UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Learning to walk on rough terrain and collision localization for legged robots with Machine Learning

Permalink

https://escholarship.org/uc/item/2gg4h581

Author

Shah, Chinmay Vijaybhai

Publication Date

2021

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning to walk on rough terrain and collision localization for legged robots with Machine Learning

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Engineering Science (Mechanical Engineering)

by

Chinmay Vijaybhai Shah

Committee in charge:

Professor Nicholas G Gravish, Chair
Professor John Tae Hyeon Hwang
Professor Michael T Tolley

2021

Copyright
Chinmay Vijaybhai Shah, 2021
All rights reserved.

The thesis of Chinmay Vijaybhai Shah is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

University of California San Diego

2021

DEDICATION

To my parents.

EPIGRAPH

The way positive reinforcement is carried

out is more important than the amount.

— B. F. Skinner

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Nicholas Gravish for giving me chance for the research and providing guidance.

VITA

2018	B. E. in Mechanical Engineering, The Maharaja Sayajirao University of Baroda
2021	M. S. in Engineering Sciences (Mechanical Engineering), University of California San Diego

ABSTRACT OF THE THESIS

Learning to walk on rough terrain and collision localization for legged robots with Machine Learning

by

Chinmay Vijaybhai Shah

Master of Science in Engineering Science (Mechanical Engineering)

University of California San Diego, 2021

Professor Nicholas G Gravish, Chair

Compared to wheeled robots, legged robots can provide a significant advantage in traversing complex, uneven terrain. The advantage is still unrealized because of the lack of effective algorithms that can give instantaneous reactions like biological systems. In recent years data-driven methods have gathered the community's interest due to the ability to learn rules/patterns from data. The black box data-driven methods remove the need for rule-based strategies while benefiting from the fast implementation. The data-driven method like Deep Reinforcement Learning (DRL) has the potential to make the system more reactive, efficient, and less reliant on the rules. The thesis examines the potential of DRL on rough terrain. While there is a lot of attention

for reinforcement learning for legged robotics, the many aspects like environment generation, episode termination criteria are undiscussed in the literature available. We also suggest strategies of generating random terrain with appropriate parameters, environment design, and we also prove the use of randomization on the terrain to guarantee higher performance.

Due to imperfect estimation of terrain by the perception system, for legged locomotion, leg collisions with the terrain are inevitable. Due to the potential of damage, the collision is an important failure mode and needs to be investigated. The black box data-driven methods remove the need for rule-based strategies while benefiting from the fast implementation. In this thesis, we present a temporal convolutional network based CDLnet (Collision Detection and Localization Network), a single neural network both for collision detection and localization. Due to the unique nature of the problem, the conventional loss functions can raise a significant error in the measurements. Therefore a new loss function called CDLloss is introduced. The CDLloss uses a combination of classification loss and special collision localization loss. Due to the ability to effectively extract information across time-domain, the network outperforms the baseline neural network with the same loss function and also outperforms the best performing physics-based method on minitaur leg. Then we discuss future work directions and potential use-cases of this method.

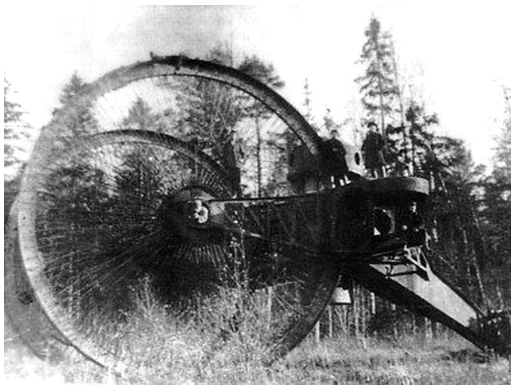
Chapter 1

Introduction

1.1 Legged Locomotion

Wheels are considered one of the earliest human innovations. It has a practical advantage on flat terrain, and many other innovations like tires and bearings made it the most crucial creation for humankind. One of the biggest problem with wheels is that it requires special and massive infrastructure like roads and tracks to be useful. Creating these type of massive infrastructures are one of the biggest and most celebrated projects in human history. It requires massive capital, planning, and human labor. These infrastructures then have to be maintained, cleared of obstacles.

On uneven terrain, the wheels are often time found not that useful. The general solution is to increase the wheel diameter. One infamous example of this is the tsar-tank[5]. It had 9 meters in diameter. They were so heavy that they started to sink into the ground. One can also look in nature and find that none of the animals have anything similar to the wheel. The problem of uneven terrain was solved by the evolution using legged locomotion. The legged locomotion is seen in the smallest ($40\mu m$) organisms like tardigrades [6] to biggest animals like elephants, from bipeds like kangaroos to millipedes (30 to 400 limbs) [7].



(a) Historic picture of Tsar-Tank

(b) Mountain Goat on verticle cliff

Figure 1.1: Example of failure of wheeled system and extreme example of legged locomotion from mountain goat [1]

1.2 Related Work

1.2.1 Commercial Legged Robots

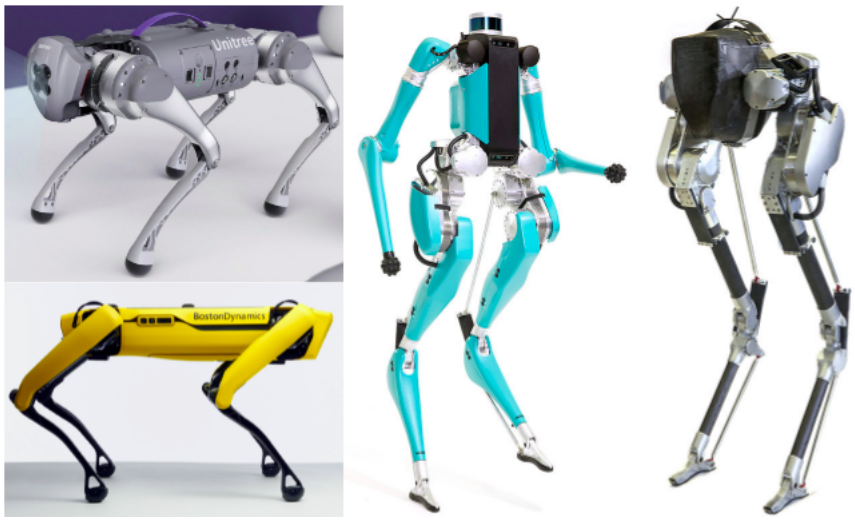


Figure 1.2: From Left Go1 (Unitree), Digit and Cassie(Agility Robotics) Boston Spot (Boston Dynamics)

Due to the advantage over wheeled robots, continually decreasing hardware costs and the availability of better hardware have caused the surge in legged robotics. The price of commercially

available legged robots has substantially come down. For example, the Boston dynamics' factory survey quadruped robot named spot is now available for the 75,000$. Recently the Go1, a quadruped robot from Unitree robotics, costs around 8,500$ with a programmable interface. Not only quadrupeds but also bipedal robots like Digit and Cassie are being commercialized.

1.2.2 Legged Robots Control Algorithms

There is an explosion of research in legged robotics control algorithms with two opposing approaches. These approaches solve the same control problem in a fundamentally different way. Both

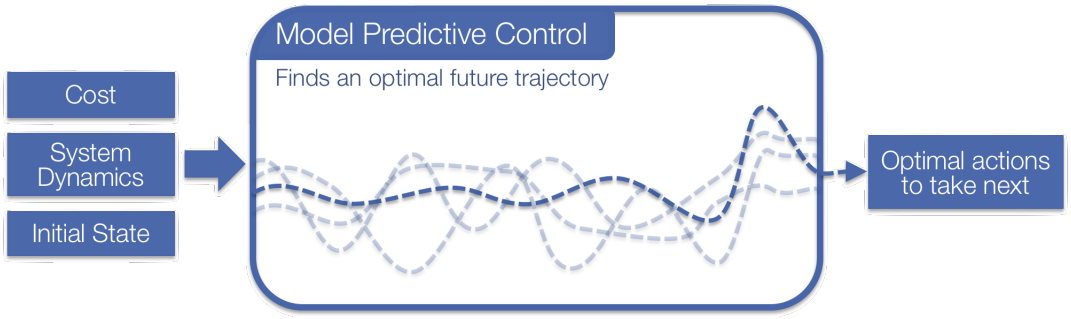


Figure 1.3: Example of Model Predictive Control [2]

1. **Model Predictive Control (MPC)**: The model predictive control a.k.a. MPC is class of Control algorithms that utilize an explicit model to predict the future response of dynamical system[8]. The Model Predictive Algorithm at each time interval optimize the future of the control system states according to some objective or cost function.

   The MPC algorithm is widely used in chemical plant processing. Due to the high reliability of output and generation of smooth trajectories, the MPC algorithms are the first choice for controlling most sophisticated robots. In reference to legged robots new whole body approaches like [9] exist that optimizes the objective functions with constraints for entire robots.

2. **Reinforcement Learning (RL)**: The reinforcement Learning algorithm is class of machine learning algorithm that maximizes reward or in other sense minimizes cost function by taking actions according to optimal policy π. A recent trend is to use neural networks for the policy. The neural network allows to used direct use of images to map out actions[10]. With the possibility of transfer learning of neural network models [11], reinforcement learning has the potential to perform well on unstructured high dimensional data. For legged locomotion, learning to walk in the simulator and transfer it to a real robot called sim-2-real is gaining recognition[12].

1.3 Minitaur

Figure 1.4: Minitaur form Ghost Robotics

The Minitaur was originally invented by ghost robotics [13]. The minitaur is direct drive quadruped with 8 D.O.F. and weights around 5 kg. Each leg uses 2 T-Motor U8 series motors. The each motor uses 12V power supply. The motors thermally sustain $100°C$ rise in steady state

with current range 3.25-9A. The upper link and lower link dimention of each leg are 11 and 20 cm respectively. The minitaur can be equiped with lidar and cameras and can be controlled by remote control.

1.4 Minitaur Pybullet

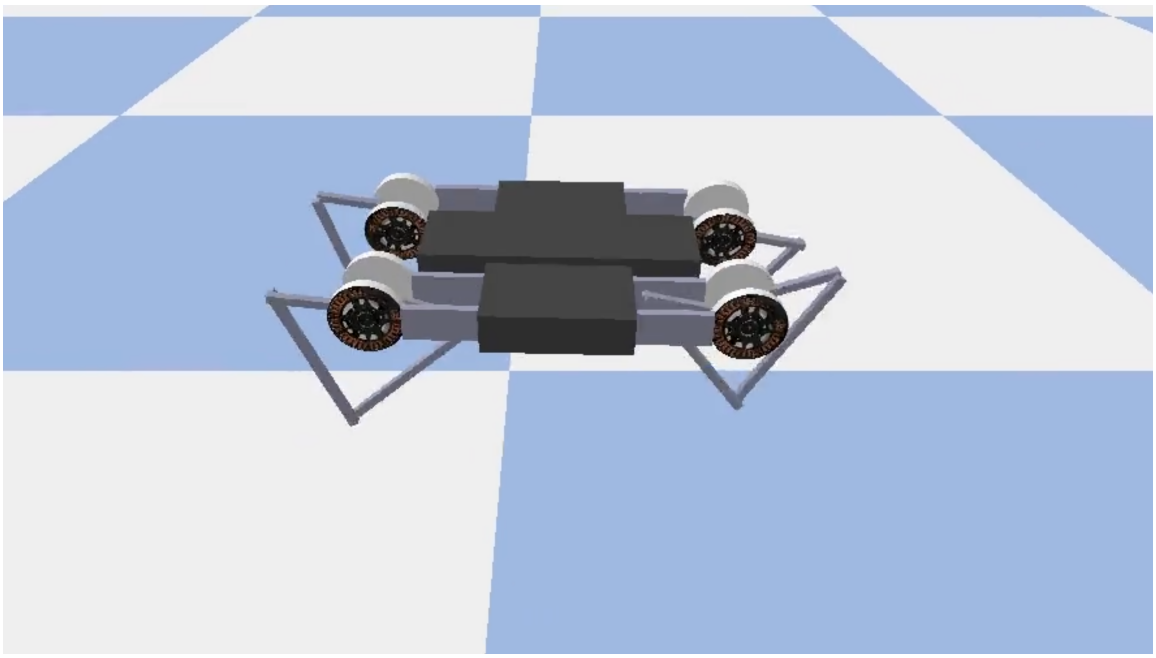


Figure 1.5: Minitaur in PyBullet

PyBullet is an open-source python wrapper for the Bullet C-API and is fast and easy to use for robotics simulation [14]. In pybullet, one can simulate forward - inverse kinematics and dynamics, collision detection. Pybullet can work with many robot description formats like URDF, SDF, and MJCF. With integration from openAI gym, pybullet provides class for minitaur robot. OpenAI gym API [15] provides a way to communicate with multiple pybullet instances for the controlling algorithms. The OpenAI gym API is mainly designed for neural-network-based reinforcement learning. A typical openAI gym class contains four methods,

1. Initializer: This method initializes the environment on a separate thread for parallel simulation.

2. Step: This method takes action and simulates the system for the next time step. This method returns the next state, reward, and episode termination information.

3. Reset: This method is used to reset the robot and environment to their initial state. This method prevents re-initialization of the environment and makes it more efficient in running.

