Training primitive skills to soccer bots

A B.Tech Technical Project Report
by

Rishabh Shah
150050006

Supervisors:
Shivaram Kalyanakrishnan
and
Parag Chaudhuri

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)
6 May 2019

Abstract

The research on soccer robot has seen rapid development in recent years. RoboSoccer is focused on training robots to play soccer at a level comparable or even better than humans in the near future. Currently, even the basic skills like balancing, walking and kicking are learned based on mechanical models that need further optimization of skill-specific parameters. Through this project, we explore a generalized method to teach soccer robots these basic skills. Reinforcement Learning's(RL) applications in robotics are of great interest because of their wide applicability. We show that RL can be leveraged to learn some simple skills. We formulate this problem as RL task where the robot takes actions by adjusting torques to appropriate joints. The rewards experienced are based on pose similarity to motion clips (designed to enact the same skill). Such motion clips provide a model for the robot to follow. The robot then learns to mimic the motion clip while simultaneously trying to balance itself in the simulated world. This is done in two phases namely Retargeting and Training. In Retargeting phase, we try to transfer "motion" from existing motion clips to our RL agent's joint hierarchy. In the Training phase, we train our agent using the data obtained from the retargeted motion clip with the help of a standard RL policy gradient method known as Asynchronous Advantage Actor-Critic (A3C). With extensive testing and tuning, we have been able to successfully train our agent some very primitive skills (both periodic and non-periodic) like hand waves, squats and walking in place.

This report explains the approach taken towards solving this task and the challenges faced in doing so.

Acknowledgements

I would like to express my deepest appreciation and gratitude towards my guides Prof. Shivaram Kalyanakrishnan and Prof. Parag Chaudhuri for providing me with the golden opportunity to work in this field. Their contribution in stimulating suggestions and encouragement have helped me to complete my project and this report.

Rishabh Shah
Indian Institute of Technology Bombay
6 May 2019

Table of Contents

Chapter 1

Introduction

1.1 RoboCup

The ultimate goal in AI, probably in robotics, is to build intelligent systems capable of displaying complex behaviors to accomplish the given tasks through interactions with a dynamically changing physical world. RoboCup simulation has emerged as an excellent domain for researchers to test ideas in machine learning and extend the reach of AI. In a generation where machines have evolved to beat humans in the game of Chess and Go, learning in the RoboCup soccer domain involves a whole new set of challenges, such as a continuous multi-dimensional state space, noisy sensors, need to cooperate with other agents to form team strategies and the need to act in real-time. Machine learning techniques have been used in the past on a wide range of tasks in RoboCup that include both individual behaviors and team strategies. We explore one such strategy to learn a generic individual behavior.

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment to maximize some notion of a cumulative reward. In context of RoboCup, we try to explore this idea to train soccer bot to learn basic skills like walking and kicking. The current walk of these bots is somewhat unnatural, and we feel that if it is made more human-like then perhaps robots could learn to perform complex tasks like passing and dribbling much more aptly and therefore get one step closer towards achieving human level competence. To make if more human-like, we model the rewards in our reinforcement learning task in such a way that robots try to mimic human motion while accounting for physical forces that act on it.

For good rewards, we need motion clips to be based on our robot hierarchy. A skeleton hierarchy is composed of a series of joints and joints chains with hierarchical relationships. Each joint in a skeleton hierarchy is a child joint and a parent joint. A parent joint is any joint higher in a skeleton's hierarchy than any of the other joints that are influenced by that joint's actions.

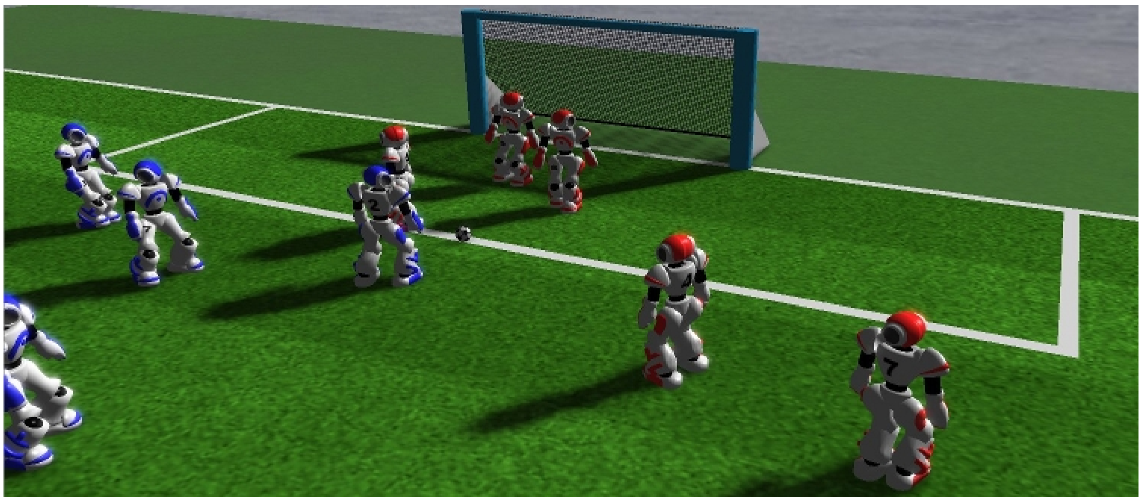


Figure 1.1: 3D RoboCup Simulation with two teams of Red and Blue Nao Agents

Standard available motion clips are based on their own specific skeleton hierarchy. Transforming these motion clips to fit on our agent's skeleton hierarchy poses a difficult challenge. Motion retargeting deals with this specific problem on a very generic scale. The objective is to adapt an animated motion from one character to another under some physical constraints. In our case, we have to overcome the issue of different joint lengths while simultaneously constraining correct foot placements.

1.2 Problem statement

Under the standard rules of RoboCup Competition[15], and physical forces like, friction, gravity and others we need to train a standard Nao agent to walk (and perform other basic skills later) smoothly without falling. This process has to be accomplished via training the soccer bots using human motion clips so that the resultant walk is more natural. This requires agent learning to provide a right set of torques to appropriate joints at right time in the simulation.

1.3 Source Code

This report along with all the code for this project can be found in this Github Link. We have also added a requirements file that contains all the necessary commands needed to setup this project as well as install its dependencies.

