Recurrent Neural Networks for Robotic Control of a Human-Scale Bipedal Robot

By
Jonah Siekmann

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 28, 2020
Commencement June 2020

AN ABSTRACT OF THE THESIS OF

Jonah Siekmann for the degree of Honors Baccalaureate of Science in Computer Science presented on May 28, 2020. Title:
Recurrent Neural Networks for Robotic Control of a Human-Scale Bipedal Robot

Abstract approved:

Alan Fern

Dynamic bipedal locomotion is among the most difficult and yet relevant problems in modern robotics. While a multitude of classical control methods for bipedal locomotion exist, they are often brittle or limited in capability. In recent years, work in applying reinforcement learning to robotics has lead to superior performance across a range of tasks, including bipedal locomotion. However, crafting a real-world controller using reinforcement learning is a difficult task, and the majority of successful efforts use simple feed-forward neural networks to learn a control policy. Though these successful efforts have demonstrated learned control of a Cassie robot, the wide range of behaviors that can be learned by a memory-based architecture, which include walking at varying step frequencies, sidestepping, and walking backwards, (all in the same learned controller) have not yet been demonstrated. In keeping with recent work which has shown that memory-based architectures have resulted in better performance on a variety of reinforcement learning problems, I demonstrate some advantages of using a sophisticated, memory-based neural network architecture paired with state of the art reinforcement learning algorithms to achieve highly robust control policies on Agility Robotics' bipedal robot, Cassie. I also visualize the internal learned behavior of the memory using principal component analysis, and show that the architecture learns highly cyclic behaviors. I show that dynamics randomization is a key tool in training robust memory-based neural networks, and that these networks can sometimes fail to transfer to hardware if not trained with dynamics randomization. I also demonstrate that various parameters of the dynamics, such as ground friction or ground slope angle, can be reconstructed by examining the internal memory of a recurrent neural network. This opens the door to automatic disturbance observers or online system-ID, which would be of significant benefit to problems which require hand-written disturbance observers requiring manual tuning.

Key Words: Reinforcement Learning, Robotics, Artificial Intelligence, Deep Learning, Neural Networks

Corresponding e-mail address: siekmanj@oregonstate.edu

Recurrent Neural Networks for Robotic Control of a Human-Scale Bipedal Robot

By
Jonah Siekmann

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 28, 2020
Commencement June 2020

Honors Baccalaureate of Science in Computer Science project of Jonah Siekmann
presented on May 28, 2020

APPROVED:

Alan Fern, Mentor, representing Electrical Engineering & Computer Science

---

Jonathan Hurst, Committee Member, representing Mechanical, Industrial and Manufacturing Engineering

---

Kevin Green, Committee Member, representing Mechanical, Industrial and Manufacturing Engineering

---

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University Honors College. My signature below authorizes release of my project to any reader upon request.

Jonah Siekmann, Author

Contents

1 Background

1.1 Robotic Bipedal Locomotion

In the field of 3D bipedal locomotion, classical control methods such as Zero Moment Point [1, 2, 3], Hybrid Zero Dynamics [4], and control hierarchies that use reduced order models (e.g. Spring Loaded Inverted Pendulum) in conjunction with model predictive control [5, 6], have made compelling progress. However, these control methods are often limited due to their dependence on local feedback or online optimization and result in either brittleness or slower than real-time evaluation. Furthermore, these approaches often require disturbance observers and state estimators to account for errors [7, 8] due to the highly unstable nature of dynamical systems. These are generally memory-based, containing some sort of hidden state which is updated in real-time, and can be seen as either predictive or history-compressing mechanisms usually requiring tedious amounts of hand-tuning gains.

The robot used in this work, Cassie, is an approximately human-scale bipedal robot, manufactured by Agility Robotics in Albany, Oregon. Cassie is powered exclusively by electric motors and can walk for up to four hours on a single charge. The robot is approximately one meter tall and has ten actuators (five in each leg). Designing a walking controller for Cassie is difficult in large part due to the complicated hybrid dynamics, the springs which create nonlinear torque response in several of the joints, and the dynamically unstable nature of bipedal locomotion. In the past, several effective classical control methods have been used to control Cassie, including using OSC paired with a SLIP reduced order representation as in Apgar et al. [9]. In recent years, methods involving applying deep reinforcement learning have found success in learning stable control policies [10], opening the door for further exploration into the application of deep reinforcement learning into dynamically unstable control problems.

