8 secs (2 frame skips included in calculation).

$$No.\ of\ variables\ for\ robot\ position = 12 + 3 + 4 = 19$$

Similarly, we have 12 joint velocities and 3 translational velocities along with the angular velocity vector.

$$No.\ of\ variables\ for\ robot\ velocity = 12 + 3 + 4 = 19$$

Hence the robot pose and velocity requires 38 variables to be fully defined.

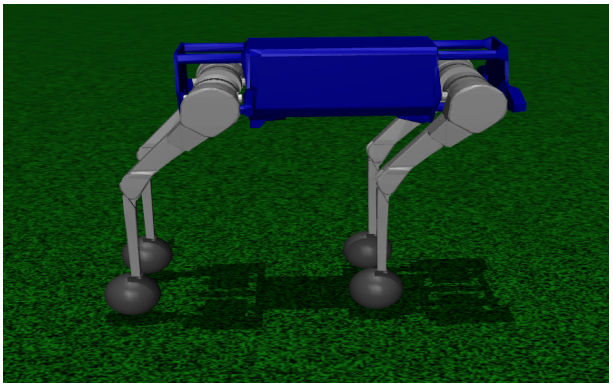

Fig. 3. Sample terrain generated by using White noise as height field

C. Learning environment

For the agent training, we use the OpenAI Gym environment. Different agent types can be created in Gym, such as TRPO, DDPG and SAC, etc.

The performance of the agent is determined by how stable its gait is along with the maximum speed it can achieve. We evaluate how the agent is able to generalize/adapt to different environments/terrains. We also evaluate how well the agent is able to reject force perturbations.

Mujoco and Gym can interact with each other and this python interface is widely used. To import Mujoco environments into Gym, a class interface was written based on []. This required writing some methods like step(), reset(),etc. Once this interface was setup, the agent being trained in Gym directly obtains the required data from the Mujoco simulator. We used this to train two different agents, SAC (off policy) and TRPO (on policy). The robot starts at the origin for each episode. We terminate the episode if the pitch or roll exceeds 40 degrees and reset the robot.

IV. Method

A. State space

We model the environment as a partially observable Markov decision process (POMDP). The control agent takes observations from the environment through different sensors as input. The output of the control agent is a 12-dimensional vector of normalized action values. The environment is deterministic and the action provided is enough to determine the future state of the robot and environment. In practice this is generally not the case and there is some stochasticity caused by actuation, sensor noise and uncertainty in environment parameters. These are not considered for this project.

B. Policy representation

We propose directly learning joint action based on the robots forward progress. We use the family of Actor-Critic agents to train the agent. In, [4], they train a half cheetah model to walk using this method and so we choose a similar strategy.

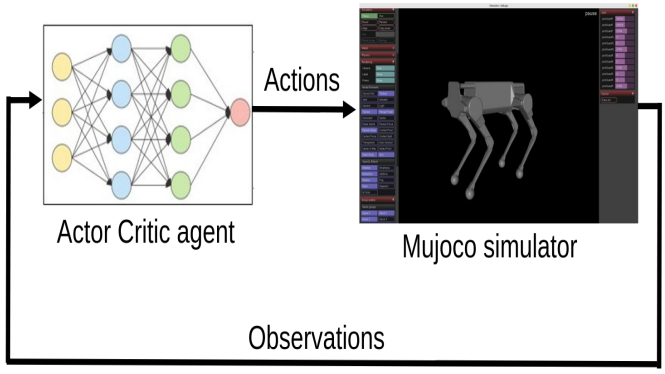

Fig. 4. Training setup

The choice of observations and reward functions is important for the agent to learn well.

C. Reward Design

The reward design should be non-sparse and the observations as informative as possible. Based on [], we specify a simple reward function of forward velocity (v_x). We apply some additional rewards and penalties to accelerate the training process.

$$R(t) = c_v \cdot v_x(t) - c_\theta \cdot \theta^2 + c_x \cdot e_x^2 - c_\tau \cdot \tau^2 - c_h \bar{h}^2 + c_{alive} \quad (1)$$

The co-efficients c_v, c_θ, c_x, c_τ, c_h must be chosen appropriately. θ is the total inclination of the robot with respect to the global Z-axis. τ is the exerted motor torque and e_x is the projection of the robots X axis vector onto the global X axis vector. c_{alive} is a constant reward provided to the robot for staying alive.