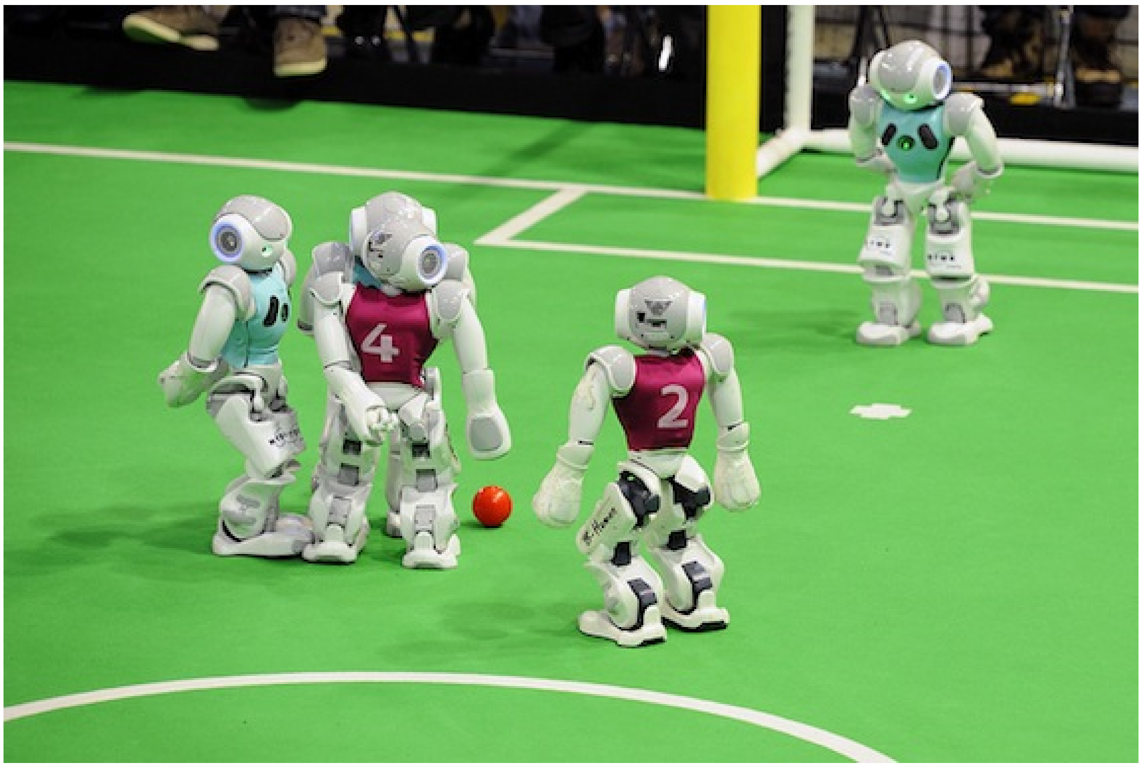


Figure 1.2: Real life 3D RoboCup competition between two teams of Red and Blue Nao Agents

1.4 Work Partition

In part 1 of this project, we had designed the general framework for training, getting rewards, running simulations that essentially enabled us to do testing and experimentation in part 2. The retargeting problem as well as other major server flaws were debugged in part 2. All the clips have been trained in this stage.

1.5 Report Outline

The rest of this report is organized as follows. In Chapter 2 we outline the existing literature related to our work. Chapter 3 describes the approaches we tried along with the challenges we faced. In Chapter 4 we provide an overview of our implementation framework. We present our results in Chapter 5 followed by conclusion and future extensions in Chapter 6.

Chapter 2

Related Work

Numerous methods have been explored for training agents to walk using RL. Since our work involves a mix of animation and learning to train agents we focus on most closely related work in motion retargeting and RL.

2.1 Training Physics Based Models

[MacAlpine et al. 2012][7] designed architecture for an omnidirectional walk to be used by a humanoid robot soccer agent acting in the RoboCup3D simulation environment. The walk is based on a double linear inverted pendulum model. Their walk engine chooses trajectory such that center of mass of the model swings over the stance foot. After finalizing the position of end effectors via this method, they use inverse kinematics to get positions for rest of the joints and use standard PD controller to calculate the low level torques to be applied to make the bot reach those joint orientations. This is a typical example of using physics-based models that rely on compact set of parameters tuned in order to achieve desired behaviors. The walk though effective, results in unnatural gaits. Also, this method cannot be extended to train robots other kind of complex skills without relying on human insight to implement task-specific strategies. Consequently, we explore RL approach to try to make the walk more human-like as well as to build a generic framework to teach various skills to soccer robot. The RL approach can also be more powerful in learning complex skills easily without much manual parameter tuning.

2.2 Reinforcement Learning

The introduction of deep neural network models for Reinforcement Learning has given rise to simulated agents that can perform a diverse array of challenging tasks. It offers one of the most general framework to take traditional robotics towards true autonomy and versatility. However, applying RL to high dimensional movement systems like humanoid robots remains a challenging problem. [Peters et al. 2003][12] discuss merits of various approaches for use of RL in robotics. Policy gradient methods have emerged as an algorithm of choice for many continuous control problems [Sutton et al. 1998][13]. However, the resulting behavior learnt through these methods appears less natural than their more manually engineered counterparts. One of the possible reasons is the difficulty in specifying reward functions for a natural movement. Crafting reward functions tailored to specific simulation task requires a substantial degree of human insight. [Xue Bin Peng et al. 2018][11] came up with innovative solution to integrate human motion clip data with the reward function for RL agent. Giving higher rewards for mimicking the human motion clip makes the policy/behavior learnt by agent more natural (human-like). We try to adapt this idea to our soccer framework to recreate the results obtained by [Xue Bin Peng et al. 2018][11] on our robot hierarchy.

2.3 Maintaining Balance and Stability

Many skills (like squats and walking) require robot to learn to balance properly throughout the simulation. For Nao[15] robot, this in itself a challenging task. [Tutsoy et al. 2017][14] have tried RL approach along with complete symbolic inverse kinematics just to balance the full lower body of Nao agent. We incorporate some of their ideas in our RL task formulation just to make sure that robot learns to both imitate the motion and maintain its balance without falling. [Xiaochen et al. 2017][5] introduced standard stability criteria and provided some common balance strategies. We try to incorporate some of these conditions while designing our reward function. [Yang et al. 2017][16] also tried to make use of deep RL strategies to learn specific small joint movements on humanoid robots like active push-off of ankles and maintaining the center of mass inside the support polygon to ensure stability. We use some of their ideas to design state space and reward functions to ensure balance.

2.4 Motion Retargeting

For tailoring motion clips to our bot's skeleton hierarchy we need to solve problem of Motion retargeting. It is essentially transfer of motion (i.e. joint orientations for all frames) from one character hierarchy to another. [Gleicher et al. 1998][3] presents a technique for retargeting motion focusing primarily on adapting the motion of one figure to another with identical structure but different segment lengths. The algorithm is not just to naively copying the angles from one joint to other but also to use inverse kinematics to satisfy end effector constraints. Most other retargeting approaches model constraints to make sure the final animated clip looks natural and similar to original. [Al Borno et al. 2018][1] explores the space of robust physics based motion retargeting. They optimize a physical model to approximate an old observed trajectory for the new skeleton.

