Gait changes in quadrupedal robots in varied gravity

Caspar Schucan

under the direction of

Konstantin Delchev
Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences
Yordan Tsvetkov
University of Edinburgh

Research Science Institute
July 31, 2020

Abstract

Quadrupedal robots offer great versatility since they can reach more places than wheeled vehicles and can carry more than flying drones. This versatility gives quadrupedal robots the potential to become very important in the future. This combination of traits could be useful, especially in space exploration. Due to the more complex nature of walking compared to rolling, gravity could interfere with their performance in environments with different gravity. If we would want to send quadrupedal robots into space, we need to know how they react to the different gravity of outer space. In this paper we trained a quadrupedal robot using reinforcement learning in a simulated environment and then tested its different gaits over various gravities. We show how adjusted gravity changes the gait significantly, at least in simulated environments.

Summary

Walking robots have many advantages over their rolling or flying counterparts. They can reach more places than rolling robots and carry more than flying ones. This combination of abilities could make them very useful to send to space and use as rovers on different planets or other celestial bodies. The walking robot could reach more extreme places there than the wheeled rovers that already exist. The problem with walking robots is how complex walking is. Even just the difference in gravity from earth to mars could significantly disturb the motion of walking. Therefore, in this paper we look at a four-legged robot that, after learning to walk, is tested in different gravities to evaluate the effect of gravity on its ability to walk.

1 Introduction

Less than half of the landmass on earth is accessible to existing wheeled vehicles; however, humans and animals can reach nearly all places[1]. Therefore, it seems advantageous to build robots that have the same ability to traverse uneven terrain.

Due to their versatility and ability to reach remote places, four-legged robots are a topic of increasing importance in a vast array of different human activities. Quadrupedal robots or any walking robots could have the ability to reach most places we humans can access too, while wheeled vehicles struggle even with for us humans easy to climb stairs. Legged robots couple this ability to reach difficult to access locations with a higher carrying capacity when compared to flying drones, who share the walking robots ability to reach most places and even outstrip it in that regard. Due to these advantages over flying and rolling robots, walking robots could also play a major role in space travel. Quadrupedal robots are only starting to realize their potential right now. In the last couple of years we have seen four legged robots like ANYmal[2] or Boston Dynamics' Spot being developed for inspection or maintenance tasks in industrial environments.

Quadrupedal robots tend to have one big disadvantage; at least with conventional approaches, producing a stable gait requires a lot of expertise and manual tuning, since for every position of the robot a sequence of movements would have to be defined or at least have a rigid system of defined rules to determine the next action. This is further complicated when *dynamic walking* is desired, *dynamic walking* leaves the walking system without constant balance. The robot falls from stable positions to stable position, very much like we do when walking and letting our bodyweight fall on the foot in front of us. This is in contrast to *static walking* where the walking system always remains in balance and could stop at any point during a step and maintain the current positions. An alternative approach to these conventional methods involves machine learning and in particular *reinforcement learning*.

In this paper we propose using *reinforcement learning* to train a quadrupedal robot to walk dynamically in a simulated environment with earth-like gravity. The robot's gait, or walking style, is then compared over varying gravity. The effects of gravity on the gaits of dynamically walking robots could play an important role in sending them to space as these effects are difficult to study in a real world environment here on earth.

2 Reinforcement Learning

Reinforcement learning, in the context of machine learning, is in short how an *agent* can learn by trial and error, the agent being any system performing actions on its environment that change the state of said environment. Reinforcement learning uses a system of reward and punishment to help the agent learn. We reward any behaviors that lead to a desirable new state of the environment while punishing anything that leads to a non-desirable state of the environment. What such a desirable state of the environment looks like is determined by the so-called reward function, which by looking at certain observations from its environment determines how "good" a state is. An algorithm playing chess for example will be rewarded for winning a game and punished for losing one. Another important keyword in reinforcement learning is *observations*. Observations are the information about the environment that the agent uses to decide about its actions. With us humans, for example, this would be all our sensory information. Most of the time the observations don't contain all the information about the environment, we humans for example can't see higher or lower frequencies in the electromagnetic spectrum. We only have to observe the information important for the decisions process of what action we take next. In reinforcement learning, the goal is finding a an ideal *policy*. A *policy* is the set of rules or guidelines that determine the next action from the *observations*. This policy can be either deterministic or stochastic. An ideal policy then, is the policy that gives us the most expected reward over time. Reinforcement learning

is a type of unsupervised machine learning. This means we don't have to give the system labelled training data but it can generate and even assign the data a value or "goodness" with help of the reward function.