4. Reward: This is evaluation of action according to some objective function.

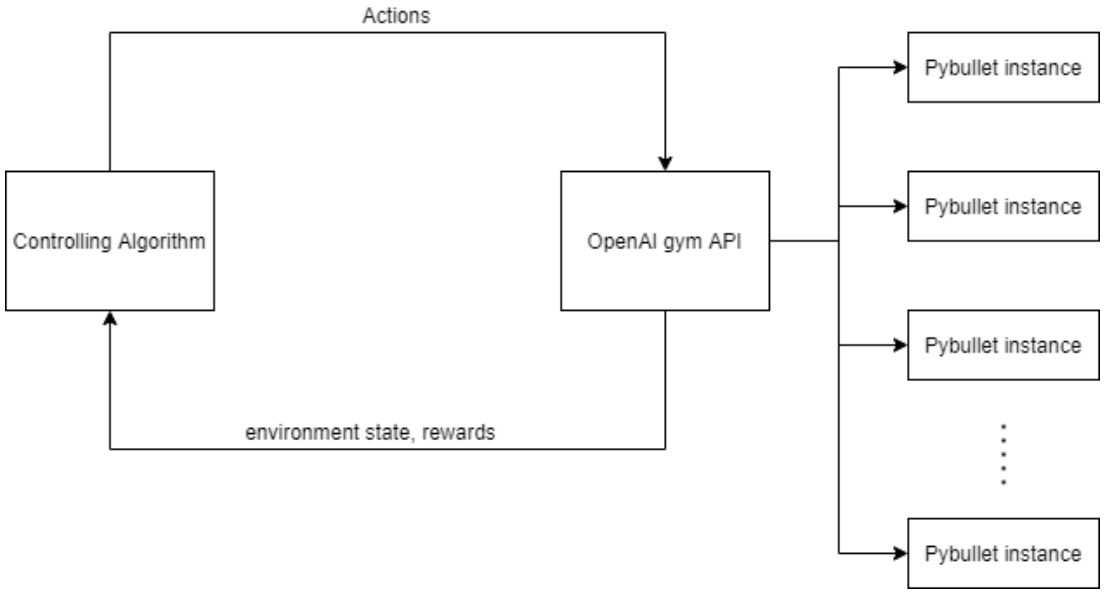


Figure 1.6: Control algorithm interaction to PyBullet through OpenAI Gym API

1.4.1 State

The state is a set of variable that can be used to desribe dynamical systems's future variables in the absence of external force. The state for the minitaur robot in pybullet environment is given by 28 dimentional vector. The state of the minitaur is given by,

1. **motor angle** : The environment returns motor angle from minitaur's local up direction.(see fig) The values are in the range $-\pi\, to\, +\pi$ radian for each motor

2. **motor velocity** : The environment returns motor velocity The values are in range of $-167.25\, to\, 167.25$ rad/s for each motor

3. **motor torque** : The instantaneous torque returned by the environment is from $-5.7\, to\, +5.7$ for each motor

4. **minitaur orientation**: Environment calculates the instantaneous orientation in quaternion with respect to world coordinate system. Total 4 values.

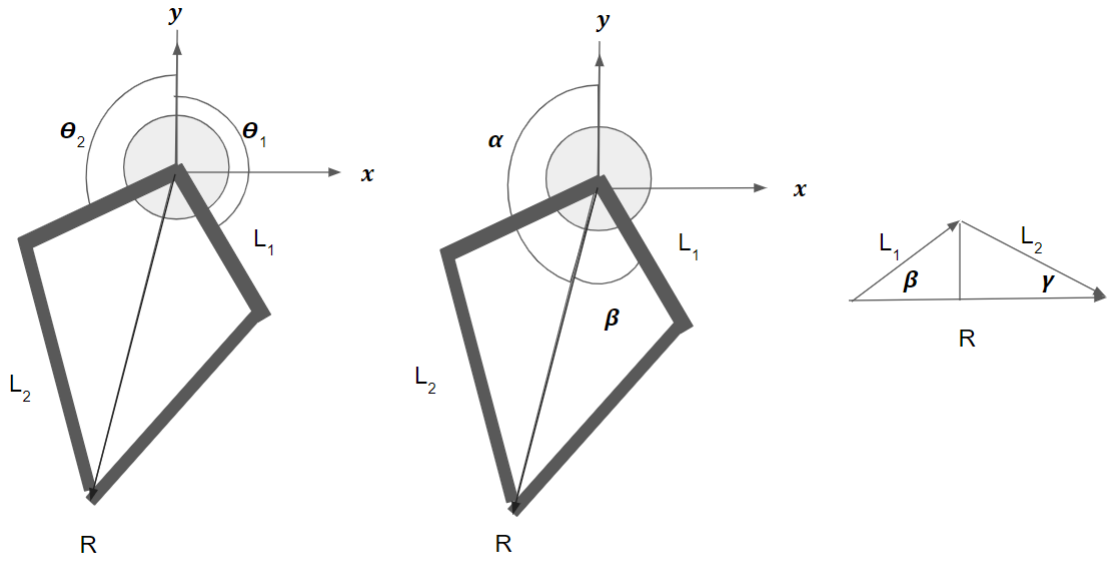


Figure 1.7: Leg model for Minitaur

1.4.2 Reward

The reward is a way of the environment giving feedback to the algorithm of its current performance. Reward functions are designed according to the task. The default reward function

in the minitaur is given below.

$$total\ reward\ =\ forward\ reward - drift\ reward - shake\ reward - energy\ reward,\ where \tag{1.1a}$$

$$forward\ reward = C1 * min((x_t - x_{t-1}),\ forward\ reward\ cap) \tag{1.1b}$$

$$drift\ reward = C2 * (-|y_t - y_{t-1}|) \tag{1.1c}$$

$$shake\ reward = C3 * ((1,1,0) \cdot vector\ perpendicular\ to\ minitaur\ body) \tag{1.1d}$$

$$energy\ reward = C4 * (-|\vec{\tau_{motors}} \cdot \vec{v_{motors}}| * dt) \tag{1.1e}$$

The default value of C1, C2, C3, C4 are 1.0, 0.005, 0.0, 0.0. The value of the coefficients are generally ≥ 0.0. The coefficients C1, C2, C3, C4 can be adjusted to serve different needs. For example, If the C4 value increases, the penalty of energy consumption increases. This makes algorithms make less energy-consuming steps. But increasing too far may also freeze the algorithm to do anything at all. Another way of giving feedback to the algorithm is to set a threshold. For example, The value of the forward reward cap can be used to control maximum speed for the minitaur. The speed further than the forward reward cap does not increase the forward reward, but the energy penalty directly related to the velocity of the motors increases. This discourages further speed increment. The shake reward penalizes the inclination with respect to the horizontal plane and makes the minitaur more stable.

1.4.3 Action Space

In standard practice OpenAI gym API takes normalized input between -1 to 1. The minitaur takes 8 commands normalized in -1 to 1 at each time step. For minitaur the leg made of 4 links with link lengths as shown in the figure. The pybullet API takes angle θ_1 and angle θ_2 as shown in the figure. The environment takes normalized angles α and β. The minitaur environment

change internally this α, βs to θ_1, θ_2, From fig (a,b), $\theta_1 + \beta = \alpha$ and $\theta_1 + \theta_2 + 2\beta = 2\pi$, solving for θ_1, θ_2,

$$\theta_1 = \alpha - \beta \tag{1.2a}$$

$$\theta_2 = 2\pi - (\alpha + \beta) \tag{1.2b}$$

One can derive x,y coordinates from α, β From fig(c),

$$L_1 * \sin\beta = L_2 * \sin\gamma \tag{1.3a}$$

$$\gamma = \arcsin\frac{L_1 * \sin\beta}{L_2} \tag{1.3b}$$

Now putting γ value for finding R,

$$R = L_2 * \cos\gamma + L_1 * \cos\beta \tag{1.4a}$$

$$= L_2 * \sqrt{1 - \sin^2\gamma} + L_1 * \cos\beta \tag{1.4b}$$

$$= L_2 * \sqrt{1 - (\frac{L_1}{L_2} * \sin\beta)^2} + L_1 * \cos\beta \tag{1.4c}$$

$$R = \sqrt{L_2^2 - L_1^2 * \sin^2\beta} + + L_1 * \cos\beta \tag{1.4d}$$

Now from fig(b),

$$x = R * \cos(\frac{\pi}{2} + \alpha) = -(\sqrt{L_2^2 - L_1^2 * \sin^2\beta} + + L_1 * \cos\beta) * \sin\alpha \tag{1.5a}$$

$$y = R * \sin(\frac{\pi}{2} + \alpha) = (\sqrt{L_2^2 - L_1^2 * \sin^2\beta} + + L_1 * \cos\beta) * \cos\alpha \tag{1.5b}$$

These x,y coordinates is used in ch.2 for generating leg coordinates.

1.4.4 Episode Termination

The episode termination function is called every timestep of environment execution. The termination function ends the episode and closes the environment. The criteria of termination should be changed according to the task. The termination functions can sometimes change reward to a huge negative value to give feedback to the controlling algorithm. For minitaur, the episode is terminated when one of the following criteria is made.

1. If the $z\,coordinate\,of\,the\,torso\ < 0.13$. It prevents controlling algorithm to learn gaits that can lay to low to the ground.
2. If $\hat{k} \cdot vector\,perpendicular\,to\,minitaur\,body) < 0.85$
3. If environment reaches 1000 steps

1.5 Neural Networks

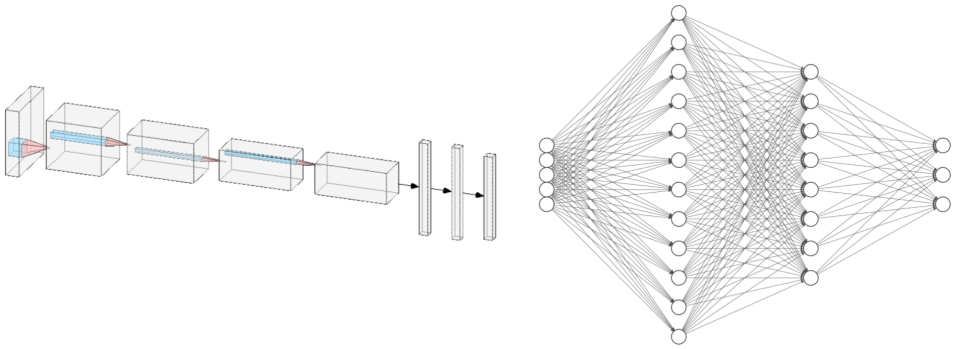


Figure 1.8: Some popular neural network architecture

In recent years, data-driven methods exploded in popularity because of the availability of large labeled datasets and affordable, fast computational hardware. The one important data-driven method is called neural networks. The neural networks are considered universal function

approximators. They can learn to express large variety of functions with proper input and loss functions. The neural network takes vector $\vec{I}$ as input and outputs the $\vec{O}$. Hidden layers connect the input and output layers. The hidden layer contains a number of neurons according to the network architecture. The input to each layer is multiplied by the weights vector w, and bias b is added. The summation is then passed through nonlinear function f. Without nonlinear function whole neural network can be expressed as single matrix multiplication. The mathematical process of neural networks can be described below.

$$\vec{O} = NN(\vec{I}, W) = h_n(h_{n-1}(...(h_0(\vec{I}))),\ where\ \ y^{k+1} = h_k(y^k) = g(\Sigma_{i=1}^{n} w_i^k * y_i^k + b^k)$$

At the beginning of training, the weights of each layer the w_i are randomly sampled according to some distribution. The training of the neural network is done by the optimization algorithm. Many optimization algorithms like stochastic gradient descent, ADAM, etc., are used to train the network. The objective function for the neural network is given by,

$$\underset{W}{\operatorname{argmin}}\, f(true\ values, NN(\vec{I}, W))$$

There are many varieties of loss functions can be according to the task and the data. The dataset is assumed to be from the same distribution and split into train and test sets. The neural network is then trained on the trainset and tested on the test set. There are numerous neural network architectures possible. In this thesis, the convolution neural network and feed-forward neural network are used.

Chapter 2

Learning to walk on rough terrain with Reinforcement Learning (RL)

2.1 Walking on uneven terrain with prespecified gaits

Generally, gaits in legged robotics gait trajectories of the legs are handcrafted and turned. This process makes walking possible for the legged robots, but there is no guarantee of optimal performance. The fine-tuned gaits works give satisfactory values for a limited amount of scenarios. For example, the gaits tuned for walking in the planer region fail to perform in the uneven terrain. The pybullet simulator provides some default gaits for the planer terrain. Later experiments found the suboptimality of the default gaits.

The default gait generated by antiphase sine waves.

$$angle\ for\ front\ motor\ = amptitude * sin(speed * t) \tag{2.1a}$$

$$angle\ for\ back\ motor\ = amptitude * sin(\pi + speed * t) \tag{2.1b}$$

The default parameters for default gaits are, amptitude = 0.5, speed = 40. This sine gait passed through the forward model (descirbed in CH.1) looks like below,

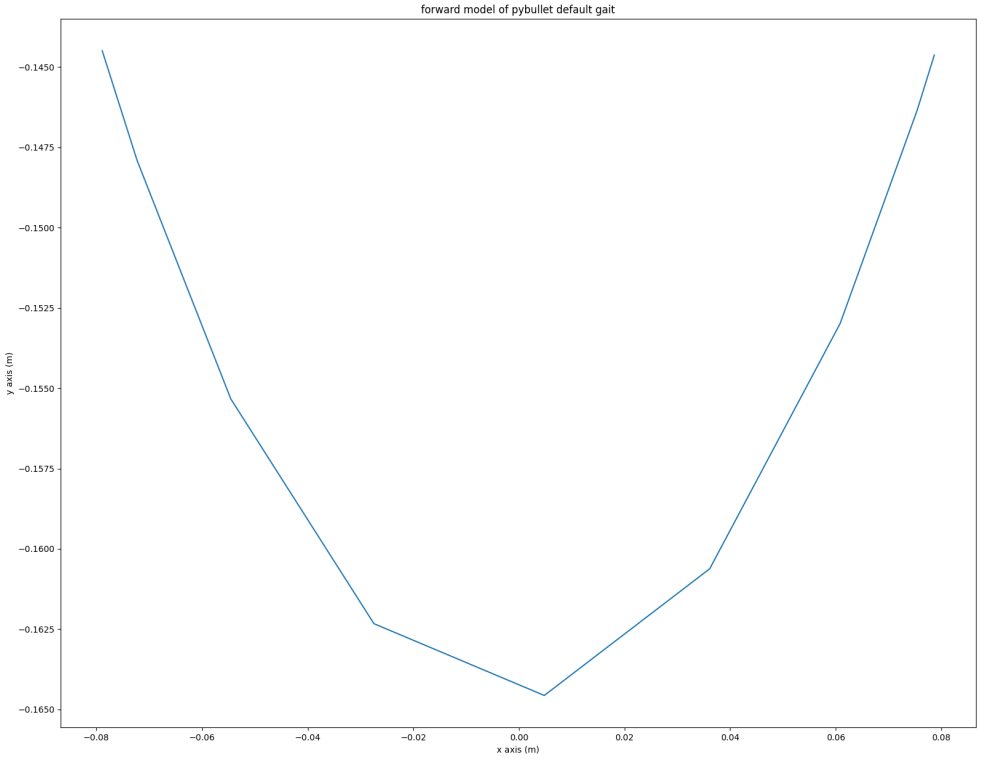


Figure 2.1: Forward model applied on the gait

The leg trajectory is limited in $\pm 8cm$ in x axis and $-14.5cm\,to\,-16.5cm$ in y axis. This gait works on planer terrain but it fails for terrain parameters [0.1, 0.1, 0.1, 0.1] (Described in later in the chapter). Because this gaits are open-loop and does not have any feedback mechanism. One episode of this gait on planer and rough terrain is visualized in figure(2.2).