D. Observation space:

We should be able to infer the robot state and any feedback from the environment from our observations. Given this is a POMDP problem, we use many observations to capture details of the entire state. While it would be better to use perception for environmental feedback, it is not done here. Namely the observations are,

Entire robot pose and velocity, i.e. all 38 variables for position and velocity
Ground contact sensors for environmental feedback
Acceleration of the CoM

Roll, pitch, and yaw rates of the torso along with angular accelerations

Angular accelerations of the hip, thigh and calf joints for each leg

Action values (torque for each joint) from the previous time step

The observations can be used to reconstruct the complete state of the robot and the environment making it a fully defined problem. Adding more observations helps the robot generalize to different environments. More observations than the above were added making it 100 variables. However the above observation set was enough to get the robot to learn locomotion skills.

Of great significance is the robot pose and velocity observation. Without that, the robot was unable to learn even the simplest actions, such as standin up, etc. An ablation study of the different observations is not presented here for the sake of brevity. The robot pose and velocity along with ground contact forces were the most imporant observations to help the robot learn.

E. Action space:

The agent generates 12 actions normalized between -1 and 1. After multiplying with a scaling factor (gear ratio), these actions correspond to the joint torque signals for the revolute joints. The overall joint torque bounds are gathered from the specification of the motors used in the Laikago robot. Laikago has different torque limits and motion range for each joint. For this report the joints were restricted below their maximum allowable values. This was done to reduce the search space of the agent and explore only stable configurations.

For all four legs, the initial values for the hip and knee joint angles are set to vlaues that keep the robot in a stable upright configuration. This configuration is used to set the neutral positions of all the joints i.e. their 0 positions.

F. Training:

We use the stable-baselines package which has an implementation of the Soft-Actor Critic and TRPO algorithms. It is built for OpenAI Gym environments and can be directly imported to train our model.

For the SAC we use a 2x64 layer MLP to implement the actor critic with a ReLU activation function. The temperature and entropy coefficient parameter are automatically calculated based on the algorithm in [**?**]. We update the Q function every 100 samples with a mini-batch size of 64. The policy learns to walk well within $1.5\text{x}10^6$ samples.

For the TRPO we set the entropy coefficient to 0 and the max. KL loss as 0.01. We update the Q function every 1000 samples using 10 iterations for calculating the conjugate gradient. It took more than $5\text{x}10^6$ samples for the agent to walk well.

V. Results

We compare the performance of the two different trained agents in terms of forward velocity, pitch and yaw errors. The comparison of the final trained agents is shown below:

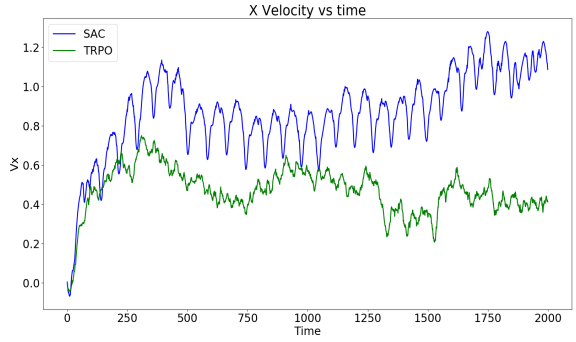

Fig. 5. Comparison of x velocities

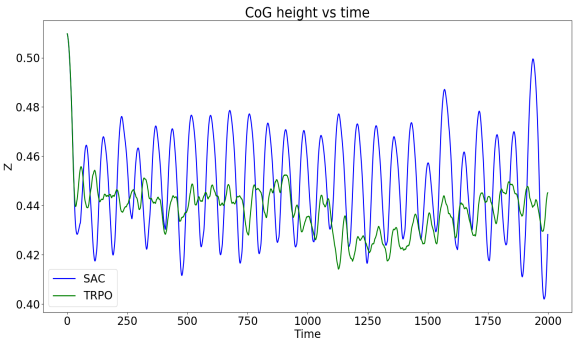

Fig. 6. Comparison of height variance

As can be seen in figures 5 to 8, the SAC policy clearly outperforms the TRPO agent in all metrics except the variance in height. Tuning the reward parameters only marginally increased the forward velocity before the agents fell. The SAC achieved a maximum velocity of 1.4m/s amongst all simulations. Ensuring pitch and yaw stay in reasonable bounds ensures that the agent learns quicker rather than exploring a bigger state space and unstable configurations.