In the context of this paper, reinforcement learning has several advantages over conventional approaches. Reinforcement learning allows for minimal manual tuning and does not require you to have a precise understanding of what exact movements of the joints comprise walking. These two characteristics make reinforcement learning more attractive compared to conventional programming techniques.

There are many ways to implement the principles of reinforcement learning into an actual program. In this paper we used an algorithm called soft actor-critic[3]. The soft actor-critic, or SAC, uses the *Q-value function* to derive progressively better policies. The *Q-value function* as described in Equation 1 represents how big the expected accumulated reward, over period of time, from a certain state of the environment s, is if we take a certain action a and if we act according to a policy π.

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{\mathrm{E}}\left[R(\tau) \mid s_0 = s, a_0 = a\right][4] \tag{1}$$

The τ describes the series of actions that follow from policy π, called trajectory, and the R function describes all the reward that we receive if trajectory τ is executed. The SAC uses two separate neural networks that compute the soft Q-value function. Two neural networks are used to increase stability, since only one neural network tends to over- or underestimate the real Q-value over time. Having two networks that are used to improve each other stabilizes this over-/underestimating. These two neural networks essentially improve each other in the way that the loss, or how wrong the neural networks are, is defined as the squared difference between their outputs. Neural networks use this loss to then compute what to adjust to minimize said loss. From these estimated Q-values we can extract policies that maximize our expected reward when the Q-value networks get better and better. Besides The Q learning,

the SAC uses *entropy maximization*. This means that the reward function has a built-in component that rewards the agent for a policy with high entropy or randomness. High entropy helps the robot try out more different strategies so it does not accidentally converge on a local maximum.

3 Experiments

To build and train this quadrupedal robot, we used three libraries. We used OpenAI gym to construct an environment fit for reinforcement learning. OpenAi Gym is an open source library and crafted specifically to allow convenient construction and use of such reinforcement learning environments. To simulate the robot and its environment we used PyBullet. PyBullet is one of the leading open source simulation frameworks. It is easy to use and already has a sizeable community. To implement the machine learning part of reinforcement learning, we used stable-baselines. Stable-baselines offers an open source implementation of many popular reinforcement learning algorithms, including SAC.

Robots

During this project, we used three different robots. Firstly, we used a robot by our own design as seen in Figure 1. This robot is optimized for simplicity. All eight joints turn around an axis of the same orientation as the others. This would make it very difficult for this robot to keep balance after being pushed or even when trying to maneuver through hard terrain. This simple design can only work in the idealized simulated environment. The simplicity of the robot has the advantage that most other quadrupedal robots have all the same structures or similar ones present. This makes this robot potentially valuable if we want to make conclusions from the behaviour of this robot to the behaviour of other quadrupedal robots. The process of designing and coding our own robot had educational purposes too.

Its eight joints, four hips and four knees, are all revolute joints, meaning they have defined limits up to where they can turn.

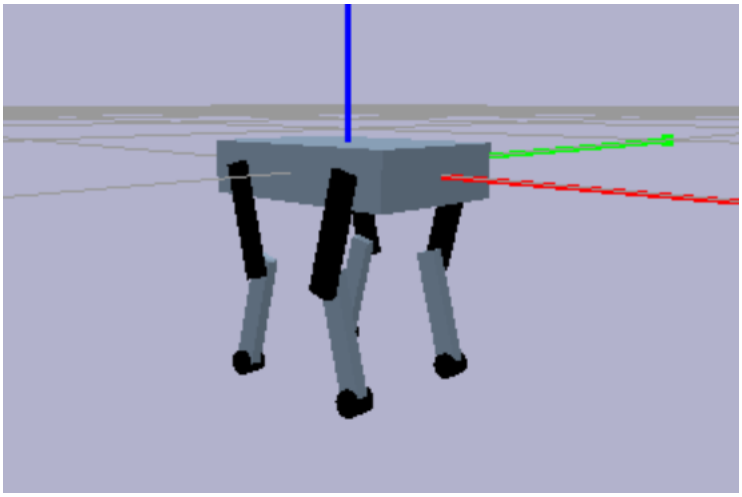

Figure 1: The self-designed robot

Secondly, we used a virtual version of the Laikago robot by Unitree, as seen in Figure 2. One important difference between our own robot design and Laikago is the addition of another joint on each leg. This joint is located at the hip and its axis of rotation is perpendicular to those of the other joints which allow for better reaction to forces from outside. This robot is included in the Pybullet library.