The sine gait is tested for 50 episodes for rough terrain, and results are shown in figure (2.3). The first plot shows the average cumulative reward from the episode. The second and third plots in the figure indicate average x and y locations after the end of the episode. The planer

terrain in pybullet gives the same results every time, so the algorithm maintains the same behavior. So standard deviation is zero in the case of planer terrain.

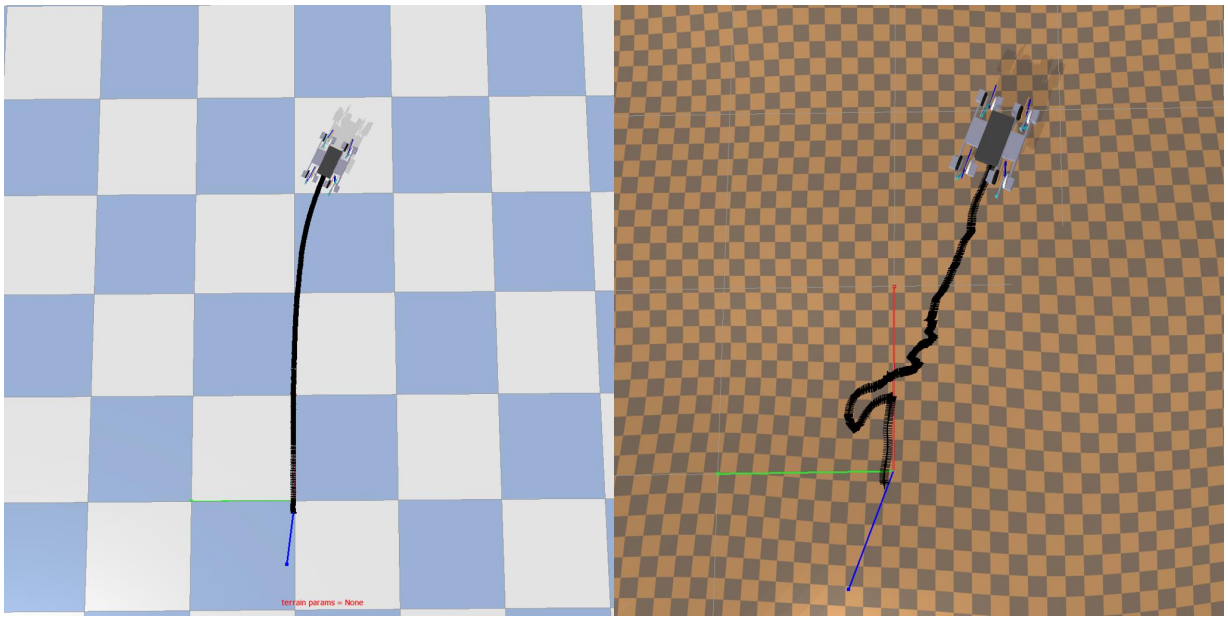


Figure 2.2: Left Sine gait on planer terrain and Right Sine gait on Rough Terrain

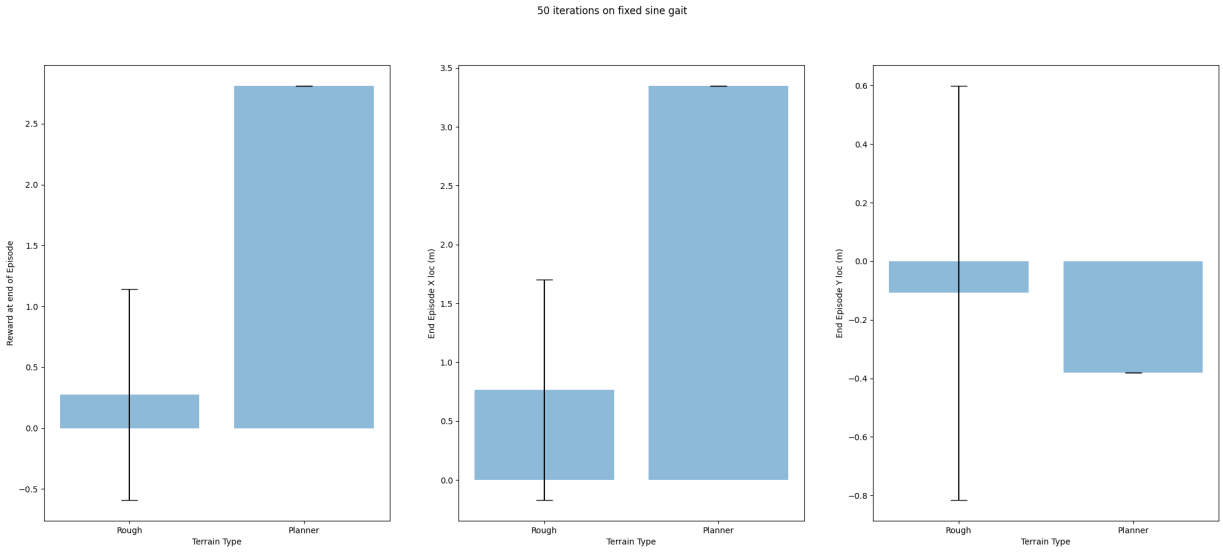


Figure 2.3: Left Cumulative Reward, Centre X location, Right Y location at end of episode for Sine gait

2.2 Introduction to RL

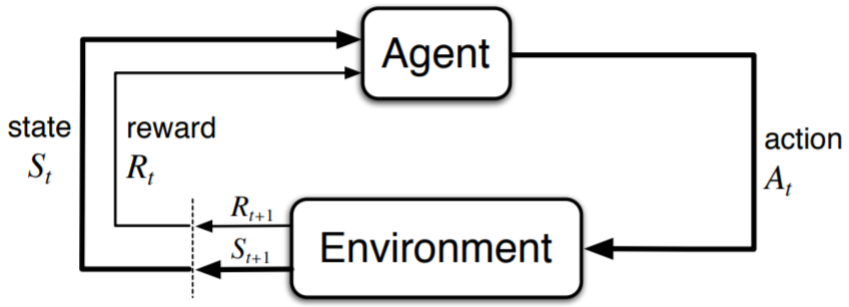


Figure 2.4: The agent-environment interaction in a Markov decision process [3]

In recent years reinforcement learning has gained the community's attention because of rapid progress in hardware acceleration and deep learning. Reinforcement learning allows the controller to learn the task optimally with trial and error. Reinforcement learning is based on Markow decision processes.

Markov decision processes, also known as MDPs, are a classical formalization of sequential decision making, where actions influence immediate rewards, but also subsequent situations, states, and future rewards as well. The learner and decision-maker is called agent, and anything outside the agent is called environment. The agent decides the actions through the state, and the environment returns the consequence of the action in terms of following state and reward. The environment state at time t is given by S_t. The agent makes decision according to function called the policy. The probability of action is defined by $\pi(a,s) = Pr(a_t = a|s_t = s)$. The next state and reward from the environment is given by S_{t+1} and $R_a(S_t, S_{t+1})$.

The reinforcement learning objective is to maximize the total reward, also known as the cumulative reward at the end of the episode. In general, both rewards and transition dynamics can

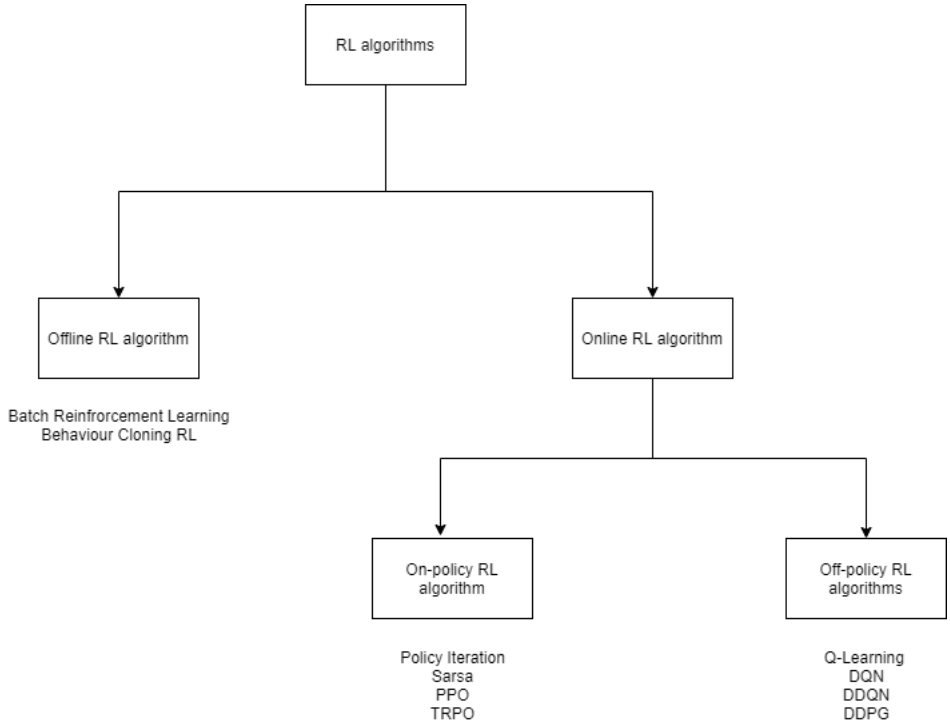


Figure 2.5: Reinforcement Algorithm Categories

be stochastic. The objective of reinforcement learning is given below,

$$\sum_{t=1}^{T} \mathbb{E}_{\tau \sim p(\tau)} = [r_t]$$

2.3 general categories RL algorithms

According to how policy is learned RL algorithm is categorized into different categories. These methods differ in how data is collected for learning the agent and how it interacts with the environment.

2.3.1 Online RL algorithm vs off-line RL algorithm

The online RL algorithms directly interact with the environment. The agent learns to maximize reward by directly interacting with the environment with actions selected by the agent's policy function. While offline reinforcement learning algorithms utilize previously collected data without additional online data collection. The agent is provided with a static dataset of fixed interactions and tasked with learning the best policy using data alone. Offline RL fundamentally supervised learning problem where data is training set.

On-policy RL algorithm vs off-policy RL algorithm

The Online RL algorithms further categorized into on-policy and off-policy algorithms. The online RL algorithm collects data using latest policy, and then using that experience to improve the policy. It is considered online interaction. While off-policy algorithm setting, the agent's experience is collected in memory buffer also known as replay buffer. The updated policy collects more data and that data is then added to the memory buffer. The policy trained then trained on the data. This setting allows use of the old data generated by the old policies. This improves sample efficiency.

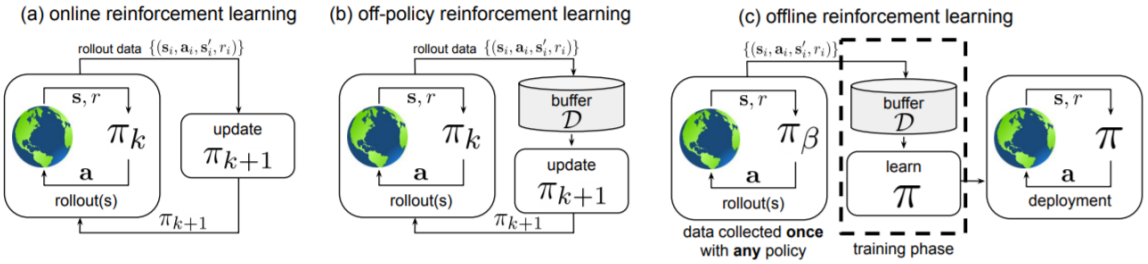


Figure 2.6: Dataflow in Reinforcement algorithm categories

2.4 Proximal Policy Optimization (PPO) algorithm

The Proximal policy optimization is online on-policy gradient method. In on policy RL, the algorithm does not have replay buffer and can only learn from data generated by current policy. Because of the stochastic nature of the environment and policy the data distribution of states and rewards generated by the policy is changes constantly. This causes instability in the learning. The ppo algorithm reduces the problem by clipping the gradients.

The agent interacts with environment with fix number of the episodes parallel running environments and then data generated by the algorithm is used for the training. The deep reinforcement learning implementation uses policy and value networks. The policy network takes in current state and outputs a action. The value network takes current state and outputs the value of that state pair.

The value network V_ϕ is parameterised by ϕ [16],

$$\phi_{k+1} = argmin_\phi \frac{1}{|D_k|T} \Sigma_{\tau \in D_k} \Sigma_{t=0}^{T} (V_\phi(S_t) - \hat{R}_t)^2$$

The actor netwokr π_ϕ is parameterised by θ [16],

$$\theta_{k+1} = argmax_\theta \frac{1}{|D_k|T} \Sigma_{\tau \in D_k} \Sigma_{t=0}^{T} min(\frac{\pi_{theta}(a_t|s_t)}{\pi_{theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t,a_t), g(\varepsilon, A^{\pi_{\theta_k}}(s_t,a_t)))$$

where D_k is dataset of trajectories, $\hat{R}_t$ is reward to go, A is advantage is normally given blow,

$$A^{\pi_\theta}(s_t,a_t) = Q^{\pi_\theta}(s_t,a_t) - V^{\pi}(s_t)$$

2.5 Parameterized Environment Generation

Current implementations of RL environments consist of the same environment without any perturbations. The algorithm can learn to solve that particular environment and can have high performance on it. This creates a problem as policy overfits some environments and fails to generalize. In the case of minitaur, the environment can be changed by controlling terrain parameters. As minitaur in pybullet is modeled after real minitaur, it does have a limitation on which terrain parameters it can walk on without fail. Parameters can include terrain slope, terrain friction, the distance between nearest hilltop, etc.