Chapter 3

Our Approach

3.1 Overview

We divide our work into two phases - Retargeting and Training Phase. In Retargeting Phase we try to create a suitable motion clip for our RL agent to mimic. In Training phase, we train our RL agent using the data obtained from the motion clip from previous phase.

3.2 Retargeting Phase

We refer to our robot's hierarchy as `nao_hierarchy` and motion clip's bvh hierarchy as `mocap_hierarchy`.

3.2.1 Tools

We chose to use Blender as our primary BVH tool due to its easier visualization and modeling interface for creating both the `nao_hierarchy` and animation. However, some of Blender's inherent flaws in visualizing local pose orientation made it hard to debug the retargeting process. Nevertheless, due to unavailability of other BVH debugging tools, we had to use blender for most of our processing.

3.2.2 Challenges

As we can see from Fig 3.2 and Fig 3.3, both the bone lengths and number of joints are different in the `nao_hierarchy` compared to the `mocap_hierarchy`. Thus, to use the motion clip for our purposes we had to retarget the motion clip to our `nao_hierarchy`. This was a non-trivial task involving unforeseen challenges. Firstly, Nao agent has different sets of restrictions on maximum and minimum angles at each hinge joint. Therefore, not all motions could be retargeted to `nao_hierarchy`. Moreover, the hip joint in Nao agent has its axis aligned 45 degrees to standard axes and the complementary left and right hip joints are restricted to have same local rotations as seen in Fig 3.4, which makes it harder to retarget any motion using any available tools online. Also, Nao agent has lesser degrees of freedom in almost all joints as compared to the

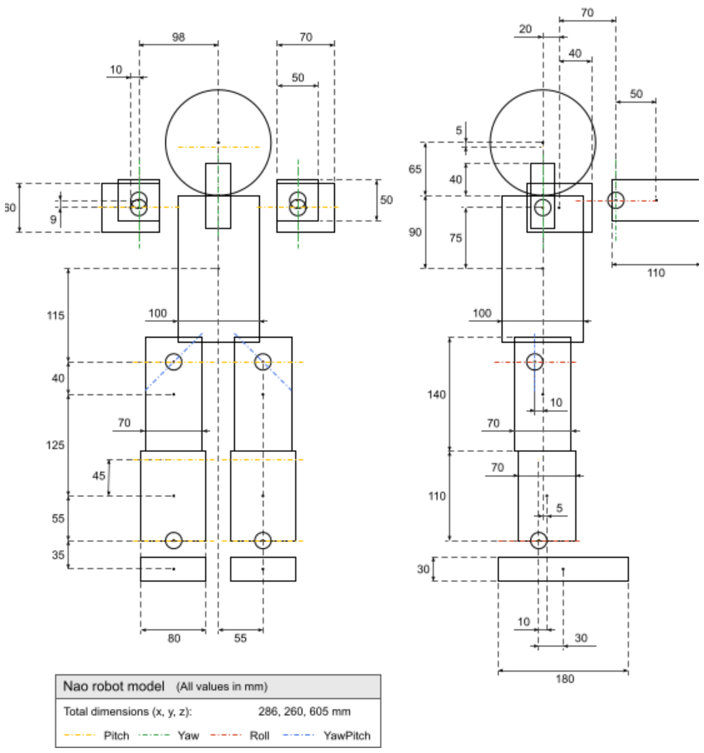


Figure 3.1: Standard Nao Agent Blueprint used in RoboCup Simulation from Simpspark specifications

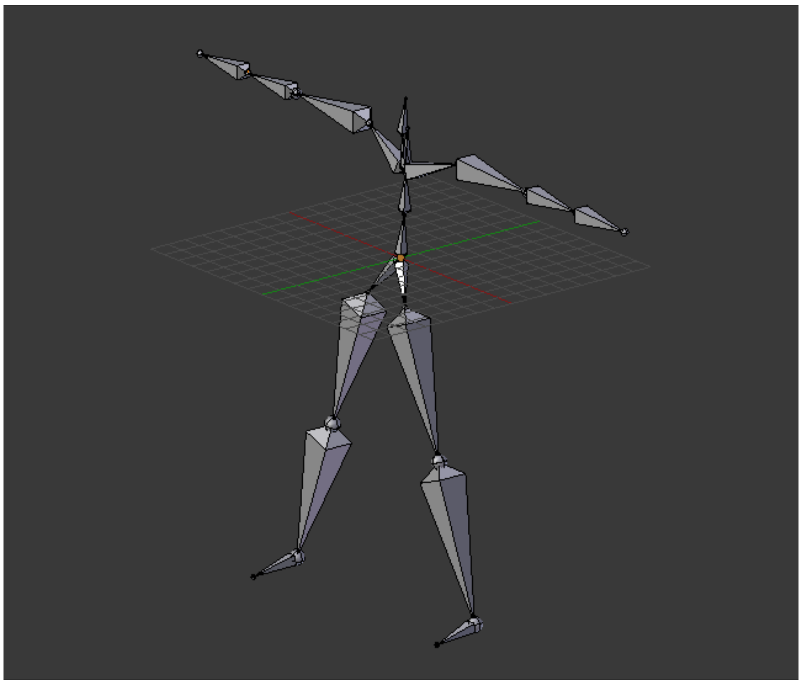


Figure 3.2: Hierarchy of motion clip from walk in place clip before retargeting

`mocap_hierarchy`. Therefore, retargeting problem for `nao_hierarchy` was not easy to solve. Thus, to remove dependencies between the two phases, we manually crafted some easy motions like squats and hand wave for debugging training phase so that none of the retargeting errors are responsible for failure in training phase.

3.2.3 Method

We manually created a hierarchy in BVH(BioVision Hierarchical data)[8] format for our Nao[15] agent (which we have referred to as `nao_hierarchy`) using the specifications from manual as can be seen from Fig 3.1.

Thereafter, we gathered motion clip data from various sources like CMU and work of [Chris Hecker et al. 2008][4] and finally chose two simple enough clips that would be easier for agent to learn. First clip described a walk in place motion where in the agent just lifts its legs one by one trying to balance itself. Second clip was a simple hand wave motion. The motion clip originally was made for a skeleton way different than Nao's architecture as seen in Fig 3.2 (which we have referred to as `mocap_hierarchy`).