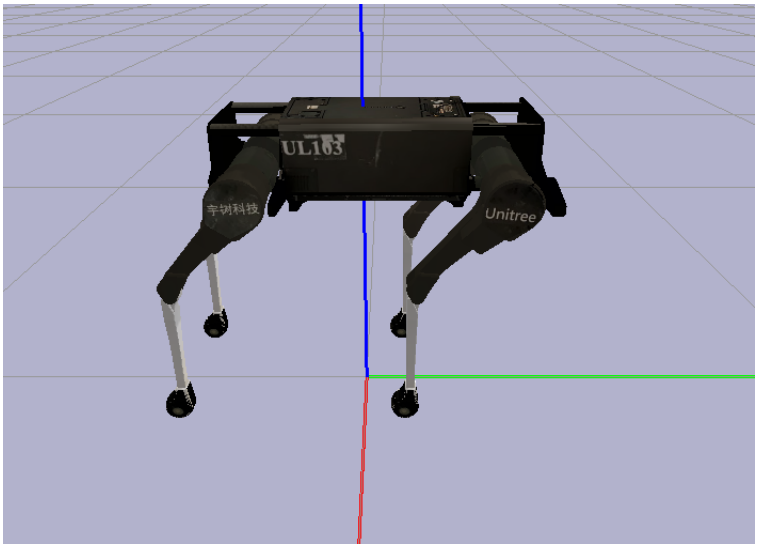

Figure 2: Laikago by Unitree

Lastly, we used a conventionally programmed robot, provided by my mentor Mr. Yordan Tsvetkov, as seen in Figure 3. The robot follows a program that calculates the next action from the current joint positions using a deterministic formula. This robot uses the same joints as the first one. The big difference being that it has been conventionally programmed and optimized to walk in earth-like gravity. We also trained this robot using SAC. It served as a more realistic version of the robot we design ourselves.

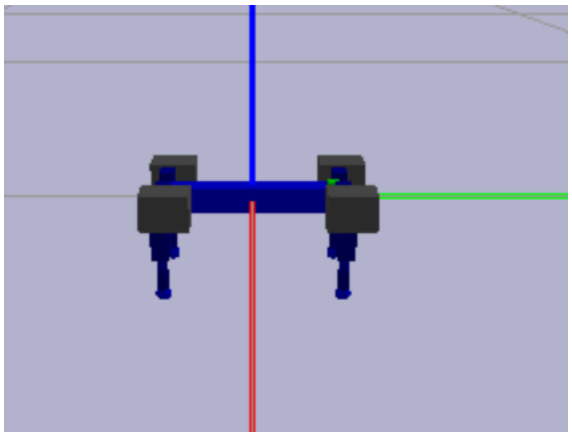

Figure 3: A robot used to test conventional approach

Actions

An action in our case consists of instructions to all joints what to do next. In this paper we used, PyBullet position control[5]. An action is defined in a list of length four or eight with the target positions of each joint. Even though we have at least eight joints on every robot used, it sometimes proves useful to only use 4 different instructions and give two of the legs the same instructions. A limit for maximum angular velocity of the joint is set in the environment at $5.23\frac{r}{s}$.

Observations

The observations are all the details that the robot knows about its environment to decide which actions it should take next. Because the robots surroundings are constant, a flat plane, no sensory information about its surroundings are necessary. We decided to observe the positions of all joints as well as the orientation of the torso around the x- and y-axes and the angular velocities of the torso around these axes. These observations should give the robot enough information to perform the task at hand[6]. The orientation and angular velocity of the torso around the z-axis is unnecessary since it is at least for our purposes not important whether the robot walks forward or backwards or even at an angle.

Reward

Two different reward functions were studied by us. The first reward function tried, as seen in Equation 2, to reward stability and velocity in a target direction. The latter is simply achieved by looking at how much distance was made in the last time step of Δx. The coefficient a is there to balance the importance of the velocity compared to the importance of the stability. Said stability is incentivized by a function that rewards slight tilts of the torso while punishing big tilts, or positions that are close to falling. The *roll* describes the angle of the torso around the x-axis that is part of the observations. The *pitch* is the angle

around the y-axis. We use the absolute values of these angles so that there is no difference in cases depending on which side the robot is tilted. The subtracted constants provide the aforementioned reward for balanced, positions.