2.5.1 Perlin Noise

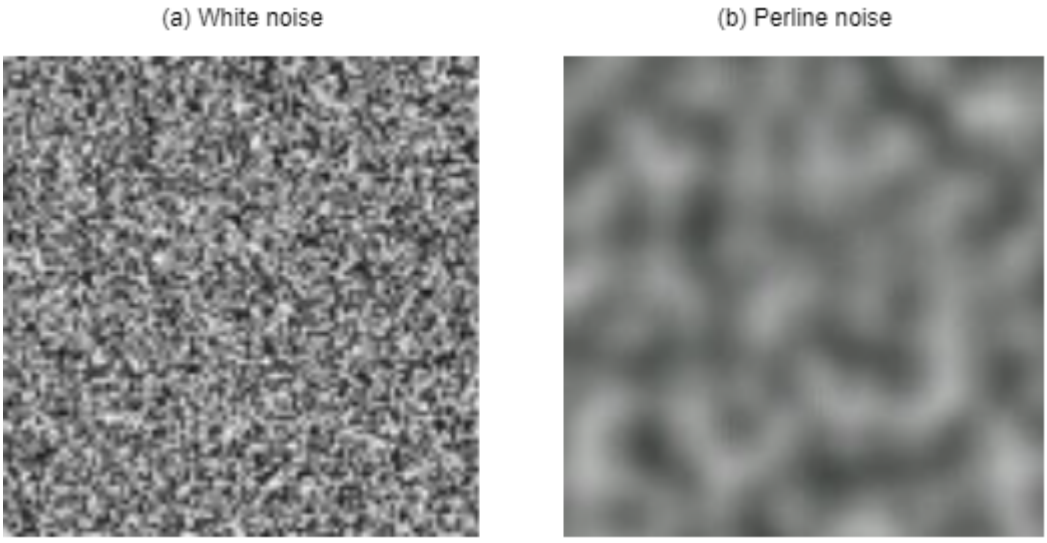


Figure 2.7: Different type of Noise

In the area of signal processing the noise is considered to be random value that can fluctuate within maximum and minimum value. The terrain generated according to totally random values is not useful because the height changes in neighbouring location is not realistic and minitaur leg can jam in the terrain bumps. The more realistic idea is to generate the terrain

according to random values that are somewhat correlated with neighbouring values, to ensure more realistic and smoother values. One popular idea from the computer graphics is to used perlin noise [17]. The perlin noise uses octaves to generate the pseudo-random numbers. For fast implementation it is advised to use perlin-numpy library [18].

2.5.2 PyBullet Environment Generation

The terrain generation function takes terrain parameters like size, list of resolution of the octaves, and list of noise coefficients. The function generated 2D NumPy array was then passed to the pybullet API for terrain generation. The arrays are generated by the size and resolution of octaves, and the noise coefficients premultiply them for the appropriate size. This approach differs from the fractal terrain generation as it allows the individual frequency of Perlin noise to be controlled.

$$height(x,y) = \Sigma_{i=1}^{4} NC_i * perlin(x,y,\ resolution_i)$$

The parameterized environment generation can be used in the curriculum of training of the agent [19]. One proposed way of generating a curriculum for agent training is described below.

The parameters are sampled uniformly in the parameter space. The parameter space is a domain of the parameter that is allowed in the current cycle of the environment generation. The optimization function of the curriculum generation is the reward function of the environment. The reward function depends directly on the current policy and the current spread of the parameters. The environment generally is stochastic and can give different end of episode total rewards for the same parameters. It is required to frequently sample the environment for given parameters and use the mean of the total end of episode rewards. The parameter space is defined by the mean and endpoints for a given parameter.

Algorithm 1 Environment Parameter Adaptive Curriculum (EPAC)

Initialize with mid point and corner points for each parameter
while $\Delta Reward \leq Reward\ Change\ Threshold$ **do**
$Calculate\ numerical\ Gradient\ of\ mid\ and\ Corner\ Points$
$Calculate\ Normalized\ Direction\ vector\ from\ mid\ point\ to\ Corner\ Point$
$Repulsive\ Force = Repulsion\ Control\ Force\ Constant * Normalized\ Direction\ Vector$
$\Delta Corner\ Point = \frac{Speed\ Control\ Const.*(Direction\ Vector + Cornet\ Point\ Corner)}{2} + Repulsive\ Force$
$Corner\ Point = Corner\ Point\ +\ clip(\Delta Corner\ Point, \pm Displacement\ Control\ Const.)$
$Corner\ Centre = average(Corner\ Points)$
$Mid\ Point = \frac{Mid\ Point + Speed\ Control\ Const.*Mid\ Point\ Gradient + Corner\ Centre}{2}$
end while

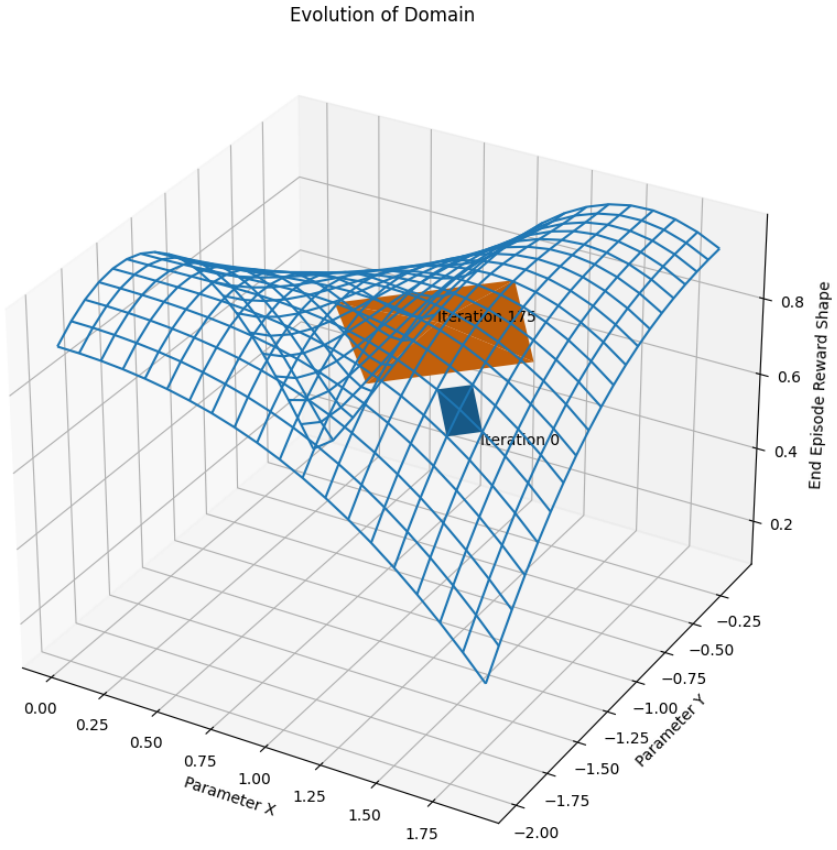


Figure 2.8: Evolution of Domain for Hypothetical End reward Function from start (iteration 0) and to iteration 175

The Environment Parameter Adaptive Curriculum algorithm is gradient based algorithm

that optimizes the parameter domain according to reward function slope. The algorithm expands and shrinks the domain according to current slope. One such example is shown in fig.(2.8). The EPAC algorithm expands the domain if the slope is low but clips it when the domain becomes to large.

Because of random inclination at origin, the minitaur initialization can be stuck inside the terrain. To prevent this normal vector for terrain at origin is using pybullet's rayTest command and minitaur at initialized according to the normal at the origin. One example of such generation is given below,

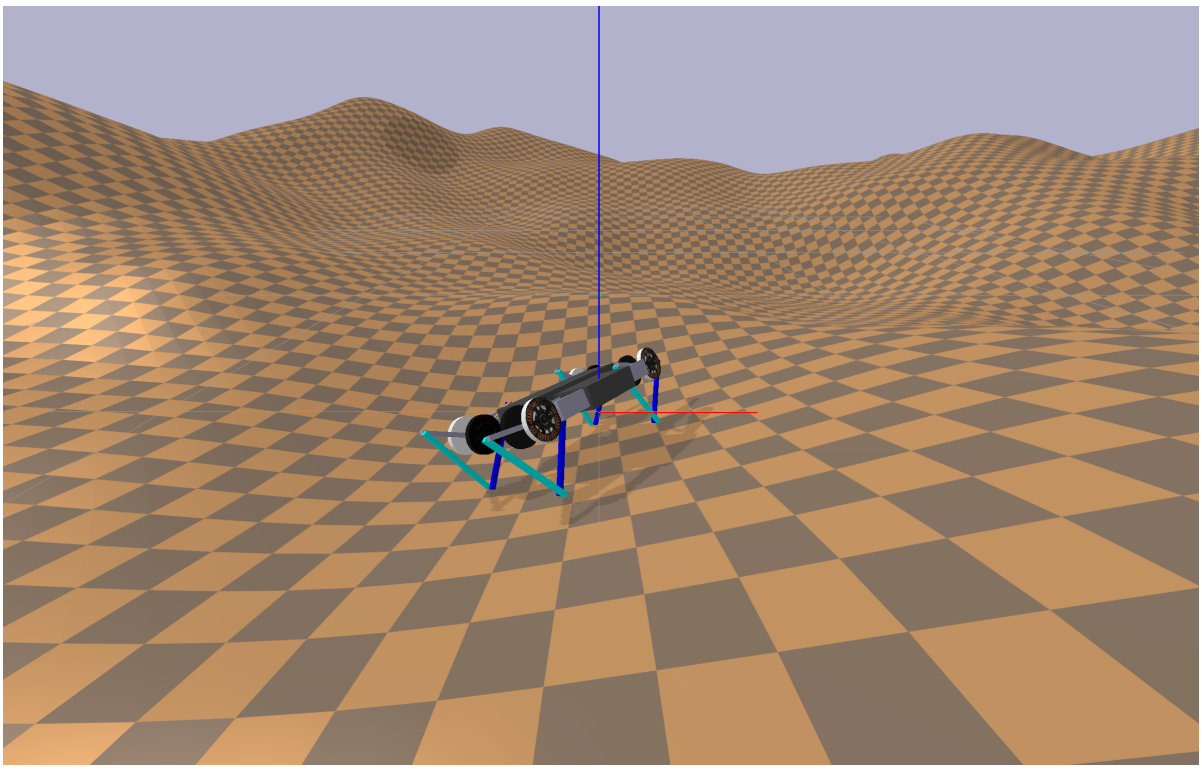


Figure 2.9: Minitaur environment with parameterized perlin noise generation

2.6 Experiment

2.6.1 Experiment Environment

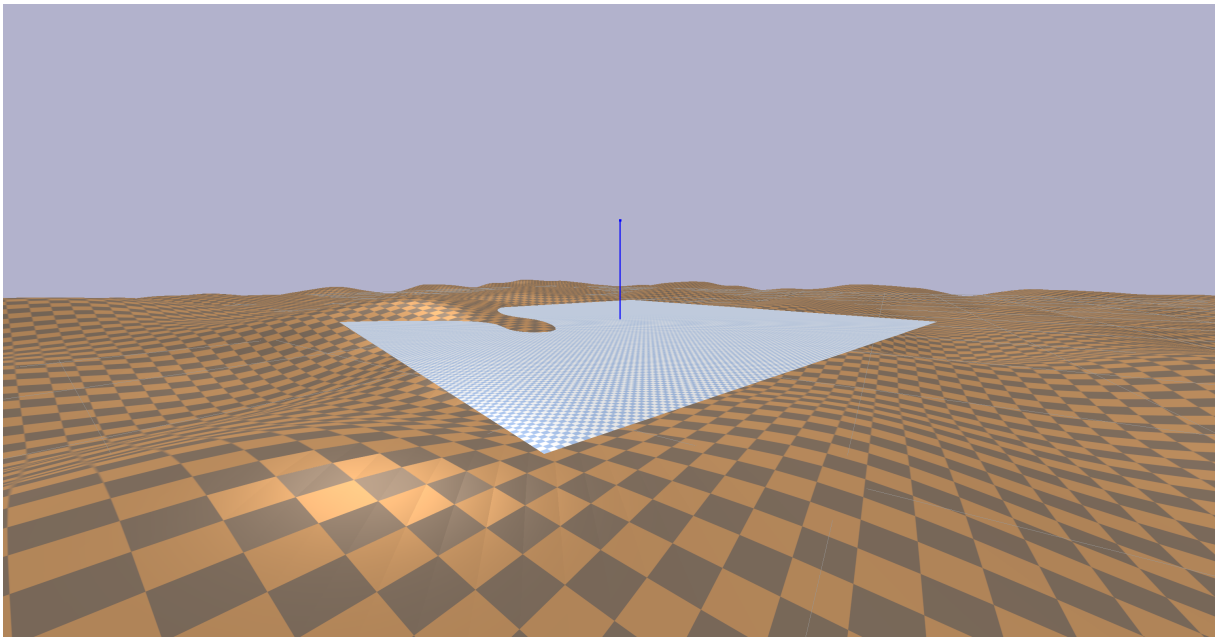
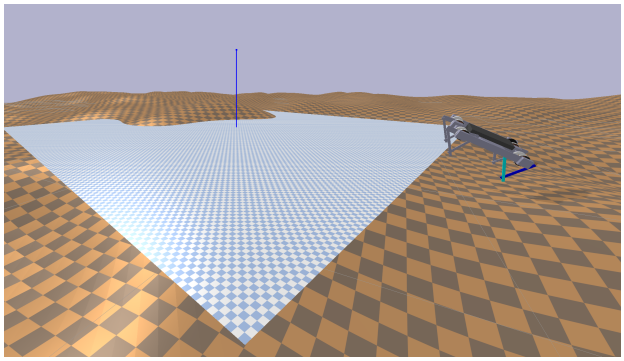
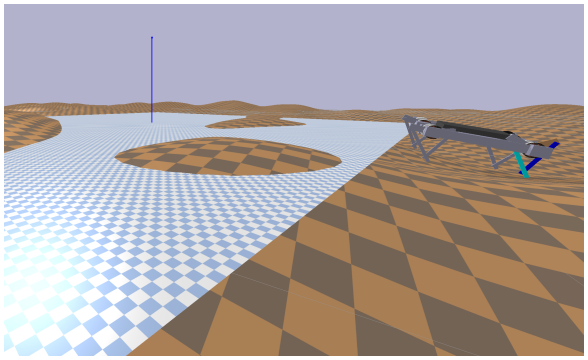


Figure 2.10: Randomly Generated Environment for Training



(a) Example of Big Gap between Plane and Rough terrain



(b) Example of Small Gap between Plane and Rough terrain

Figure 2.11: Examples of Gap between Plane and Rough terrain

The environment is generated at each reset is shown above. The plane is placed with the perlin terrain to help stabilize the minitaur in the beginning of the training. As the perlin terrain is

randomly generated the edge between the plane and perlin terrain have variable height. Some of the example of the gap shown in figure (2.11). The RL agent tries to learn falling recovery from falling. The termination criteria is changed to be more appropriate according to the task in hand. Walking on uneven terrain demands minitaur's torso to make larger angle from the z axis. Allowing larger angle with z axis also helps in fall recovery at the edge and does not terminate the episode.

New episode termination criteira is given below,

1. If $(\hat{k} \cdot vector\ perpendicular\ to\ minitaur\ body) < 0.5$
2. If environment reaches 1000 steps

2.6.2 Algorithm Details

The algorithm implemetation is taken from [4] [20]. The parameters of implementation with appropriate changes is given below, The implementation takes 2.5 days to complete 30M steps. The training of neural networks is several order magnitude is faster than data gathering (simulation running).