After taking ideas from the approach of retargeting manually from [Gleicher et al. 1998][3] we devised a mechanism to retarget any generic motion onto our `nao_hierarchy`. This involves

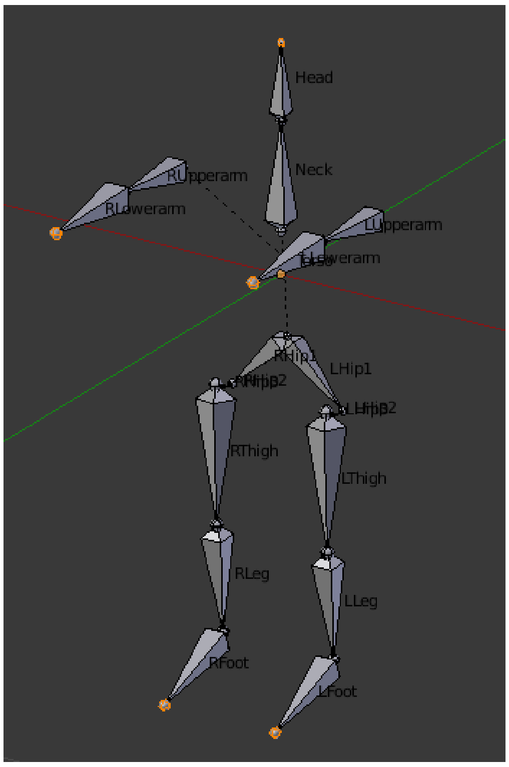


Figure 3.3: Blender view of Nao Agent Hierarchy constructed from specifications from Simspark sever and axis restrictions

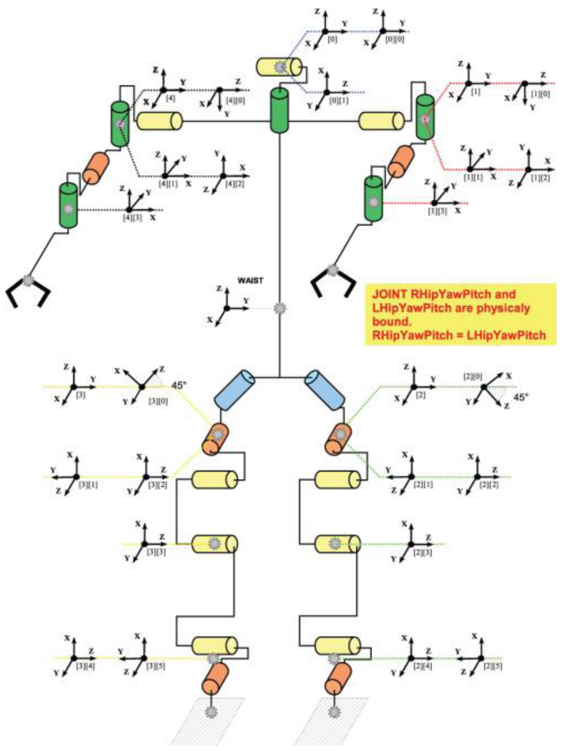

WAIST
JOINT RHipYawPitch and LHipYawPitch are physicaly bound.
RHipYawPitch = LHipYawPitch
45°
45°

Figure 3.4: Nao parameters indication constraints and axis alignments for all joints

firstly mapping all joints in `nao_hierarchy` to some unique joint in `mocap_hierarchy`. Now endpoint of these joints would serve as target positions for our animation to achieve. In second step, we adjust the rest pose (where all the local joint angles are 0) and copy all rotations for every frame of motion clip from `mocap_hierarchy` to `nao_hierarchy`. Finally, we adjust global target positions of every `mocap_hierarchy` joint for the difference in bone lengths and use standard jacobian full body inverse kinematics solver to fine tune the local rotations to make sure corresponding joints of `nao_hierarchy` reach the respective target positions in appropriate frames. Some manual tuning was involved to make sure the final animation didn't have sudden large changes in rotations across adjacent frames. To tackle the issue of lesser degrees of freedom in our Nao agent, we convert local retargeted rotations into Euler angles, take the required components and discard the remaining ones. This means that its possible that even though the retargeted motion looks good on unconstrained `nao_hierarchy` but might not work after removing some specific channels. We made tweaks to rest pose and IK (Inverse Kinematics) parameters to ensure this did not happen for the motion clips we decided to retarget.

3.3 Training Phase

Now to train the agent, we resorted to policy gradient methods due to their proven advantages in domain of continuous action and state spaces. We used the renowned method of Asynchronous Advantage Actor Critic[9] to train our agent.

3.3.1 States and Actions

In our case the action space consists of the angular velocity values of 22 joints (4 on each arm, 6 in each leg, 2 for head and neck as can be seen in Fig 3.4) of the agent. For state space, initially we just used these 22 joints' orientation in degrees, but robot was not even able to learn simple motion clips like hand waves. Applying ideas from [Yang et al. 2017][16] we included various other features in our state. These include, the gyrometer values, accelerometer values, height of pelvis from the ground. For learning not to make quick movements we also had to provide input individual joint linear velocities as input to our state. Finally, to learn to periodically repeat motions, we included time as another parameter so that by only resetting time at the end of motion we can make the bot perform periodic motions. We also normalized these inputs by using empirical maximum and minimum values as seen in various simulations. Without this step, learning was very slow for some motion clips which relied on more than one kind on inputs to generate proper torques. The state space now consists of more than 50 values, as can be seen in Fig 3.5

3.3.2 Rewards

Since our motion clips don't involve agent moving forward or any major changes in position of the robot over time, we had to worry about only two kinds of reward functions. First one is *imitation reward* that makes sure that our agent mimics the motion clip to its best extent. Second one is *balance reward* that is specifically purposed to make sure our agent doesn't fall down/lose balance while performing these complicated skills.

Imitation Reward

Given the current time in the simulation, we found the corresponding pose in the motion clip. By pose, we mean a tuple consisting of orientation of all 22 joints in the `nao_hierarchy`. This pose was used to calculate a similarity metric between current state of and the desired pose in the motion clip. We experimented with various reward functions for capturing the similarity of our agent's pose and target pose of motion clip. Firstly we tried negative of the euclidean distance between the two joint orientation vectors as the similarity metric as it captures the essence of robot's deviation from its current expected pose.

$$R_{imitation} = -\sum_{i=1}^{22}(n_i - w_i)^2$$

`where` n_i `is the ith joint orientation in nao agent pose in degrees`
`and` w_i `is the ith joint orientation in motion clip pose in degrees`

This wasn't working very well with the A3C algorithm as the learning curve suddenly dropped after 20-30k iterations leading to poorly trained model. After further experiments we found another function of the euclidean distance to stabilize the learning process.

$$R_{imitation} = e^{-\sum_{i=1}^{22}(n_i - w_i)^2}$$

`where` n_i `is the ith joint orientation in nao agent pose in degrees`
`and` w_i `is the ith joint orientation in motion clip pose in degrees`

Balancing Reward

We experimented with various strategies for this reward too. One way we tried was to check if the agent is about to fall or is already down. This involved tuning falling condition threshold by using accelerometer data from the agent. We then terminated the episode with large negative reward whenever this happened.

$$R_{balance} = \begin{cases} -k & \text{if agent fallen} \\ 0 & \text{otherwise} \end{cases}$$