$$r = a \cdot \Delta x - b \cdot (|roll| - 0.2 + |pitch| - 0.15) \tag{2}$$

The second reward function used, as shown in equation 3, kept the velocity element of the first reward function but focused more on efficiency instead of stability. The efficiency is incentivized by a term that punishes big angular velocities and torques in the joints. In the formula, this term is shown as the scalar product of the vector with all joint torques and the vector of all angular velocities. This reward function was proposed for dynamic quadruped locomotion by Jie Tan et al.[6]. We only used this reward function on Laikago.

$$r = \Delta x - w\Delta t \, |\tau_n \cdot \mathbf{q}_n| \tag{3}$$

Training and Tests

Models were trained for one million time steps. Training takes place in an environment with earth-like gravity. The neural networks used to estimate the Q-value function both consist of two layers of 256 neurons. The neural networks use an activation function called *ReLU*. ReLU is a very simple activation function where a neuron is activated or "fires" if the incoming values exceed a certain threshold. After training, the gait of a robot is observed and analyzed in four different gravities. We tested the robot in earth-like gravity as a control group and then look at how its gait changes in the lower gravities of the moon and mars and also test how it reacts to a higher gravity, which we test in an environment with Jupiter-like gravity.

4 Results

Using a conventionally trained robot, the effects of gravity are big. The robot can only walk in a stable manner in a range between $0.8g$ and $1.5g \pm 0.05g$. In all tested gravities the robot fell with the exception of the earth-like gravity it was built to walk in. This is very well documented by Table 1. This table shows how far the robot could walk in 40 seconds or until it fell down. Due to the deterministic nature of how the robot acts in every position, we don't need to look at multiple tries because the robot will behave the same way every time. The conventionally trained robot walks with the legs moving in diagonal pairs. This type of gait is very often seen in the animal world. It is not unlike the trot of a horse, although much simpler since the legs of horses have an additional ankle joint. The rough progression is seen in Figure 4. Important to note when comparing the different frame sequences is that the time passing between the pictures varies between the different robots tested.

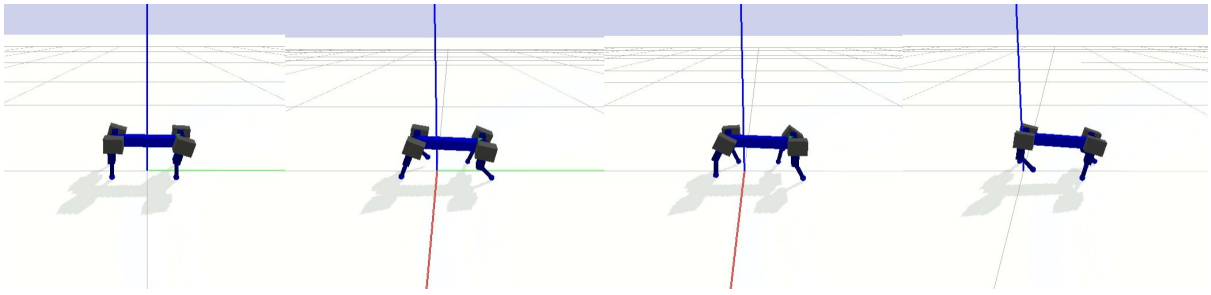

Figure 4: The conventionally programmed robot walking in earth-like gravity. Time between pictures equals approximately $\frac{1}{3}s$

Distance in 40s or until falling in cm	**Gravitational acceleration in $\frac{m}{s^2}$]**
57.9	24.79 (Jupiter)
1037.4	9.81 (Earth)
47.8	3.71 (Mars)
52.9	1.62 (Moon)

Table 1: The distance walked by the conventionally programmed robot in 40 seconds or until falling over, compared over different gravity

The conventionally programmed robot failing to adapt well to different gravity does not come as a big surprise. The sequence of movements is regular and is not dependent on the orientation of the torso and does not try to adjust this orientation to achieve balance. This inability to adjust to any changes of the environment indeed can be attributed to the relative simplicity of the robot and its gait.