Table 2.1: Network quantity for the experiments [4]

Network Quantity	Policy Network	Value Network
input dimension	28	28
first layer	200	200
activation function	ReLU	ReLU
second layer	100	100
activation function	ReLU	ReLU
output dimensions	8 dim for mean 8 dim total 16 dim for log std	1 dim

Table 2.2: Parameters for the experiments [4]

Parameter	Parameter Value
number of running in parallel agents	30
evaluation episodes	30
policy updating after	30 episodes
initial mean factor	0.1
initial log standard deviation	-1
discount factor	0.995
initial kl penalty	1
target kl divergence	1e-2
optimizer name	Adam optimizer [21]
optimizer learning rate	1e-4
kl cutoff factor	2
kl cutoff coefficient	1000
total number of environment steps	3e7
maximum episode length	1000
perlin noise parameters	0.1, 0.1, 0.1, 0.1
perlin octave resolutions	2, 4, 6, 8
perlin terrain size	256 x 256

The neural network architecture is shown above. In training time, the policy network outputs action mean and standard deviation. During training, action is then sampled from mean and standard deviation. During the testing time, the only mean part of the neural network is used. The output of the policy network is passed through the Tanh function to bound between -1 to 1. The value network does not have any activation at the output. The activation function used in networks according to input value is shown below,

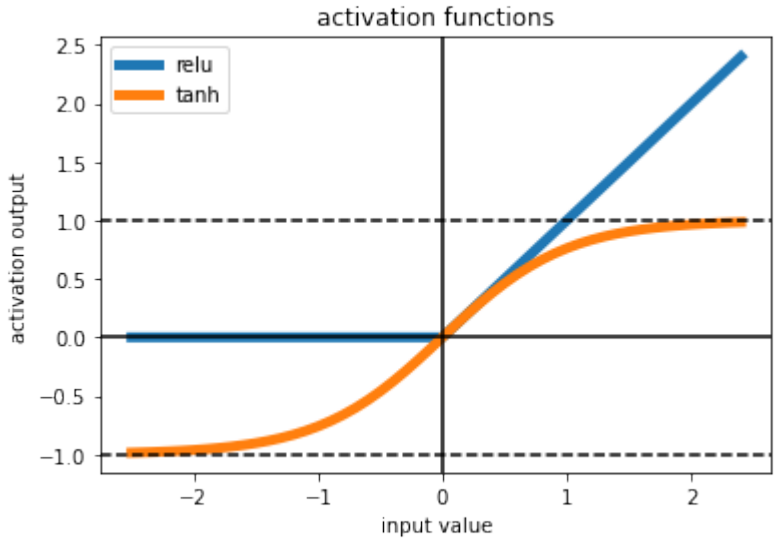
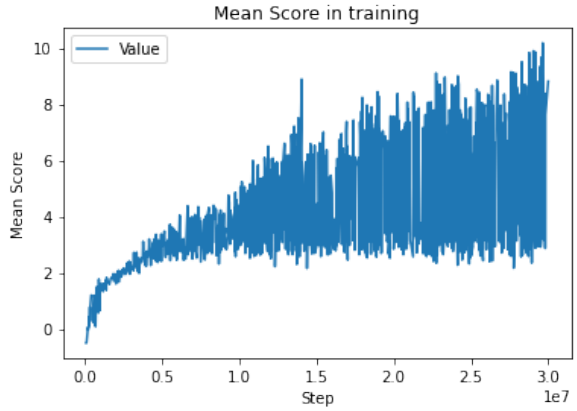


Figure 2.12: Different Activation Functions

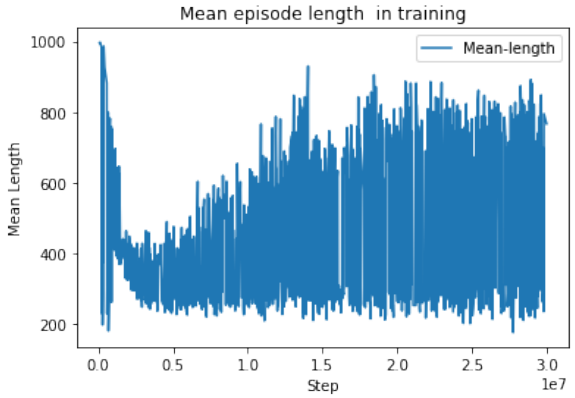
2.6.3 Training Results and Discussion

The training curve for RL trained on the random environment is shown below. The first curve is the mean score in the episode, and the second is the mean episode length. In the beginning, the policy network puts random actions. The mean score is low, and episode length fluctuates between max episode length and lowest episode length. As more training is done, the mean value of reward increases, and episodes length fluctuates less.

The minitaur starts to go towards the edge of the plane and falls down on the ground. The policy then tries to learn two things first walking on totally rough ground and do not crash at the edge of the plane. The recovery from the fall behaviour emerges. The minitaur can not take big falls many times. This can seen by the lower bound on both plots after 5M steps. The mean score is approximately lower bounded by 2.2 and mean length is lower bounded around 220 steps. The minitaur learns to survive more on the fall and upper bound on the mean score and mean length increases.



(a) Mean score in training for random environment RL



(b) Mean episode in training for random environment RL

Figure 2.13: Training Plots

2.7 With and without environment Randomization for the RL training

One of the main questions asked is does randomization of the environment is really necessary. For that one another agent is trained on the fixed environment. The test environment is 25m long in x and y direction. The comparison of such runs are discussed in figure (2.14),

The fixed environment RL agent was 50 times tested on randomly generated environments. As one can see, the fixed environment RL outperforms the sine gait. It gains a more cumulative reward and larger average X dimension at the end of the episode than sine gait. But the performance is not consistent, and the standard deviation is very large on all the criteria. The trajectory minitaur takes during the test for both planer, and rough terrain is shown in figures (2.15) and (2.16). Most failures were seen at the beginning of the episode. The agent on random rough terrain does not reach the end of the environment, and the path is nonsmooth.

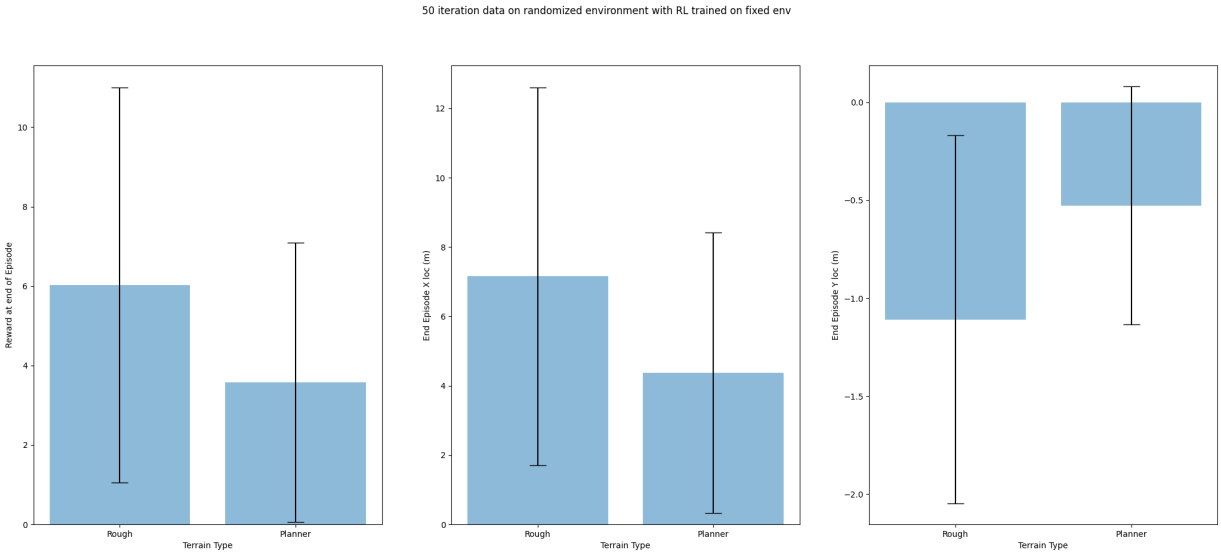


Figure 2.14: from left Cumulative Reward, Centre X location, Right Y location at end of episode for RL trained on fixed rough terrain

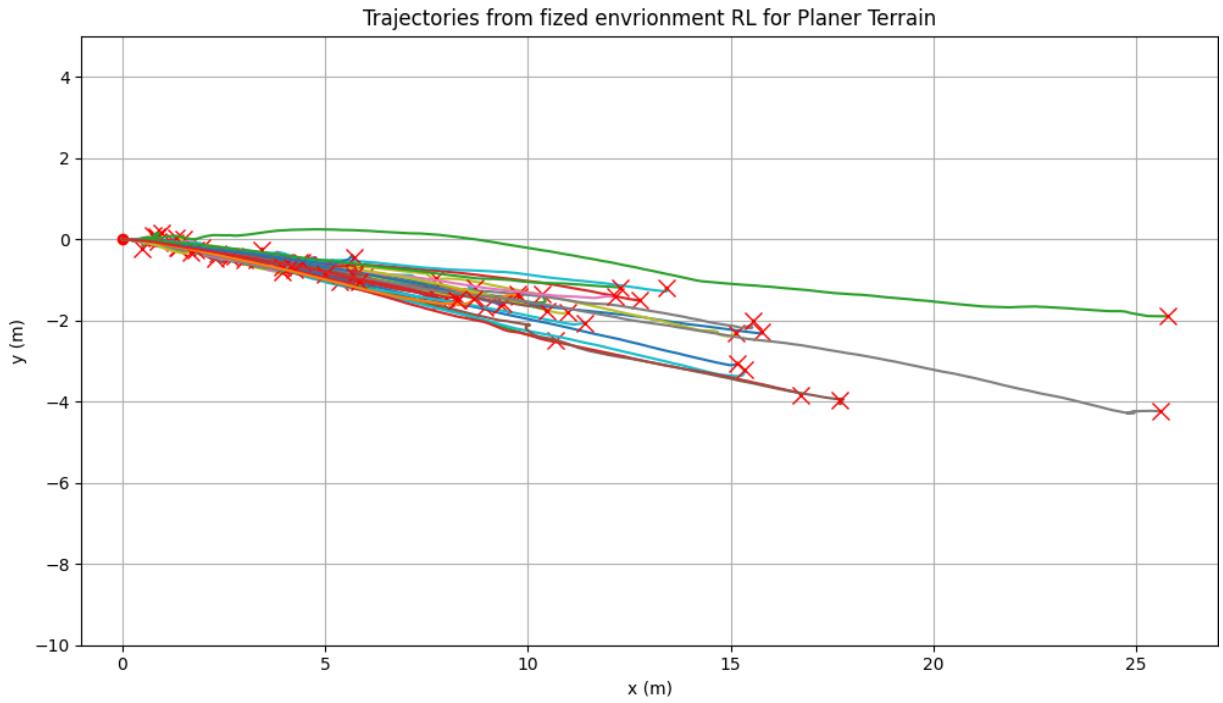


Figure 2.15: Trajectories from Fixed Environment RL for planer Terrain

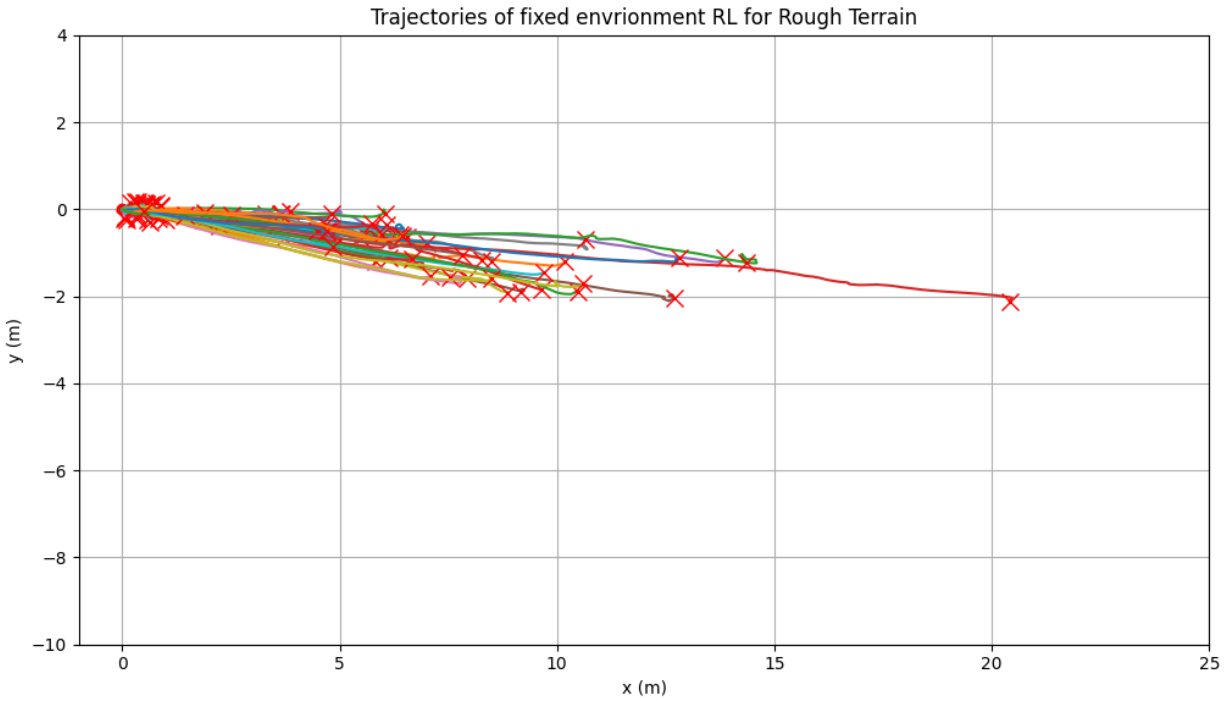


Figure 2.16: Trajectories from Fixed Environment RL for Rough Terrain

The random environment RL agent was tested 50 times. The random environment agent outperforms all the other agents by a significant margin. The failure rate was very lower, and the robot could reach the end of the environment for more than 50% of the time. As expected, the RL agent tested on planer terrain achieves a higher mean cumulative reward and higher X location at the end of the episode than rough terrain, but the difference is rather small. The trajectory minitaur takes during the test for both planer, and rough terrain are shown in figures (2.18) and (2.19). The failures were marginally because of large slopes randomly generated by the terrain generation function. The trajectories are smoother than the fixed environment RL agents.