1.2 Deep Reinforcement Learning

1.2.1 Reinforcement Learning

Reinforcement learning (RL) is a class of methods which train agents to maximize the expected return over some period of time using trial-and-error [11]. Reinforcement learning problems are usually formalized as the interaction between the agent and a Markov Decision Process (MDP) wherein at a given timestep t the agent receives a state s_t and produces an action a_t using its parameterized policy π_θ, and finally receives a reward r_t which varies depending on the 'goodness' of state-action pair. The agent's goal is to find the policy that maximizes the expected discounted sum of rewards, or $J(\pi_\theta)$:

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

Where π_θ is a *policy* parameterized by θ which the agent uses to choose actions , T is a

time horizon which limits the length of a state-action sequence, and γ is a coefficient which discounts the reward at a given timestep t which is typically in the range [0.9, 0.99].

One of the most popular categories of RL algorithms is the family of policy gradient methods. The guiding principle behind all policy gradient methods is optimizing the agent's policy by finding the gradient of J; in other words, they compute $\nabla_\theta J(\pi_\theta)$. Often, analytical computation of the gradient of J is expensive or impossible. Though its derivation [12] will not be covered in this work, the gradient can be approximated as follows:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta} \left[Q_{\pi_\theta}(s, a) \nabla_\theta \ln \pi_\theta(a|s) \right]$$

Where $Q_{\pi_\theta}(s, a)$ is a heuristic denoting the value of a state-action pair following the policy π_θ and the expression $\ln \pi_\theta(a|s)$ represents the log probability of selecting an action a given state s following the policy π_θ.

Recent algorithms like PPO [13] or TRPO [14] replace the Q function with an advantage function $A(s, a) = Q(s, a) - V_{\pi_\theta}(s)$, where $V_{\pi_\theta}(s)$ is a function which represents the value of a particular state when following the policy π_θ. Thus, the advantage function is a version of the Q function which has lower-variance due to the subtraction of the baseline state value. Intuitively, the advantage function can be seen as a measure of how much better an action was than the average action conditional on the state when following the policy π_θ. In the context of deep reinforcement learning, the policy π, state-action value function Q, and state value function V are usually represented by neural networks.

Recently, RL has shown significant promise on a variety of non-trivial control problems. These include problems in manipulation [15, 16, 17] and in legged locomotion, both in quadrupedal [18] and bipedal [10] locomotion. Reinforcement learning has the potential to remove a large degree of complexity from control systems, which tends to optimize over a much longer horizon, and reduce computational cost at the price of a black-box behavior. Yet while many attempts have been made attempting to use deep reinforcement learning to learn controllers, fewer have been made learning system identification or disturbance observation.

1.2.2 Dynamics Randomization

Applying reinforcement learning to robotic control problems often requires large amounts of data to be sampled in order to learn a useful policy; the number of samples required can be as high as tens of millions [10], sometimes equivalent to thousands of years of simulation time [16, 17]. Because of the prohibitively data-hungry nature of RL algorithms, training is usually done in a simulator. Training in simulation rather than the real world affords not only orders-of-magnitude faster training time, but also massive parallelism, further increasing time-efficiency.

However, using simulators to train reinforcement learning agents has serious downsides. Approximations or inaccuracies in the physics model, errors in the physical properties of the physics model, or other potential disagreements between the real world and the simulated world are the cause of what is known as the sim-to-real problem, or reality

gap [19]. Training in simulation may lead the agent to exhibit behaviors that exploit quirks of the simulator, but lead to poor or infeasible behavior in the real world. Training agents to learn policies that are robust to potential differences between simulation the real world is an open problem, but a few approaches appear to significantly improve the success rate of sim-to-real transfer; recent work has found randomizing parameters of the simulation dynamics, such as joint mass, joint damping, or ground friction, (known as dynamics or domain randomization) [15, 18] during training to be an effective tool to overcome the reality gap. By randomizing these parameters during training, agents are exposed to a wide variety of possible simulation dynamics, thus increasing robustness to the inevitably different set of dynamics experienced in the real world.

1.2.3 Neural Networks

Neural networks are a class of function approximators which have recently enjoyed growing popularity in the field of machine learning for being easy to train and useful for a variety of practical applications while bearing a superficial resemblance to networks of biological neurons.

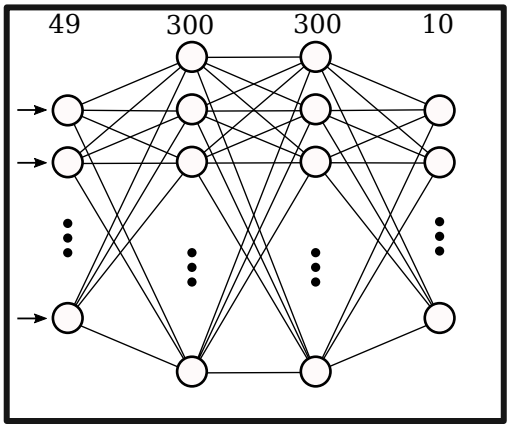

Figure 1: A possible structure of a conventional neural network.