When training the robot from Figure 4 using SAC we gave the same instruction to the diagonal pairs of legs just like in the conventionally trained version. The reward function used was Equation 2. The gait, as seen in Figure, looks vaguely similar to one produced by the conventionally trained version although it does not move its body over the legs like the conventionally programmed robot does. This is likely due to the fact that the reward function for this robot includes a term for stability, and moving the body over the legs is difficult to do while staying upright. Similarly to the conventionally programmed version, this robot failed to walk stably in all tested gravities, as is shown by the example of the robot falling in martian gravity in Figure 6, except in earth-like gravity where it was trained. Table 2 shows how the mean reward per episode changes when gravity is changed. This data cannot be directly compared to the other robots trained using a reward function of the same type as Equation 2, since the parameters a and b vary. An episode ends after 40 seconds or when the robot falls down. We average the reward over 10 episodes. This small sample size is sufficient

because while the robot does act stochastic, meaning somewhat randomly, it does always behave similarly.

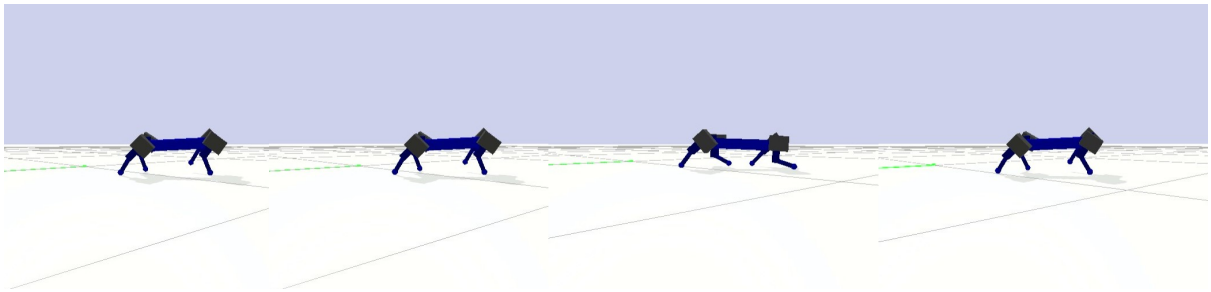

Figure 5: Robot walking trained with SAC and diagonally paired legs in earth-like gravity. Time between pictures equals approximately $\frac{1}{10}s$

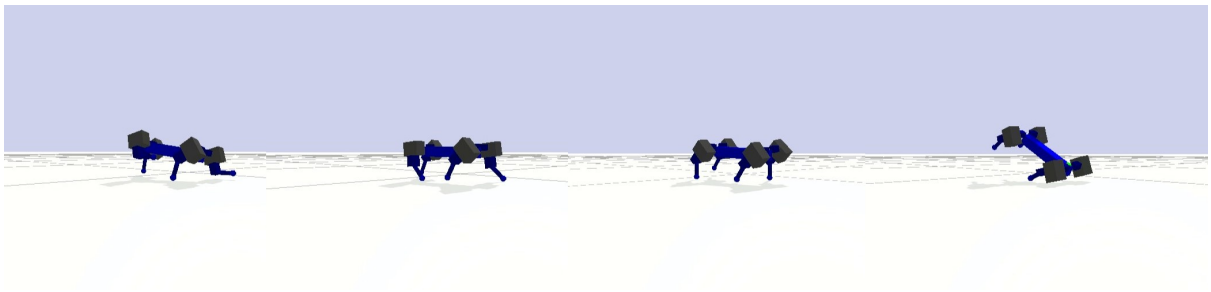

Figure 6: Robot trained with SAC and diagonally paired legs falling in martian gravity. Time between pictures equals approximately $\frac{1}{10}s$

Mean reward per episode	**Gravitational acceleration in $\frac{m}{s^2}$**
58.2	24.79 (Jupiter)
10700	9.81 (Earth)
194.8	3.71 (Mars)
907.1	1.62 (Moon)

Table 2: The mean reward per episode by a robot trained using SAC with diagonally paired legs, compared over different gravity

Using our own design, the biggest challenge was achieving any gait at all. Although, we have found robots with a stable walk, these walks use non-symmetric leg movements that

often appear jerky and irregular. In addition, these movements are often that small that the robot nearly seems to glide over the ground in a fashion not unlike a hedgehog or similar small quadrupedal animals. Due to the erratic nature of the walking, comparisons between gaits in different gravities are still difficult. Still, the robot trained using reinforcement learning and without paired legs seemed less affected by varied gravity and managed to walk in a stable manner for all tested values. The robot fell sometimes too, in all tested gravities. The mean reward per episode over the different gravities, seen in Table 3, shows us a very different picture to what we saw happening to how far the conventionally programmed robot could walk in forty second, as depicted in Table 1. It is important to know that the reward also includes a term for stability in this case, so there is no one-to-one correspondence possible, but the fact that the learned robot is able to generate more reward in lower gravity suggests at least some adaptability. This type of reward function seems to perform better when gravity gets really low. We have seen a rise in mean reward per episode when comparing mars- and moon-like gravity in the robot trained with SAC and paired legs too.