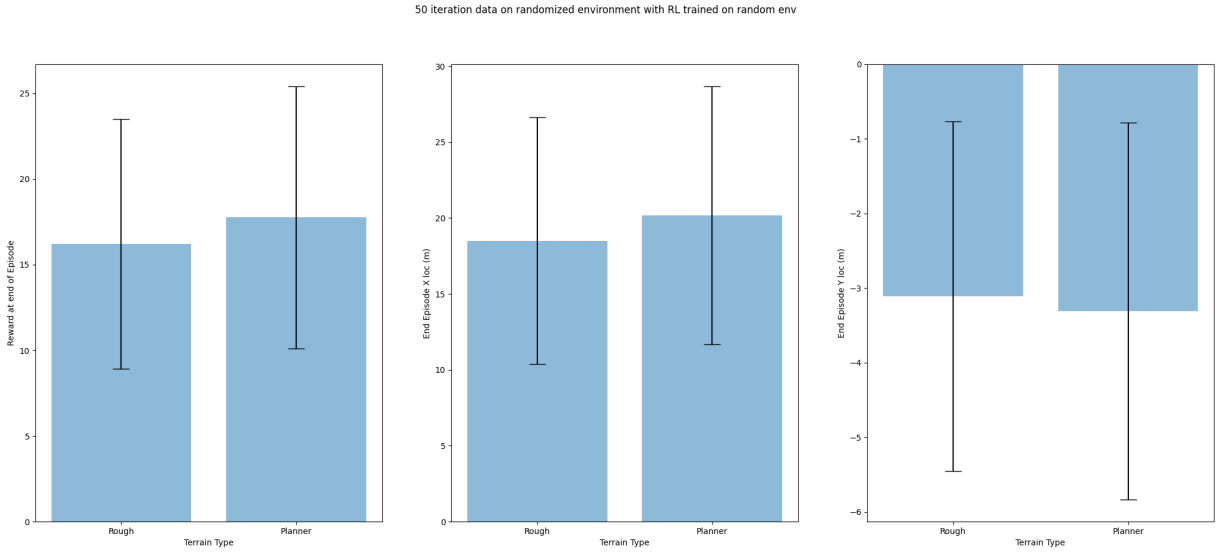


Figure 2.17: from left Cumulative Reward, Centre X location, Right Y location at end of episode for RL trained on random rough terrain

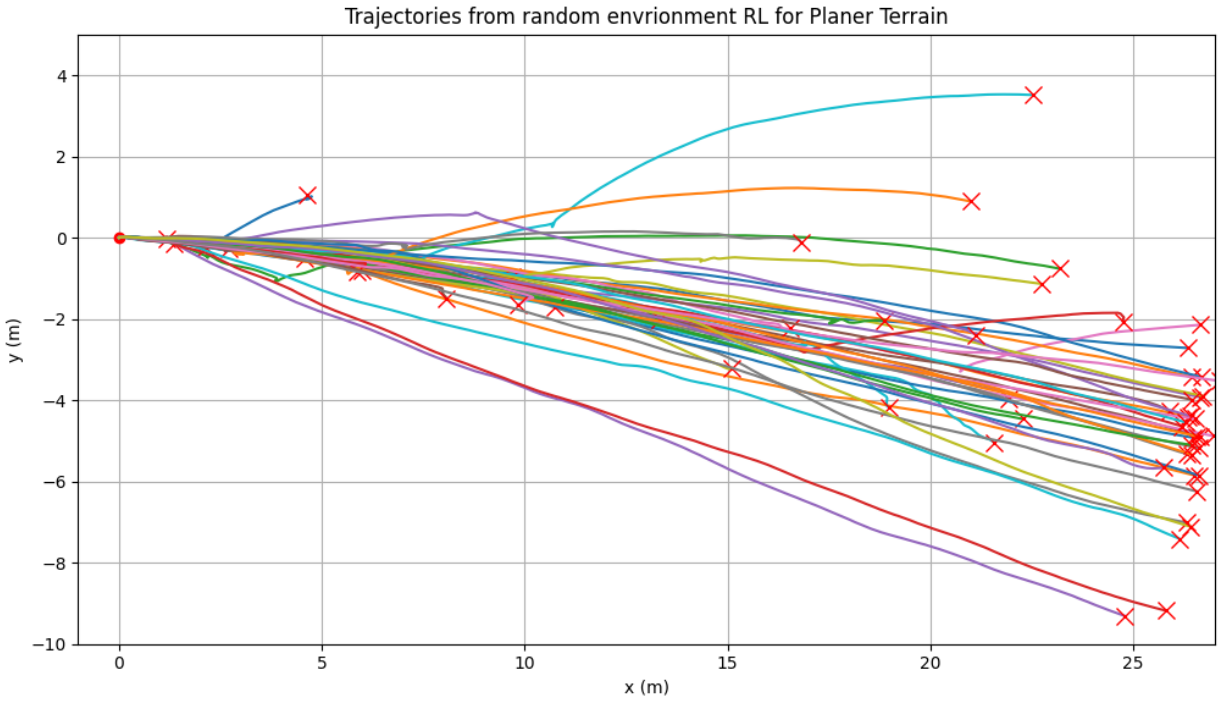


Figure 2.18: Trajectories from Random Environment RL for planer Terrain

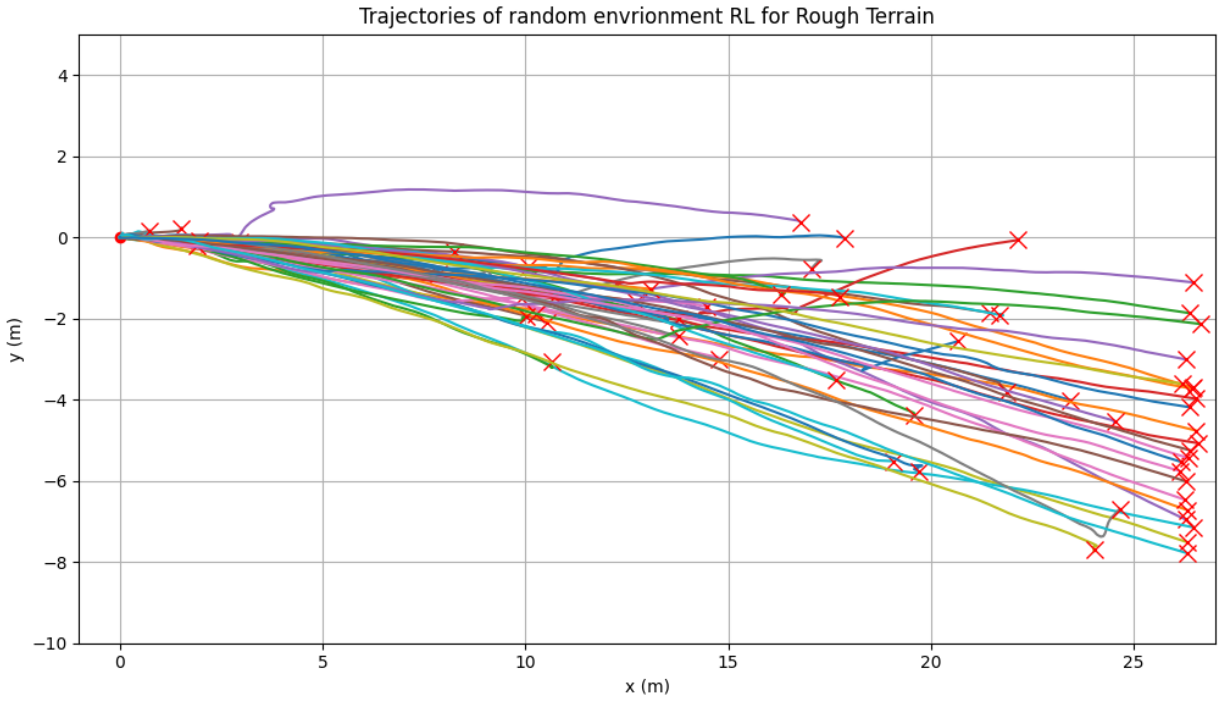


Figure 2.19: Trajectories of Random Environment RL for Rough Terrain

Chapter 3

Collision Detection and Localization

3.1 Introduction

In this chapter, leg collision is defined as any impact between a robot leg and the environment other than on the toe. The chapter explores the motivation, feasibility of doing such analysis with respect to torque drive types. The chapter then discusses the neural network-based data-driven approach with a novel loss function.

3.2 Motivation for proprioceptive methods for the collision detection and localization

Many advanced robots like Boston Dynamics's spot and ANYbotics robotics's ANYmal robot can be equipped with perception systems like cameras and a lidar. These systems gives mapping of the environment to the robots. Though these perception and mapping systems have certain limitations.They are also not perfect and can give a noisy estimation of the terrain. They can be very inefficient in tall grass environments where these sensors cannot map the ground. Because of this, it is unavoidable not to hit the leg other than the toe. The leg collision analysis

is vital because the robot can fail in the mission, can damage itself, payload, or the human. The leg collision is an important failure mode, which should be used to do mitigation analysis. These points make the collision sensing by way of tactile or other proprioceptive methods more necessary.

3.3 Feasibility of sensor less leg collision analysis for Minitaur

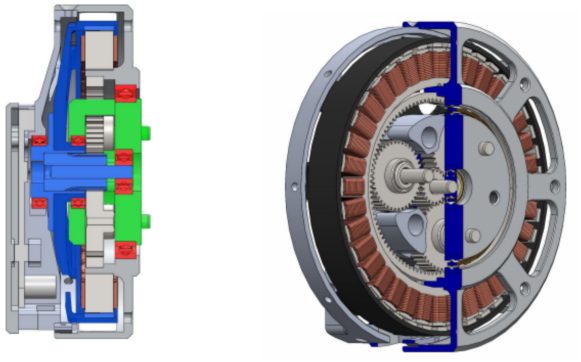


Figure 3.1: Geared drive in Minicheetah

Figure 3.2: Direct drive in hybrid-bipedal robot from MAE 207

Most robots in the modern world use some gear-down mechanism to produce high torques while operating on an optimal power band. This approach is hard in the context of collision detection and localization because the correlations between motor encoder values and collision information are diluted. The collision is spread across a large motor turn angle, and the addition of noise and backlash makes it hard to detect and localize. This is the reason that geared drive can not be generally used for sensor-less contact analysis. But minitaur does not use the geared-down approach. It uses direct drive. The direct drive is a torque transmitting mechanism that does not use any gears or belts. This method is has been relatively simple and intuitive but did not gain popularity because of the unavailability of motors that can produce sufficient torques. Minitaur is equipped with T-Motor U8, which provides enough torque. The direct drive can provide a good correlation between motor reading and collision. Many simple physics-based approaches like finding a discrepancy between the motor command and motor positions/torques can help to detect collisions [22]. The constraints of the mechanism and motor torques can be used to predict the location of the contact [23]. So, in essence, these type of methods requires two steps, namely:

1. Collision detection with discrepancy analysis
2. Collision localization with physics-based methods

These steps use the motor commands and motor readings (e.g., torque, velocity, etc.) for the prediction of collision detection and localization. It shows that the signal for collision analysis lies in the motor commands and motor readings. So our data-driven method should also use these readings for prediction. Due to the ability to work with high-dimensional input-output data, the neural network approach is selected.

3.4 CDLnet (Collision Detection and Localization Network)

The neural network architecture generally depends on the nature of data, data dimension, required output nature, and dimensionality. To our best knowledge, The neural-network based

approach for both collision detection and localization is never examined before. The various aspects of a neural network like input, output, and loss function is discussed below.

3.4.1 Input to the Network

In recent years many physics-based approaches which been based on forces, torques, and velocity constraints. (add the citations here). The underlying process is still a time series analysis. For example, the torque-based method runs online. The torque measurements are then kept track. The sudden change of the torque values is an indicator of the collision. The sudden change causes because the point of collision restricts the further movement of the leg. The error between commanded trajectory and real leg location increases; therefore, current increases, and hence torque generally increases. Then, this information can be further processed to get the location of the collision. The time scale of collision is much smaller than the time scale of the gait. As we are interested in a more recent collision, the values beyond a certain point in past history are not helpful. From that, one can conclude that the data-driven method should be given the past n state-action pairs and current state to predict the collision-related information. In our case, the algorithm should take the entire 28-dimensional state and 8-dimensional action state for n past time step and current state as input.

3.4.2 Output from the Network and Loss Function

For our problem, One naive way to approach this problem is to use a single network and predict unattainable coordinates of XY location when contact does not happen and train the network against that. This approach does not work in reality because there is an imbalance of classes of training data. The data has significantly more number of data points when the collision does not happen. Because of this statistical imbalance, the network will learn only to predict those unattainable values and will predict with a very large error when the collision occurs. The

other approach is to create two neural networks, one for collision detection and one for collision localization. At first, the collision detection network is trained, and then the collision localization network is only trained on data where the first network finds collision. In principle, this approach can work and be very accurate but is a lot slower and consumes more memory and energy than using a single network. We propose a neural network-based method that combines the speed of a single neural network and the accuracy of a double neural network by using a unique loss function for prediction. The network is from the class of temporal convolutional neural networks. The temporal convolutional neural networks use filters. The parallel implementation of filters makes the convolutional network fast. The other advantage is the number of tunable parameters is way lower than the fully connected networks [24]. In temporal convolutional neural networks (TCN), the filters (sliding windows) are applied across the time domain to extract useful features for prediction. The first layer of the network is 1D - convolution, and then the 2D convolution filters are applied. The 1D convolution extracts the features from time step t, and then the 2D convolutions are applied to that features. The 2D convolution learns the features across time to predict the required quantity. The other architectures like the fully connected neural network do not have any notion for feature extraction in the time-domain, and architectures like the recurrent neural network suffer from slow training and execution time. The output from convolutional layers is then combined with current state values and passed through a single layer network. The current time state is considered because collision can just occur or about to start occurring at the current timestep. []

The output of each layer is then passed through ReLU non-linear activation function. The neural network is essentially a BlackBox algorithm and at each time returns a number of the exact number of the outputs that are programmed when initialized. The last layer activations are selected according to the prediction requirement.

Our network outputs two type of values.