`where` k `is a large positive constant`

This was not working well due to the fact that agent learnt to "almost" fall at the end of motion clip losing stability way too soon getting overall high rewards. So, to make sure agent never ever loses stability we used a reward corresponding to maintaining pelvis height from ground above certain level and keeping gyrometer values below some empirical threshold. Using this reward we found that agent doesn't learn to fall but tries to maintain stability throughout the simulation.

$$R_{balance1} = \begin{cases} k1 & \text{if } |\text{gyr}| < \text{t1} \\ 0 & \text{otherwise} \end{cases}$$

$$R_{balance2} = \begin{cases} e^{\frac{1}{h}} & \text{if } \text{h} > \text{t2} \\ 0 & \text{otherwise} \end{cases}$$

`where` $k1$ `is a small positive reward in case the gyrometer values (`gyr`) are less than a threshold (`$t1$`)`
`and` h `is the height of pelvis from the ground`
`and` $t2$ `is another threshold for the height`

Finally the total reward at the end of one action step is as following

$$R_{total} = a \times R_{imitiation} + b \times R_{balance1} + c \times R_{balance2}$$

`where` a, b, c `are weights tuned empirically for each of the three different rewards)`

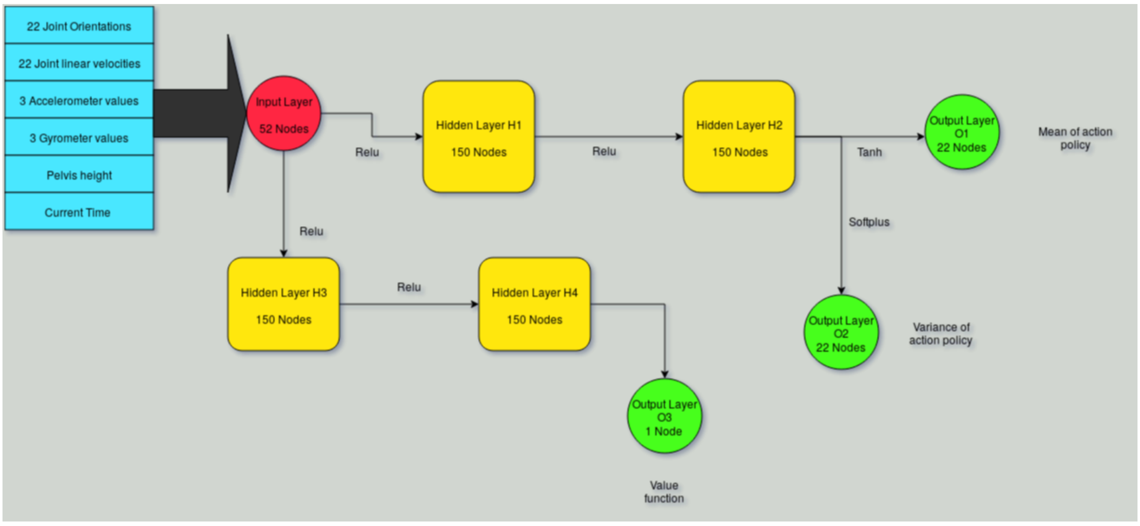


Figure 3.5: Neural Network architecture for design used to train robot via A3C.

3.3.3 Neural Net

As with other parameters of this phase we experimented with various architectures for neural network. Finally, we settled for an architecture where policy was modeled by two neural networks sharing two hidden layers to model mean and variance of policy (actor network). We used an independent additional network for computing value function (critic network). The two hidden layers enable the robot to learn even complex actions for every joint. The number of nodes in hidden network was experimented with, and we found that since the state space is large, models trained with around 150 nodes in each hidden layer are good enough to mimic the motions to great accuracies. Also, this kept a balance between time to learn and level of skill learnt.

For mean of the policy, we tried training both with and without tanh activation at the output layer(can be seen in Fig 3.5). Since some motions might need larger angular velocities than the unit value from tanh's output, we had to scale the means by a span threshold. When threshold was kept small, the agent learnt to alternate between extreme ends just to make sure to follow the motion clip. This created undesired vibrations in final trained model. Therefore, we tuned the threshold to stop such oscillating behavior of our agent. Without the tanh activation, in some cases where the means values went very high, they didn't return to their stable values (due to unpredictable behavior in other robot joints at high angular velocities). Thus, we settled on using tanh as the activation at the output layer for mean. For variance to make it positive we used a standard softplus activation. Regarding inner layer activations, we found that neither sigmoid nor tanh activations gave good results perhaps due to dampened gradients and slow overall learning. Relu activations proved to be best amongst all others and made it to the final architecture.

Chapter 4

Implementation

4.1 Libraries Used

We chose Python as our primary language due to its fantastic machine learning support. PyTorch libraries provided excellent API for working with neural nets. The code involves 2 parts. One is the client(agent that is updating its neural net) and other is the environment that client is interacting with. Here the environment encapsulates the simspark server which actually executes actions and returns new state for the agent and runs the whole simulation. Many minor techniques of getting right acceleration values and checking if robot has fallen were obtained from repository of UT Austin Villa's code[6].

4.2 Server Client Communication

Python Simspark agent[2] was used to communicate with the simspark server. All communication between python-agent and simspark server was relayed through Environment class. We designed its interface similar to python open AI Gym environments. So we abstracted our reward calculation and implementation of features so that the agent could get the next state, reward and boolean flag for termination of episode from a `step(action)` function. This takes action as input and gives the above mentioned outputs after performing one step in simulation. This is equivalent to performing a transition in the MDP[13] for this problem. After every episode, we reset the agent connection to start over.

We faced a lot of problems with this server environment. Since simspark server development is still under progress, it crashes if multiple connections are made and therefore, we have coded a robust environment handling all server side errors and subtly making A3C agent discard the given episode and restart whenever connection between server and client fails abruptly. Another bug in the server was the fact that instead of providing with the next state it gave the current state as the output at every time step. To handle this, we introduced a dummy 0 action between every two consecutive actions sent by the agent. This was meant to do nothing but get

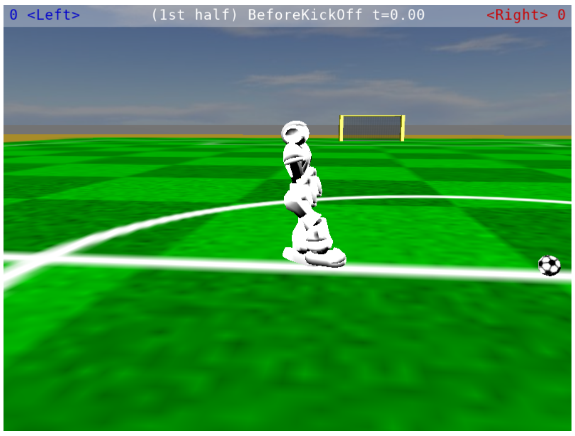


Figure 4.1: Visualization of learning of robot using A3C on a standard Simspark Monitor

next state for the current action. Another issue with this server was the initial position of robot. Instead of being in rest pose on ground, the server initialized the agent in a free fall state. The agent landed on the ground and stabilized in about 1.2 seconds. Such errors took a lot of time to debug due to lack of proper documentation for simspark. Moreover, corrupting the input data this could have been the reason for our initial failures.