The cumulative rewards for each of the agents during the training phase is shown in figure 9 and 10. Clearly the SAC

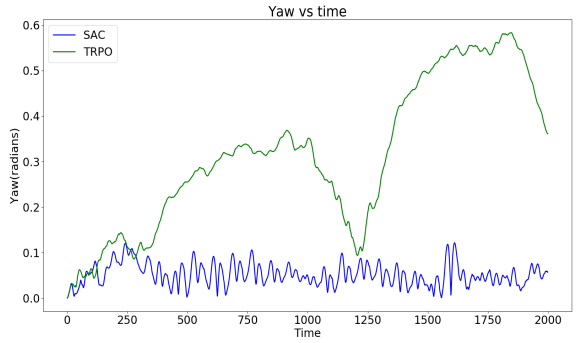

Fig. 7. Agent yaw with time

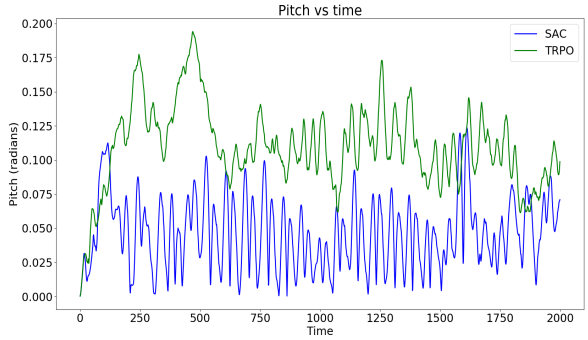

Fig. 8. Pitch comparison of learned policies

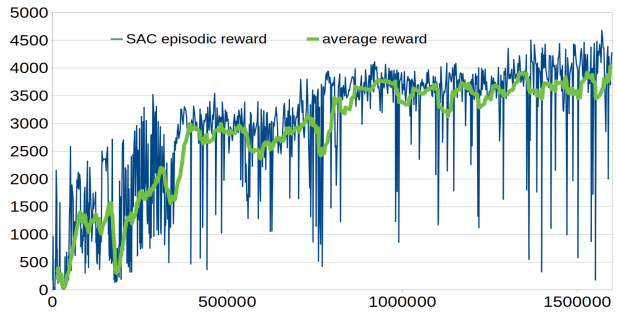

Fig. 9. SAC rewards vs time steps

is able to learn faster than the TRPO algorithm. It should be noted that the choice of hyperparameters for the different agents influences the performance finally obtained agents. So a fair comparison may be difficult but nonetheless, on-policy optimizations are in general sample inefficient compared to SAC.

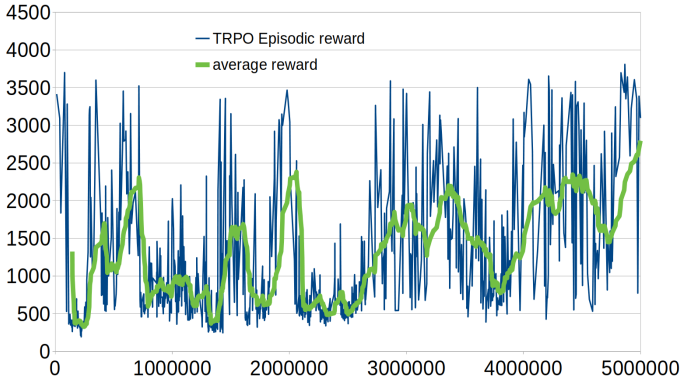

Fig. 10. TRPO rewards vs time steps

Another observation of the agent behaviour is the wide variation of actions to the extremes. In figure 11, we plot the joint commands in the Front Right (FR) leg for both agents. Notice how they keep moving between -1 to +1 rapidly. This is possibly because the agent is trying to get as much reward by going fast. Hence it tries to exert as much effort as possible. Balancing the c_v term and c_τ is very difficult. The agents

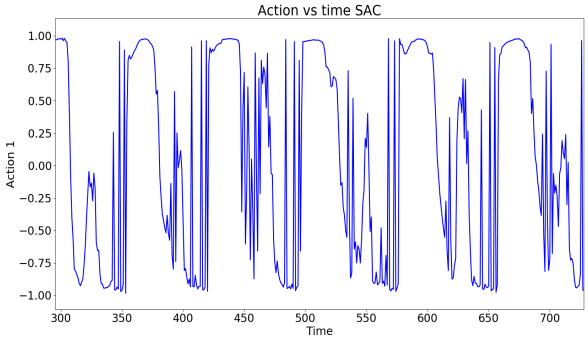

Fig. 11. FR calf action vs time

are quite sensitive to them, increasing c_τ penalty causes the robot to not take any action at all and reducing it results in even greater variance in actions. For simulation purposes, the robot will be able to perform this action but this agent cannot perform well on a real robot. Thus some reward shaping is require to make the agent actions smoother and implementable on a real robot.

A. *Policy robustness*

To measure policy robustness we use 2 different metrics:

Change in terrain

Change in mass (simulating payload like sensors, etc.)

The policy robustness is measured by the distance travelled in these new scenarios. The baseline is 7 metres in 200 timesteps for the original simulation. We only evaluate the robustness of the SAC algorithm here. We calculate the average distance travelled by SAC in 5 episodes.