1. Collision probabilities for each leg: values ranged from 0-1 with a sigmoid activation

function total of 4 values

2. XY location of the contact for each leg in leg coordinate system: 8 values with no activation function

The problem is the data has an imbalance of classes, and we wish not to train the XY location of the contact for each leg when the collision does not happen. The solution of the given problem is given by the following loss function. Total Loss is given by,

$$Total\ Loss = Binary\ Cross\ Entropy\ Loss\ +\ \mu\ *Special\ Collision\ Regression\ Loss \tag{3.1}$$

The loss function has two parts

1. Binary cross entropy loss: This loss function is used when data has binary label (0-1) for each category. It trains the network to give the probability of the collision for each leg for given time window. The binary cross entropy loss also called as log loss given below,

$$Binary\ Cross\ Entorpy\ Loss = \frac{-\Sigma_{i=1}^{N}[\Sigma_{i=1}^{L}(C_{ij} * \log(p(C_{ij})) + (1 - C_{ij}) * \log(1 - p(C_{ij}))]}{N} \tag{3.2}$$

2. Special Collision Regression Loss: The special collision regression loss is inspired by the YOLO paper [25] and does work in the following way. The network outputs the XY coordinates of collision for each leg. The loss function takes network output and true collision values and calculates squared error between two. This squared error is then premultiplied with the collision binary label. This binary label multiplication works as a gate and allows the backpropagation algorithm to flow back the gradients if and only if the collision binary label is 1.

$$Special\ Collision\ Regression\ Loss = \frac{\Sigma_{i=1}^{N}[\Sigma_{i=1}^{L}C_{ij} * ((X_{ij} - X_{ij}^{net})^2 + (Y_{ij} - Y_{ij}^{net})^2)]}{N} \tag{3.3}$$

Because of the gate mechanism, the value of special regression loss is several orders smaller than binary cross-entropy loss. The special regression loss is multiplied with tunable variable μ. The μ value is used to balance values of the binary cross-entropy loss and special regression loss. The value of μ can be determined by running training script and evaluating values.

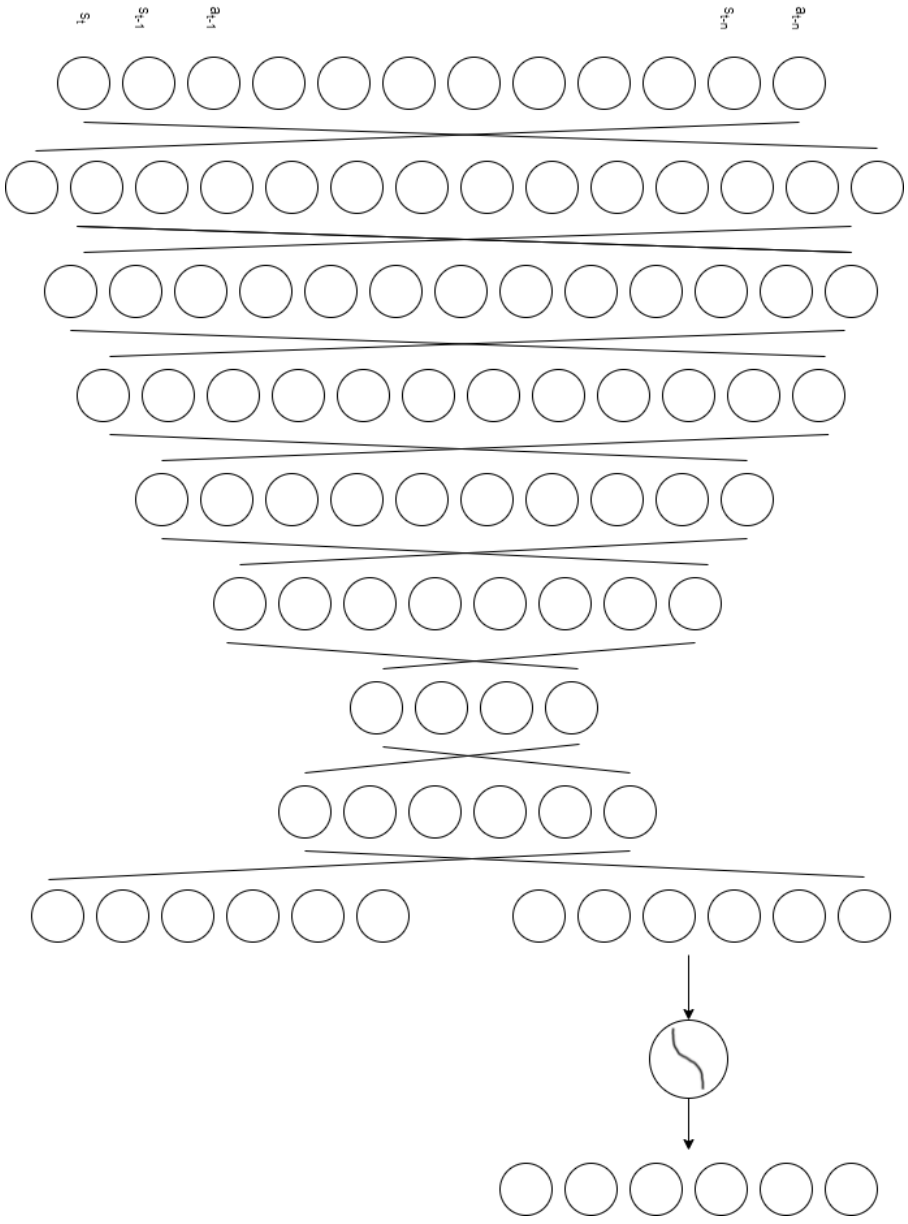


Figure 3.4: Baseline Feed-forward network

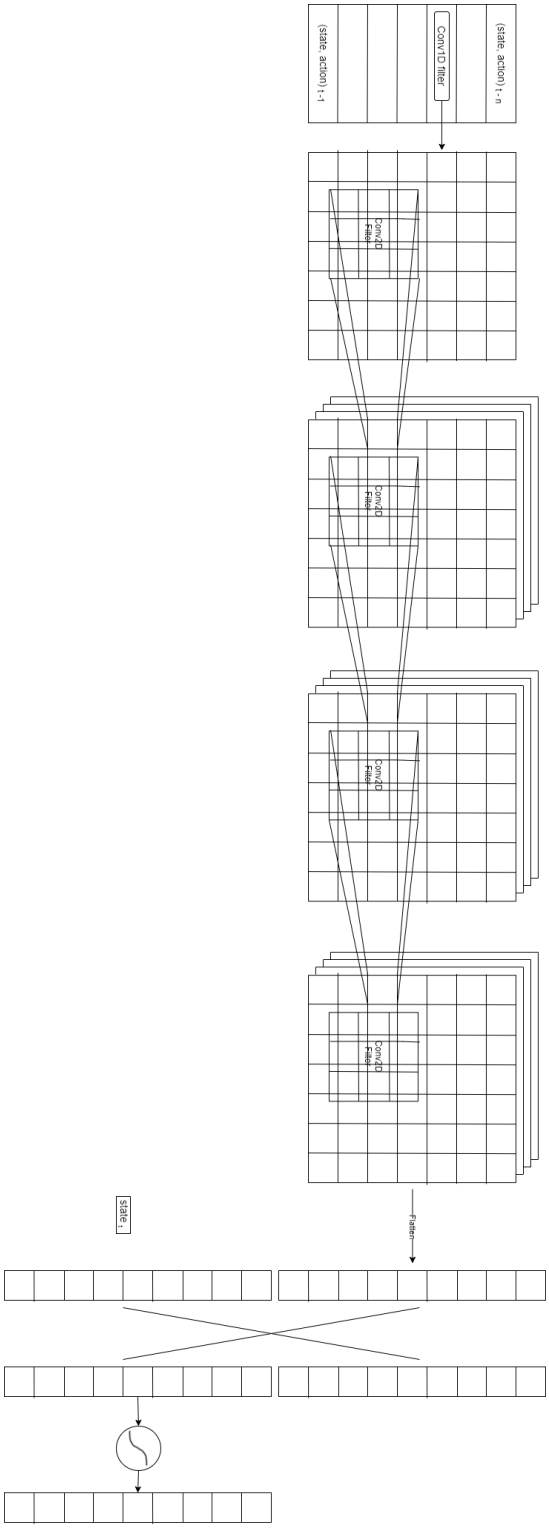


Figure 3.3: CDLnet (Collision Detection and Localization network)

3.5 Baseline

For testing the effectiveness of the CDLnet, it should be tested against the baseline. Most tabular data problem uses a fully connected neural network. There can be a huge amount of network architecture with different numbers of neurons, activation functions, and hidden layers. It is not possible to test for all of them. So for the baseline inspired by IMEDnet [26]. The baseline is trained with the same loss function with the same μ to a fair comparison.

The output of each layer is then passed through ReLU non-linear activation function. The output of the network is neural network is processed the same as described in the section before.

3.6 Experiment

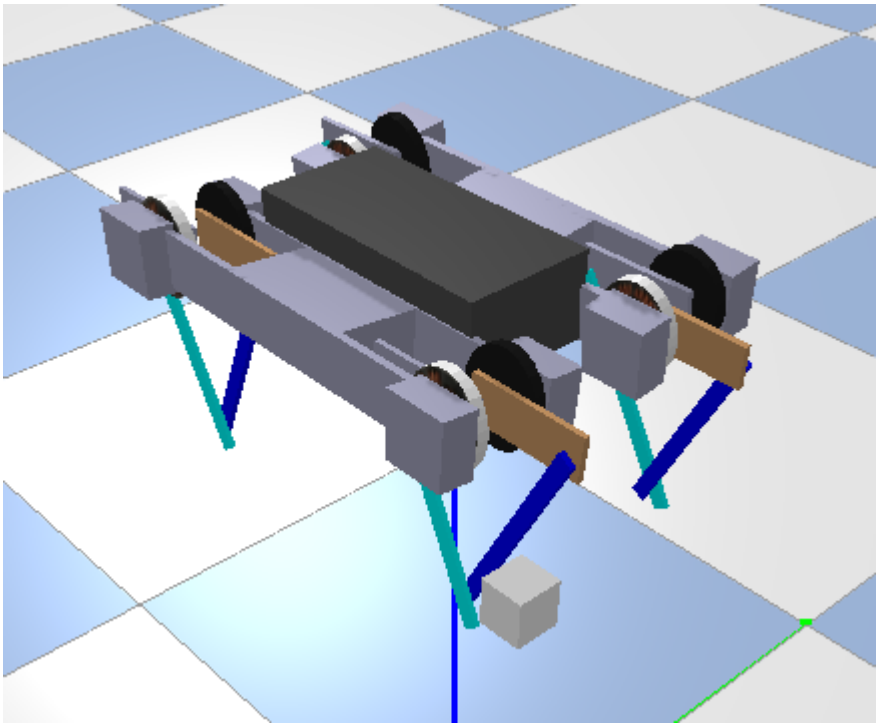


Figure 3.5: Experiment setup

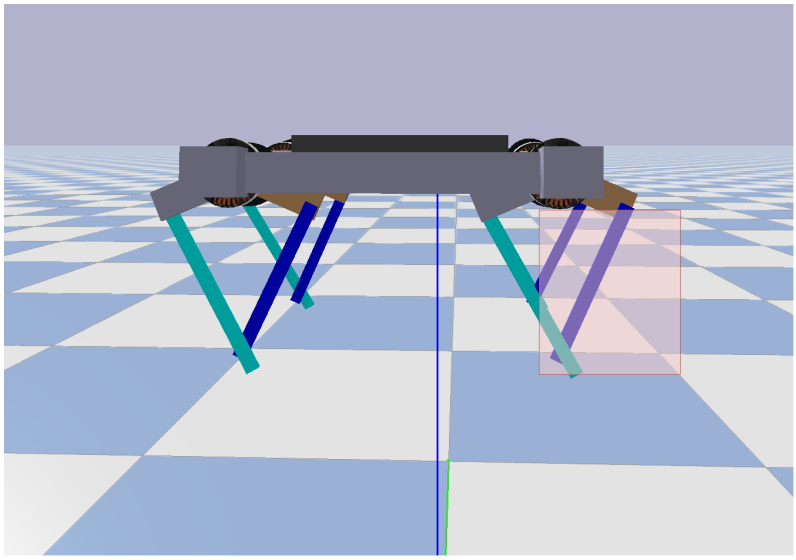


Figure 3.6: Potential area of collision

The experiment uses the simulated minitaur environment from the pybullet as described earlier. The minitaur robot's torso was put on the rack. The rack restricts the movement of the minitaur.

The script of generating such environment selects between random leg or the no obstacle at all to get as many variations in data as possible. The environment can generate random obstacles in the leg workspace, as shown in the fig below. The pybullet gives contact information by getContactPonits command. The command provides information in the world coordinate frame. Appropriate transformation in the local leg coordinate frame is made by the transform library. For data generation, The script uses sine gait from ch.2 and samples velocity in range 27-47 and amplitude in the range 0.3-0.5. The training data was generated by 250 such episodes. The saved data consist of the current observation, action, and next observation for the entire episode. The same way generates the testing data with a different seed. The training and testing data is saved in separate files and never mixed. Implementation details of network architectures are given below,

Table 3.1: Parameters for the experiments

Parameter	Baseline	CDLnet
Learning Rate	0.0001	0.0001
Max Iteration	100	100
Mini Batchsize	128	128
Optimizer	Adam optimizer [21]	Adam optimizer[21]
Input dimension	280	past 7 (s, a) and current s
Output dimension	12	12
μ	100	100

Table 3.2: the Baseline Architecture

Layer and Parameters	Baseline
Linear Layer 1	input dim = 280, output dim = 750
Linear Layer 2	input dim = 750, output dim = 650
Linear Layer 3	input dim = 650, output dim = 500
Linear Layer 4	input dim = 500, output dim = 300
Linear Layer 5	input dim = 300, output dim = 100
Linear Layer 6	input dim = 100, output dim = 20
Linear Layer 7	input dim = 20, output dim = 35
Linear Layer 8	input dim = 35, output dim = 12
Activation Function	Tanh
Total Parameters	1208131

Table 3.3: the CDLnet Architecture

Layer type	Input channel	Output channel	Kernel size	Stride	Padding
Conv1D Layer1	36	128	(3)	(1,1)	(1,1)
Conv2D Layer2	128	64	(2,2)	(2,2)	(0,0)
Conv2D Layer3	64	64	(2,2)	(2,2)	(0,1)
Conv2D Layer4	64	64	(2,3)	(2,2)	(0,1)
Linear Layer	1052	12	NA	NA	NA
Activation Function	ReLU				
Total Parameters	72208				

3.7 Results

The training curve for the CDLnet and baseline shown figures []. The loss value of the training curve stagnates above value 0.4. while the CDLnet was able to move below value 0.2. It is noted that in the beginning the binary cross entropy loss dominates in comparision to the μ multiplied. When the loss function goes below the 0.4 the network's accuracy of the collision detection is above 80 %. As discussed earlier the fully connected network does not able to learn features across time domain, because of that it cannot lower its loss function below 0.4. The training loss function for the CDLnet goes below 0.2. In this domain the binary cross entropy loss is smaller than the special regression loss. The network's ability of collision detection is above 99.5 %. The CDLnet obtains very high level of collision detection accuracy around 20 epochs.