Mathematically speaking, a neural network is a series of linear projections with nonlinear 'activation functions' interspersed between the projections. An example of such a neural network can be seen in Fig. 2. The edges connecting any two layers in a neural network can be seen as collectively representing a matrix multiplication, and the neurons in those layers as a nonlinear function applied to individual scalars in the projected vector. This function is usually an S-shaped logistic function, like the sigmoid or tanh functions.

Often, neural networks are trained using a method known as gradient descent, wherein the partial derivative of an objective function with respect to each of the neural network's parameters is calculated, and used to modify the parameters so that the objective function is either maximized or minimized. To calculate the gradient of the objective function with respect to the parameters of the neural network, the backpropagation algorithm (effectively

a highly efficient ordering of derivative calculation through the chain rule) is used. When using gradient-based optimization methods, it is important that all the functions used in the neural networks are differentiable (or at least subdifferentiable) so that the partial derivatives of the neural network's parameters can be computed for the parameter update.

Neural networks in conjunction with reinforcement learning have been found to be very effective for solving control problems, due to their computationally simple nature, flexible structure and ease of training. They are universal function approximators, meaning that they can in theory approximate any continuous function, but not discontinuous ones. This implies that they could be ill-suited to solving the complex hybrid dynamics task posed by bipedal locomotion, which are discontinuous.

1.2.4 Recurrent Neural Networks

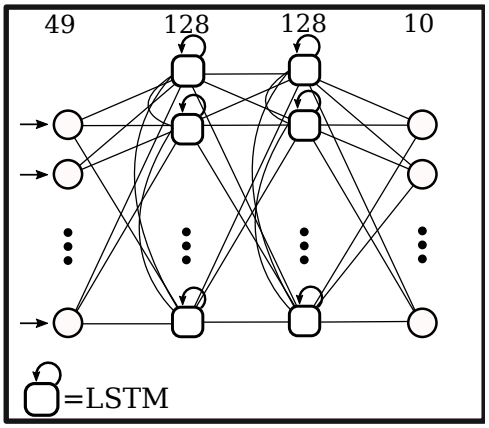

Figure 2: A diagram detailing the structure of a recurrent network. The recurrent policy, in this case an LSTM, has a memory-like functionality produced by the connections which loop back onto themselves, allowing the network to remember things from previous timesteps.

Recurrent neural networks produce outputs that are conditioned on some sort of internal memory, often assumed to be a compressed history of states. One crucial distinction between conventional neural networks and RNNs is that RNNs operate exclusively on time-series data, whereas conventional neural networks operate on individual points without regard to time. This also means that calculating the gradient requires a special case of the backpropagation algorithm known as backpropagation through time; because the state of the memory at any timestep plays a role in the output of the neural network in future timesteps, the chain rule expansion requires gradient information from the future. This means that the neural network must process an entire time-series before any gradient calculation can be done, in contrast to conventional neural networks.

Intuitively, an RNN has an awareness of the context of the state it observes at any point in time. This allows it to model or account for conditions that aren't directly observable, or learn to remember information that could be useful later. For instance, RNNs are known to

be able to approximate context-free grammars [20]. For robotics applications, this means it has the potential to perform things like implicit online system identification [15, 21].

This ability endows RNNs with (theoretical) Turing completeness. They are capable of approximating any valid computer program [22], which includes the class of discontinuous functions, conferring an obvious advantage over conventional neural networks (which have difficulty approximating discontinuous functions). For the sake of completeness, it should be noted that it appears to be very difficult to harness this theoretical potential [23]. Nonetheless, the theoretical ability of RNNs has implications for solving the discontinuous hybrid dynamics involved in bipedal locomotion. Where RNNs have been used to learn control policies, they have often achieved superior performance to feedforward networks [24, 25], with authors noting that RNNs are particularly adept at dealing with partial observability of an environment.

Another motivating factor for using RNNs to solve robotics problems is the fact that classical control methods often use some sort of memory-based mechanism for observing disturbances or aberrations from the expected behavior of the environment, like disturbance observers [7, 8]. Using an architecture that can potentially mimic these mechanisms [15, 21] would likely be of much benefit to the learned control policy, especially if the task in question is partially observable [26].