4.3 Getting rewards and next state

Server provided most of the information to generate rewards and next state. The essentials things we received were orientations of 22 joints, our local position, accelerometer and gyrometer values. The rest of the state variables were calculated in the Environment class and abstracted out to the client.

4.4 Initial Pose and Rendering

For retargeted motion, we had to use almost 50-100 frames to get to the initial pose of the motion via smooth transitions. This was ensured by getting one motion clip pose ahead of time and calculating the difference in orientations and then gradually decreasing this difference. To visualize the progress while training the agent, we used the default standard monitor from simspark. The rendering part of the story was decoupled from the actual learning in the sense that one could visualize the agent only when required and not all the time while learning.

4.5 Training A3C Agent

The A3C[10] was trained asynchronously with 4-5 different local threads learning the weights using the different instances of same environment independently. These threads periodically synchronized their updates to a global neural net. The weights were updated after each episode using a fixed learning rate.

Chapter 5

Experiments and Results

5.1 Retargeting

In the retargeting phase, we have successfully transformed two motion clips that enact skills of *hand wave* (Fig 5.1) and *walk in place* (Fig 5.2). We were able to retarget these simple motion clips from model in Fig 3.2 to our Nao model (in Fig 3.3). From the below embedded videos, we can clearly observe the smoothness of movements and synchronization with the original clip in term of foot joints, hand and leg movements.