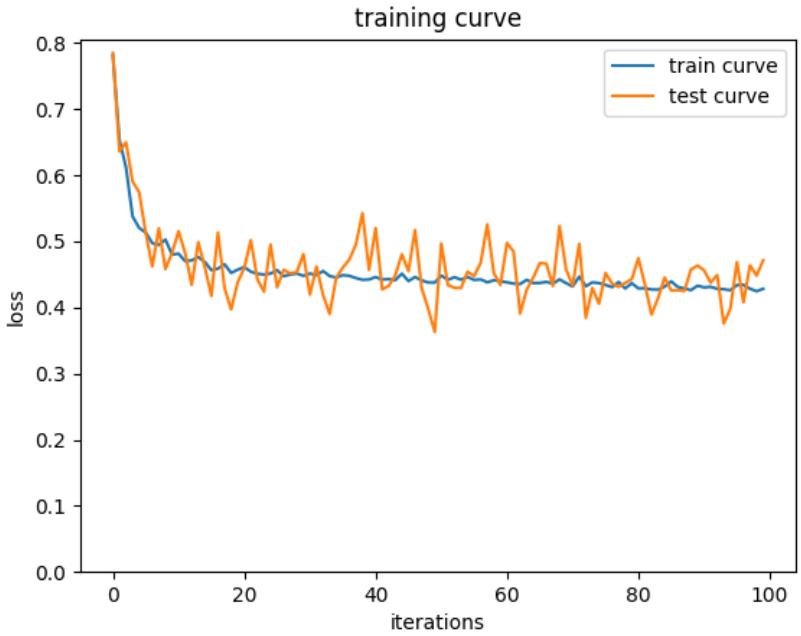


Figure 3.7: Train/Test curve for Baseline network

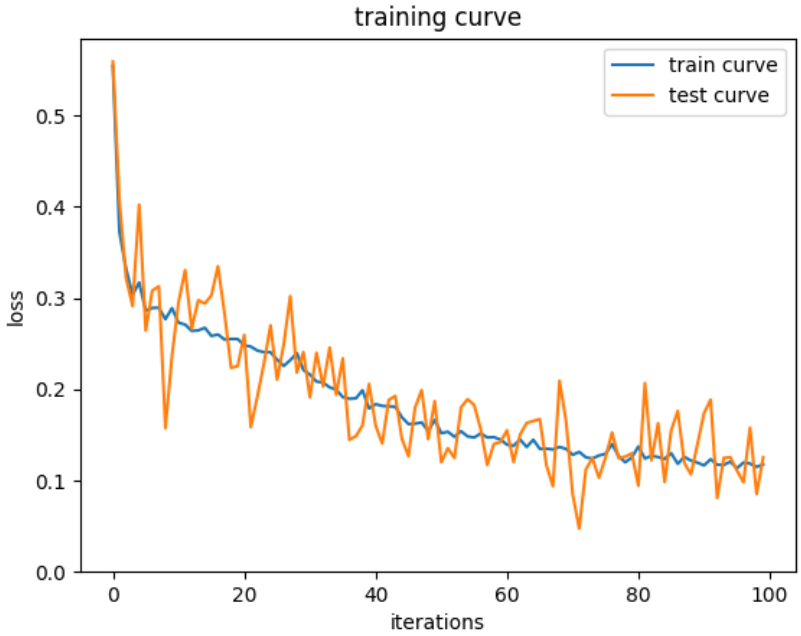


Figure 3.8: Train/Test curve for CDLnet

The baseline network also not able to learn the xy coordinates of the collision as shown in the histogram of the distance between true xy coordinates and xy coordinates predicted by the network. The baseline's output is really spend almost uniformly. The average error for xy coordinate prediction over all prediction goes over 3 cm for all runs. But for CDLnet, after 20 epochs the special regression loss becomes major loss function. As the number of epoch increases the network prediction of the xy coordinates improves. As seen by the plot of the histogram. The the average value of error on xy coordinates is 1.5 cm over mutiple runs. The CDLnet outperforms [23] by 0.5 cm.

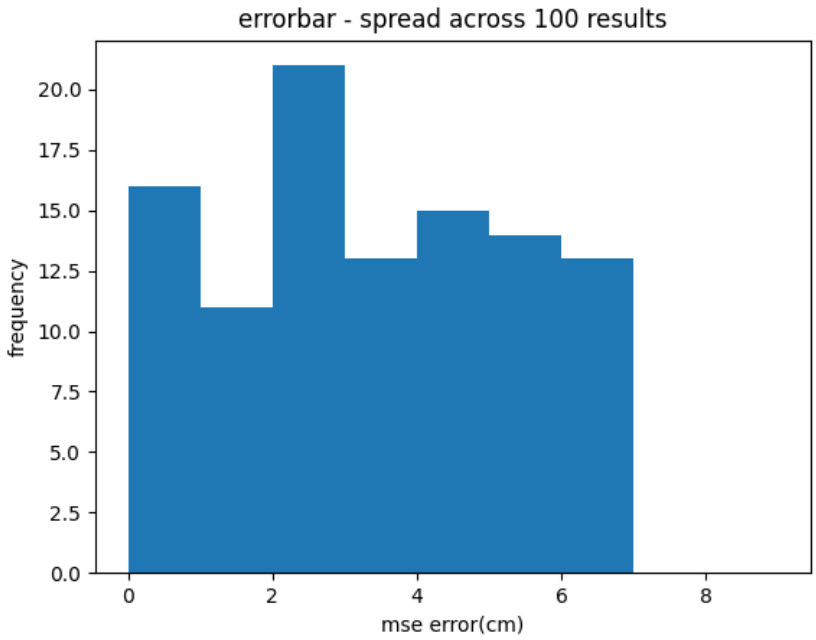


Figure 3.9: Histogram of distance error between True XY coordinates and Baseline network output

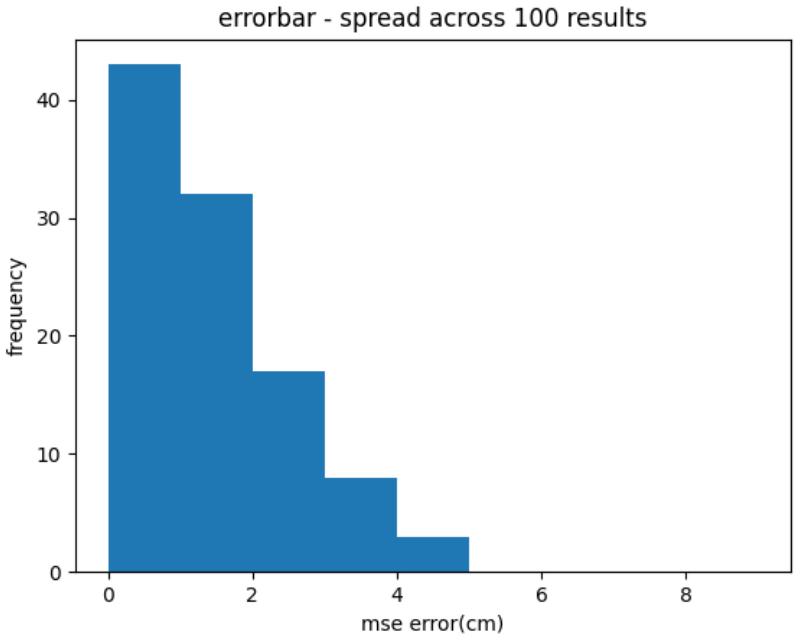


Figure 3.10: Histogram of distance error between True XY coordinates and CDLnet network output

The some of the network outputs are shown for both for baselines and CDLnet. The results are shown for randomly generated data in figures () and (). The results gathered over multiple runs. In many cases baseline network also fails to learn and gives some average value for XY coordinates. The dots represents correct XY coordinates from training data. The star represents the network output. The dots given red color it network was not able to recognize the collision detection. As network outputs XY coordinates regardless of the collision probability. From plots one can see that baseline performs performs far worst than CDLnet.

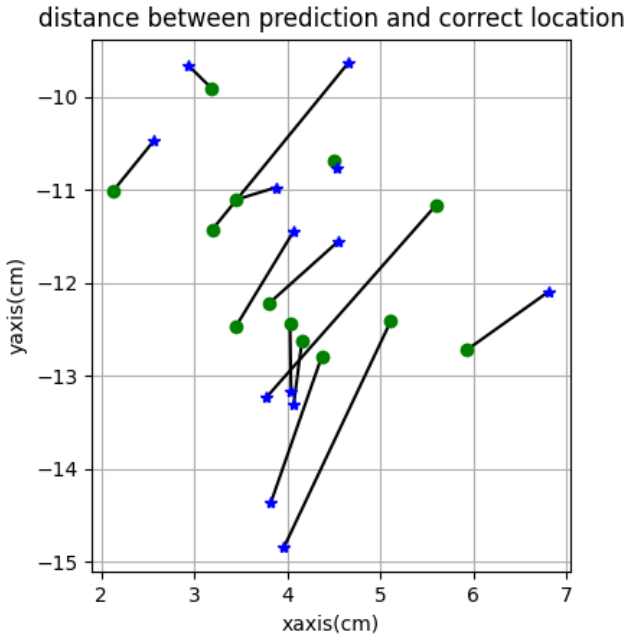


Figure 3.11: The CDLnet XY coordinate output wrt to real collision

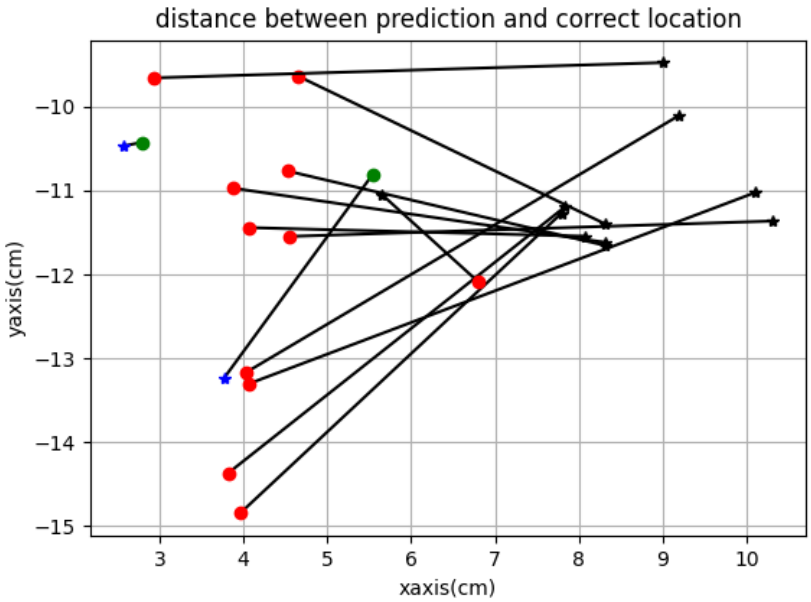


Figure 3.12: The best Baseline XY coordinate output wrt to real collision

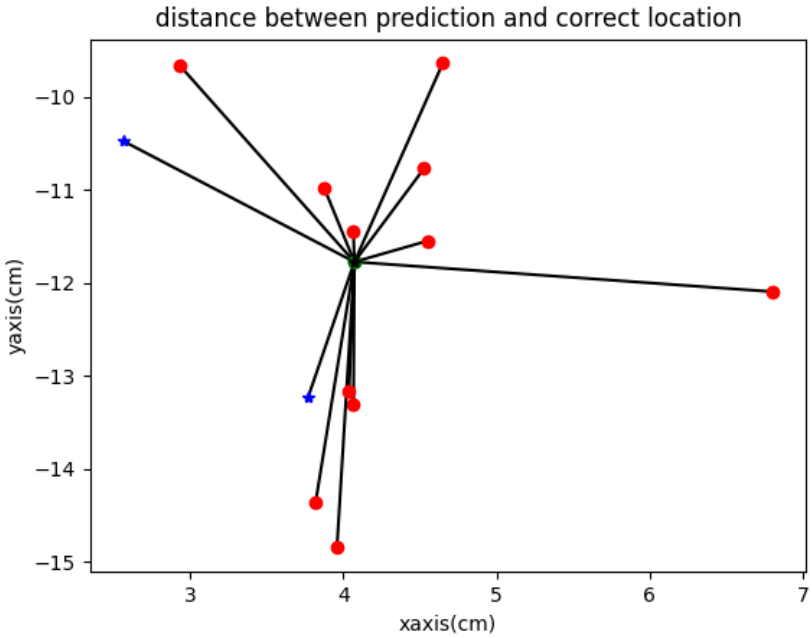


Figure 3.13: The worst Baseline XY coordinate output wrt to real collision

Bibliography

[1] JulieK/juliek.org(CCBY-NC-ND3.0). Accessed: 2021-11-21.

[2] https://locuslab.github.io/mpc.pytorch/. Accessed: 2021-11-21.

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

[4] Erwin Coumans and Yunfei Bai. Agents at pybullet envs at pybullet, a python module for physics simulation for games, robotics and machine learning. http://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym/pybullet_envs/agents, 2016–2021.

[5] S. Zaloga and J. Grandsen. *Soviet Tanks and Combat Vehicles of World War Two*. Arms and Armour Press, 1984.

[6] Jasmine A. Nirody, Lisset A. Duran, Deborah Johnston, and Daniel J. Cohen. Tardigrades exhibit robust interlimb coordination across walking speeds and terrains. *Proceedings of the National Academy of Sciences*, 118(35), 2021.

[7] https://www.britannica.com/animal/millipede. Accessed: 2021-11-21.

[8] S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.

[9] Marko Bjelonic, Ruben Grandia, Oliver Harley, Cla Galliard, Samuel Zimmermann, and Marco Hutter. Whole-body mpc and online gait sequence generation for wheeled-legged robots, 2021.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[11] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[12] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots, 2018.

[13] Gavin Kenneally, Avik De, and D. E. Koditschek. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1(2):900–907, 2016.

[14] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2020.

[15] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

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

[17] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, jul 1985.

[18] `https://github.com/pvigier/perlin-numpy`. Accessed: 2021-11-22.

[19] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions, 2020.

[20] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *CoRR*, abs/1804.10332, 2018.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[22] Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. Robot collisions: A survey on detection, isolation, and identification. *IEEE Transactions on Robotics*, 33(6):1292–1312, 2017.

[23] Sean Wang, Ankit Bhatia, Matthew T. Mason, and Aaron M. Johnson. Contact localization using velocity constraints. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7351–7358, 2020.

[24] `https://cs231n.github.io/convolutional-networks/#overview`. Accessed: 2021-11-21.

[25] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[26] Rok Pahič, Barry Ridge, Andrej Gams, Jun Morimoto, and Aleš Ude. Training of deep neural networks for the generation of dynamic movement primitives. *Neural Networks*, 127, 04 2020.