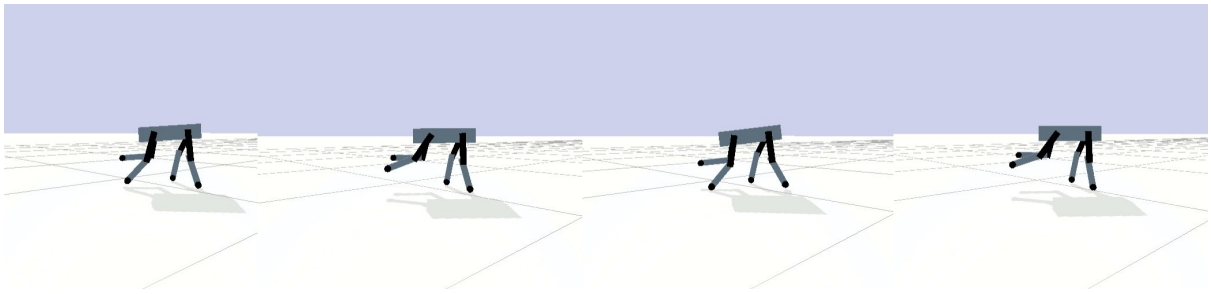

Figure 7: The robot of our own design trained using SAC walking in earth-like gravity.Time between pictures equals approximately $\frac{1}{30}s$

Mean reward per episode	**Gravitational acceleration in $\frac{m}{s^2}$**
920	24.79 (Jupiter)
3700	9.81 (Earth)
5197	3.71 (Mars)
7025	1.62 (Moon)

Table 3: How much the reward the robot gets on average in the first 40s, or until falling, over the different gravities

The reasons for our own robot's failure to walk in a regular fashion could be found in many places. These include but is not limited to a bad reward function, non-ideal size or activation function of the neural networks used, or a flawed robot design. The improvement of the mean reward per episode in lower gravity is most probably an effect of the specific reward function, but even if we raise gravity, and the reward per episode goes down, this drop is less significant than the one seen in the conventionally programmed robot, but we also saw the mean reward per episode drop very far in the robot trained with SAC and paired legs.

We did not produce enough experiments to produce a stable Laikago walk. Instead we decided to migrate to the robot provided by Mr. Tsvetkov and do more experiments using that robot

5 Future Work

In a first step, we have to improve the regularity of the walk of a robot trained using reinforcement learning in earth-like gravity. If the observed effects persist, one interesting possibility to mitigate said effects could be to randomize the gravity during the training phase so that the robot can acclimatize to a range of different gravities. It would also make sense to test very different learned robots to see if their relative adaptability when gravity

is varied holds true for a broader spectrum of reinforcement learning algorithms and reward functions. Another option to explore in the future would be to try changing from position control to torque control which is the dominant method of robot joint control, right now but is less intuitively understandable.

6 Conclusion

We have shown in this paper that quadrupedal robots change their behavior and performance significantly when subjected to varying gravity. We have also seen how robots that learned to walk using reinforcement learning. More specifically with the SAC algorithm and a reward function that prioritizes speed and stability, are able to better adapt to a changing gravity than conventionally programmed robots, although this effect is greater in a more erratically acting robot.

7 Acknowledgments

I would first like to thank my mentors Mr. Konstantin Delchev and Mr. Yordan Tsvetkov. Without their tips, feedback and always well thought through answers to my questions how simple they may be. I want to thank Dr. Jenny Sendova for her tutelage and providing endless energy in every meeting. I want to express my gratitude to the first week teaching assistants Shloka Janapaty and Albert Wang. I would also like to thank Caitlin van Zyl and Elijah Stanger-Jones who gave fantastic feedback on some of my paper's drafts. I want to thank Dr. Amy Sillman, all people working in the background to make this possible, RSI, CEE and MIT for creating this wonderful opportunity. I'd like to thank my counselor Basheer AlDajani and my fellow Rickoids. Last but not least, I want to thank my family for putting up with my schedule during their holidays.

References

[1] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

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

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[4] J. Achiam. Part 1: Key concepts in rl¶, 2018.

[5] X. B. Peng and M. van de Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–13, 2017.

[6] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.