For the terrain test, we create a height field based on Gaussian noise as seen in figure 3. The variance of the Gaussian noise is increased to simulate more uneven terrain.

For the mass test, we simply add a point mass directly above the CoM location of the torso. This simulates addition of sensors or a manipulator arm on the robot.

As can be seen in figures 12 and 13 the perfomance of the policy is very good until a certain threshold after which it precipitously drops.

VI. Conclusion

In conclusion, we were able to successfully learn a locomotion policy from pure exploration of the environment. The learnt policy was very brittle and did not generalize well to changes in environment. This can be mitigated in two ways. Since the policy was applied as is, i.e. a zero shot transfer, it performed very poorly in the new environment. We could allow it to collect data online for the new environment and retrain itself with some experiments. The second method is to train a more robust policy by getting more environmental feedback, like perception.

The second observation is the saturation of actions to the extremes. This needs to be reduced by tuning the reward better to achieve smoother actions. This reward could also be learnt

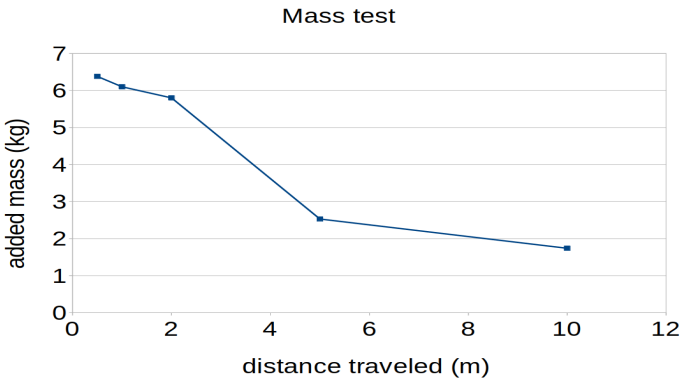

Fig. 12. Mass test performance of SAC policy

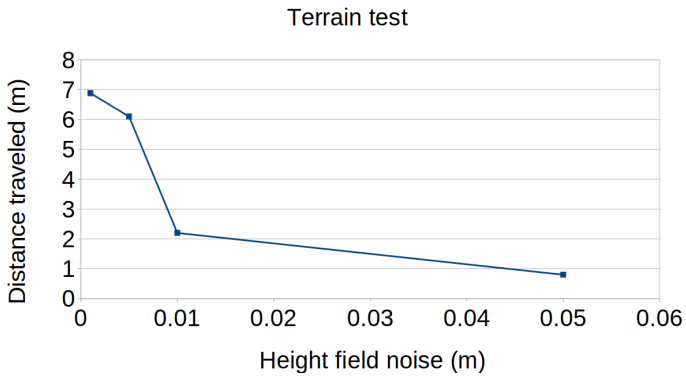

Fig. 13. terrain test performance of SAC policy

using a maximum entropy IRL method. This method would also allow different gait types to emerge from the training process. In this work only the bounding gait was successfully reproduced.

We compared two different methods for learning the the agent policy. We use an off-policy method (SAC) and on-policy (TRPO) to train the agent. We found that the SAC was much more sample efficient and robust compared to TRPO. More robustness measures can be tested in the future like adding force perturbations, etc.

Finally this whole setup can be imported into Robosuite to take advantage of its extremely modularity and rich environment options. This is left as future work.

REFERENCES

[1] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[2] Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[3] Boston Dynamics. Introducing wildcat. boston dynamics. URL https://www.youtube.com/watch?v=wE3fmFTtP9g.

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

[5] Donghyun Kim, Jaemin Lee, J Ahn, Orion Campbell, Hochul Hwang, and Luis Sentis. Computationally-robust and efficient prioritized whole-body controller with contact constraints. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[6] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *CoRR*, abs/1909.06586, 2019. URL http://arxiv.org/abs/1909.06586.

[7] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[8] openai. Mujoco-py. https://github.com/openai/mujoco-py, 2013.

[9] Hae-Won Park, Patrick M Wensing, and Sangbae Kim. High-speed bounding with the mit cheetah 2: Control design and experiments. *The International Journal of Robotics Research*, 36(2):167–192, 2017.

[10] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 07 2020. doi: 10.15607/RSS.2020.XVI.064.

[11] Ludovic Righetti and Stefan Schaal. Quadratic programming for inverse dynamics with optimal distribution of contact forces. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 538–543. IEEE, 2012.

[12] Wenhao Yu, Jie Tan, Yunfei Bai, Erwin Coumans, and Sehoon Ha. Learning fast adaptation with meta strategy optimization. *IEEE Robotics and Automation Letters*, 5 (2):2950–2957, 2020.