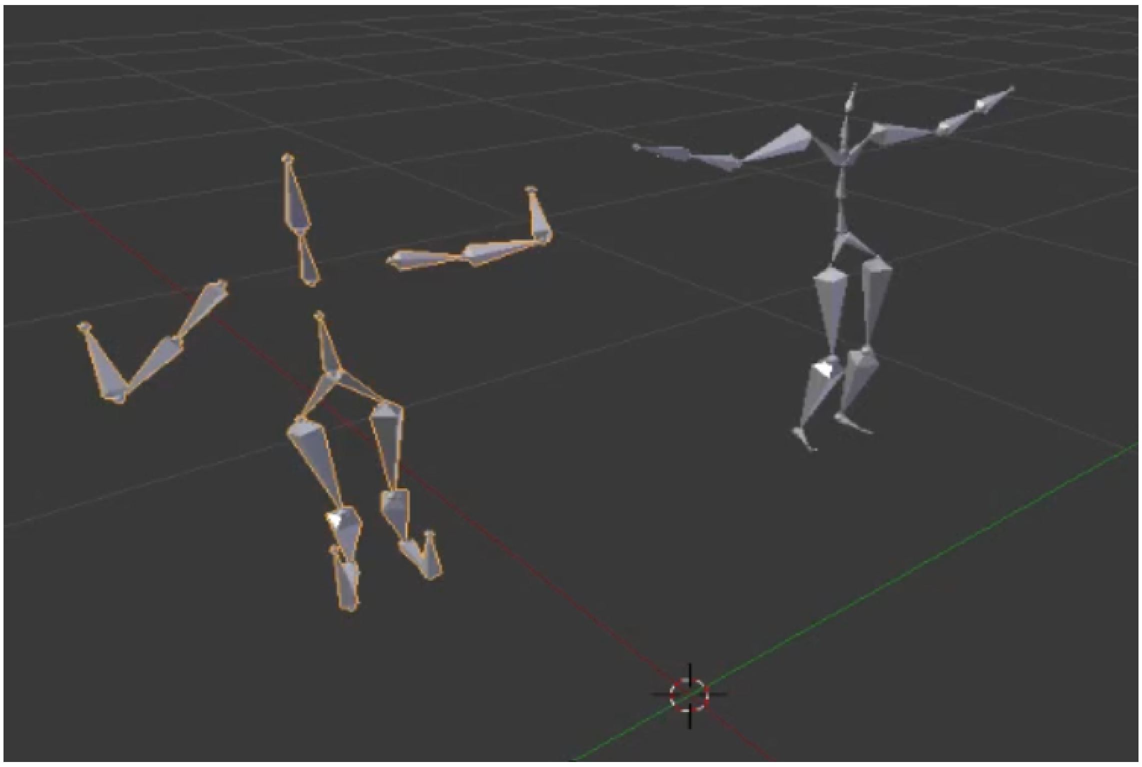


Figure 5.1: Retargeted motion clip for hand wave skill

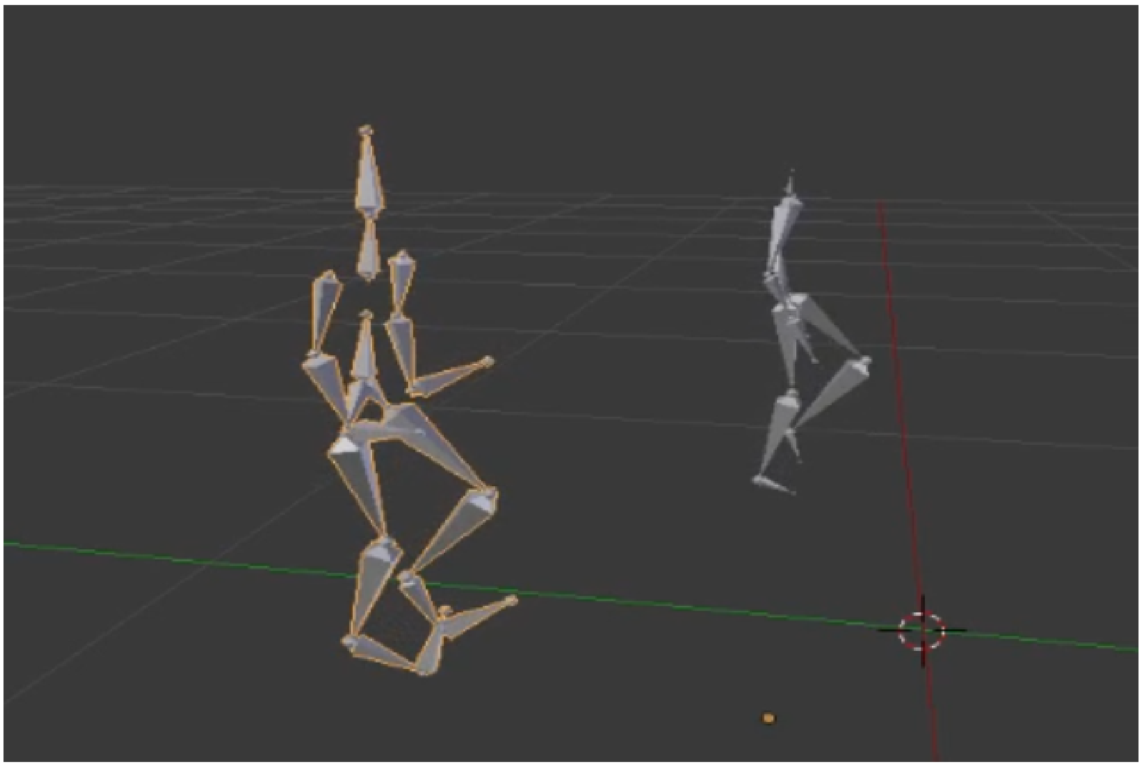


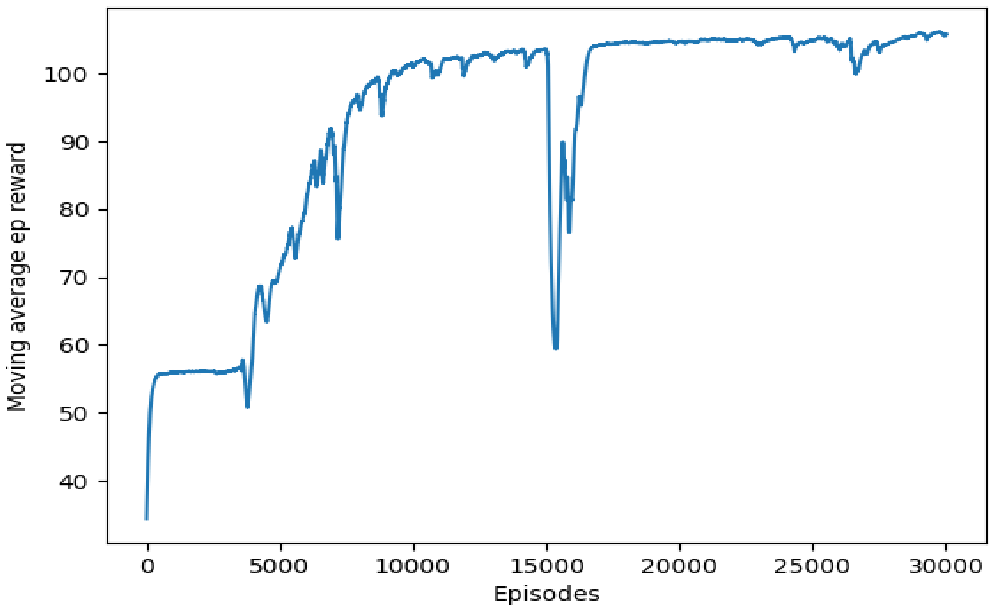
Figure 5.2: Retargeted motion clip for walk in place skill

5.2 Training without Retargeting

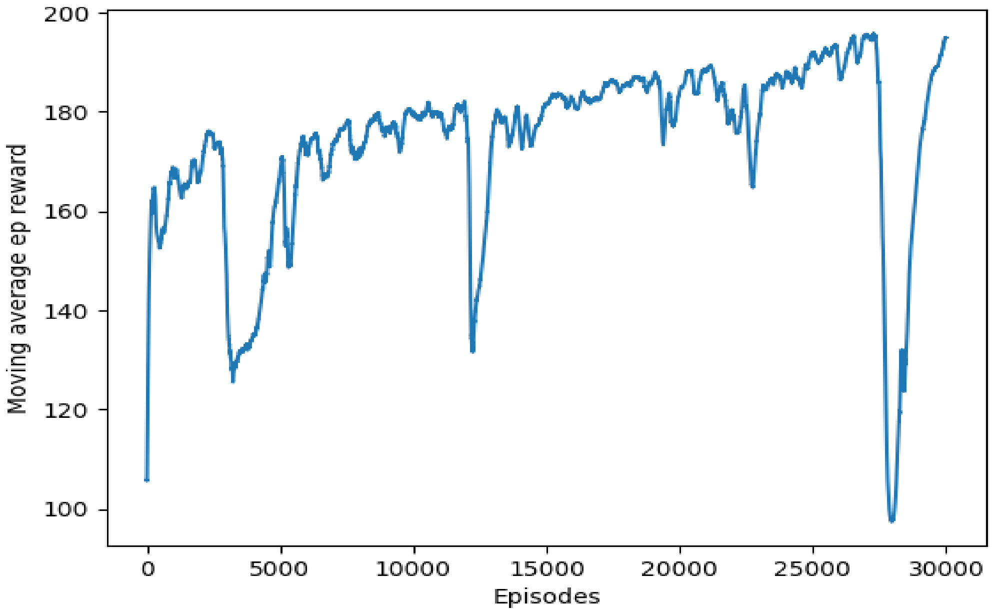
In the Training phase, initially to test the results of both phases independently we had manually created some motion clips which were free from retargeting errors. These include motion clips like hands opposite motion (in Fig 5.4a), periodic squats motion (in Fig 5.4b), manual partial walk in place motion (in Fig 5.4c). As you can see from these videos, that the robot is mimicking the motion to quite a good extent and further training will only improve these skills. All these skills have been trained using A3C code for around 50-60k episodes each using same reward function but with different state and action space. This space has been limited to only those joints whose orientation has non zero change involved in the motion clips (instead of all the original 22 joints).

5.3 Learning Curve

Since 60k episodes take almost an entire day to learn, we implemented a mechanism to pause the learning process. This is done by saving current weights (as well as at some intermediate points) and then loading them to pick up where we left off. This not only helped us save the partially trained model in presence of runtime bugs but also helped us resume from particular checkpoints which were not corrupted. In the Fig 5.3a, we can see that in first 30k episodes, the learning curve (moving average of cumulative rewards v/s episodes) increased very slowly for almost 5k episodes and then picked up the pace to reach the maximum achievable value (which in this case was 120). In the Fig 5.3b, we can observe that in next 30k episodes, the growth in learning curve is slow as expected. Also, many times the moving average suddenly drops and grows fast again. Such events are marked by frequent spikes in both learning curves. This might be because of the agent trying out newer approaches/trajectories to see if there is any possibility of better cumulative rewards.

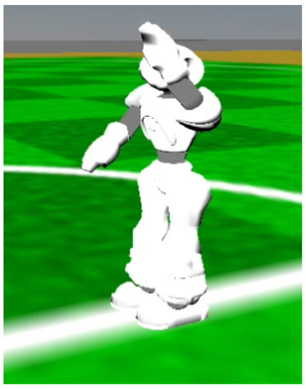


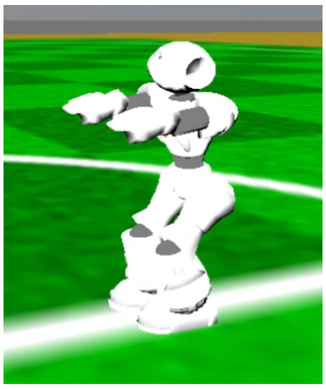
(a) Hands Opposite motion clip 1 - 30k episodes



(b) Squats motion clip 30k -60k episodes

Figure 5.3: Learning Curve for various motion clips in different stages


(a) Hands Opposite Skill


(b) Squats Skill

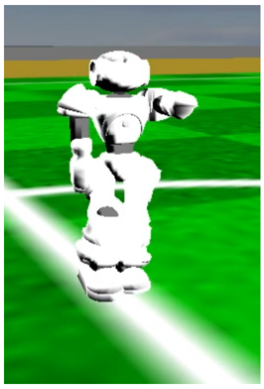
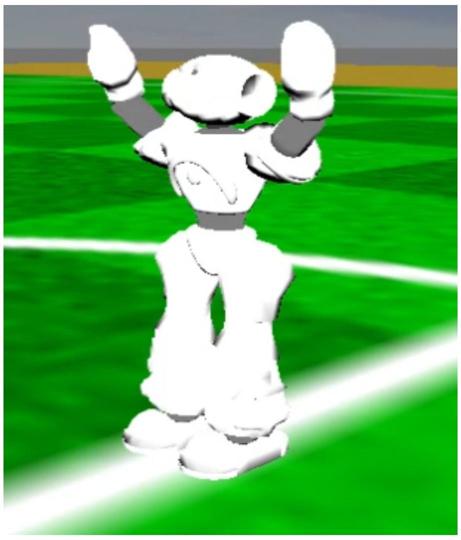

(c) Walk in Place Skill

Figure 5.4: Trained models for manually designed motion clips

5.4 Training with Retargeting

After thorough testing, we also tried simulations over retargeted clips. These include hand wave motion (in Fig 5.5a) and partial walk in place motion (in Fig 5.5b). As you can see from the video of hand wave motion, that since the robot is well-balanced it can learn to mimic this motion quite well. However in case of walk in place motion, robot is not able to lift its leg as it has perhaps found a local minimum where lifting its leg only makes it fall (resulting in lower total reward) but partially copying the motion clip while maintaining balance gives higher cumulative reward. In this scenario, even though the retargeting seems smooth, learning to balance is a difficult task for our agent.


(a) Hand Wave Skill

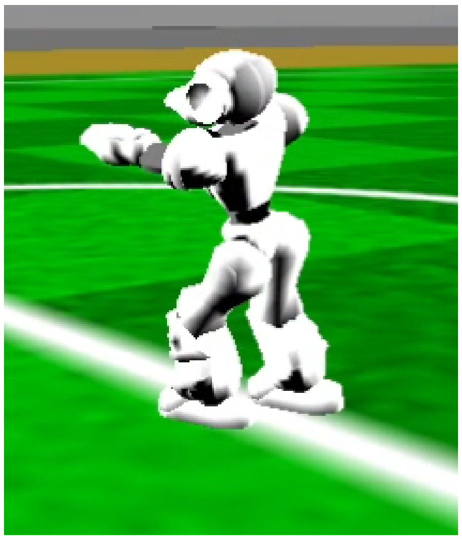

(b) Walk in Place Skill

Figure 5.5: Trained models for retargeted motion clips

Chapter 6

Conclusion and Future Work

Clearly, our method shows that RL can be used to learn skills easily with the help of motion clip rewards. For retargeted motions, the success is limited to motions where balancing is not required (like in hand wave motion) but for motions where we require balancing the robot struggles with handling balancing simultaneously with mimicking the motion clip. So if we could capture better motion clips tailor made for our nao agent we can surely make it able to not only walk but perform much more complex skills.

There is another important idea from [Xue Bin Peng et al. 2018][11], that could possibly make this RL-motion clip reward system work. The idea is to start the simulation from not just the initial pose but also arbitrary in between poses. This would clearly help the robot learn later parts of simulation equally well. Currently, without this method learning longer (more than 3-4 sec) motion clips is difficult and requires a lot more episodes. We tried to implement this idea in our project but getting robot in stable initial pose is in itself a challenging task and would require a lot of manual effort and human insight. Extensions of this work could try to implement this random restart method and see if it actually improves the results or not.

Furthermore, other target specific positional rewards could be used to get the robot to move along the ground(like described by [Xue Bin Peng et al. 2018][11]). After the agent learns a bunch of motions we can begin to teach it to switch between multiple motions at the right time which is very crucial during a soccer match.

In conclusion, We tried an innovative and promising approach of combining animation with reinforcement learning applied to a completely different domain which can make soccer agent's behaviors more human-like and perhaps more effective. Unlike earlier methods proposed, our approach doesn't require human intervention and creativity to optimize parameters for every particular type of behavior and agent can itself do what's best, provided we tune the reward functions in right manner once.

References

[1] Al Borno, M., Righetti, L., Black, M. J., Delp, S. L., Fiume, E., and Romero, J. (2018). Robust physics-based motion retargeting with realistic body shapes. In *Computer Graphics Forum*, volume 37, pages 81–92. Wiley Online Library.

[2] Edison, M. (2018). Python implementation of simspark agent. `https://github.com/edison-moreland/py-simspark`.

[3] Gleicher, M. (1998). Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA. ACM.

[4] Hecker, C., Raabe, B., Enslow, R. W., DeWeese, J., Maynard, J., and van Prooijen, K. (2008). Real-time motion retargeting to highly varied user-created morphologies. In *Proceedings of ACM SIGGRAPH '08*. `http://chrishecker.com/Real-time_Motion_Retargeting_to_Highly_Varied_User-Created_Morphologies`.

[5] Lai, X., Wu, X., Fang, Y., and Wang, X. (2016). A survey of balance control strategy for humanoid robots.

[6] MacAlpine (2018). Ut austin villa robocup 3d simulation team base code releas. `https://github.com/LARG/utaustinvilla3d`.

[7] MacAlpine, P., Barrett, S., Urieli, D., Vu, V., and Stone, P. (2012). Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*.

[8] Maddock, M. M. S. (2001). Motion capture file formats explained.

[9] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[10] Morvan, Z. (2018). Pytorch implementations of asynchronous advantage actor critic. `https://github.com/MorvanZhou/pytorch-A3C`.

[11] Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14.

[12] Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

[13] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[14] Tutsoy, O., Erol Barkana, D., and Colak, S. (2017). Learning to balance an nao robot using reinforcement learning with symbolic inverse kinematic. *Transactions of the Institute of Measurement and Control*, 39(11):1735–1748.

[15] Xu, Y. and Vatankhah, H. (2010). User manual simspark. http://www.cs.utexas.edu/ pstone/Courses/344Mfall10/assignments/Manual$_3d.pdf$.

[16] Yang, C., Komura, T., and Li, Z. (2017). Emergence of human-comparable balancing behaviours by deep reinforcement learning. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 372–377. IEEE.