2 Methods

2.1 Distinction between Group A and Group B

At the beginning of the research, we found a bare-minimum combination of dynamics randomization and reward function terms that produced recurrent policies which consistently transferred to hardware over the course of extensive hardware trials. As research progressed, we made modifications to the dynamics randomization ranges and reward function as it became apparent what the policies were sensitive to. Though we conducted some hardware tests on the newer policies, we failed to conduct the same rigorous hardware testing we did for the original group before the COVID-19 pandemic made it impossible to do further testing. Thus, to maintain a clear separation between policies which were tested rigorously on hardware and those which were not, policies which were tested early enough in the experimentation phase to be run on hardware are denoted as belonging to Group B, while policies which were trained with a more expressive reward and more aggressive set of dynamics parameters as belonging to Group A. While some policies from Group A were tested on hardware and even outdoors, no rigorous testing was done and so no quantitative hardware results for Group A will be presented, while all quantitative training and simulation results will be for Group A.

2.2 Input Space

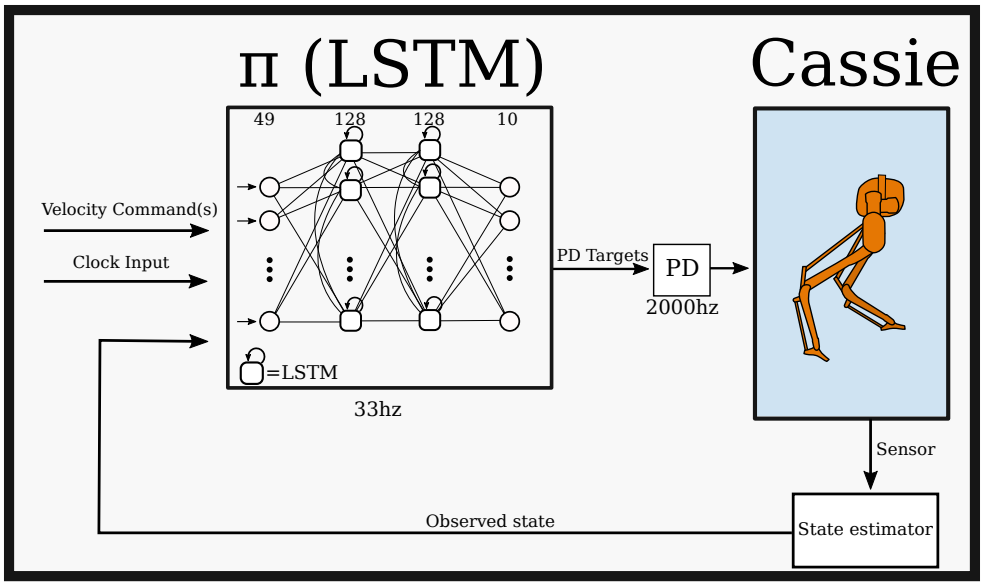

Figure 3: The control setup we use to control Cassie using a neural network. The network receives a cyclic clock input and velocity command in addition to the robot state, and outputs PD targets at 33hz. These targets are translation into motor torques at 2000hz by onboard PD controllers.

All policies received information from the built-in state estimator on the Cassie robot, including joint positions and pelvis orientation. All policies also received clock information in the form of a pair of sine and cosine inputs which correspond to the stepping frequency of a reference trajectory used to determine the reward for a given timestep.

Notable differences in the input include the addition of a desired sideways speed for Group A, as well as the removal of information about pelvis height (which was found to be a noisy and often biased estimate of the true pelvis height), as well as pelvis acceleration (which was found to be more or less inconsequential). Group A was also trained with step frequencies that varied throughout training, and so the clock input can be sped up or slowed down to achieve a desired stepping frequency.

2.2.1 Input Space of Group A

The policies in Group A received the following information from the robot's state estimator and operator commands:

$$X_t^A = \begin{cases} f_{vel} & \text{desired forward speed} \\ s_{vel} & \text{desired sideways speed} \\ \sin(\phi) & \text{clock input from gait phase} \\ \cos(\phi) & \text{clock input from gait phase} \\ \dot{\omega}, \dot{\rho} & \text{pelvis translational and rotational velocity} \\ \hat{q}, \dot{q} & \text{robot joint positions, velocities} \end{cases}$$

2.2.2 Input Space of Group B

The policies in Group B received the following information from the robot's state estimator and operator commands:

$$X_t^B = \begin{cases} f_{vel} & \text{desired forward speed} \\ \sin(\phi) & \text{clock input from gait phase} \\ \cos(\phi) & \text{clock input from gait phase} \\ h & \text{pelvis height from ground} \\ \dot{\omega}, \dot{\rho} & \text{pelvis translational and rotational velocity} \\ \ddot{\omega} & \text{pelvis translational acceleration} \\ \hat{q}, \dot{q} & \text{robot joint positions, velocities} \end{cases}$$

2.3 Action Space

The output of all policies was a symmetric set of motor PD position targets, one for each of the five motors per leg, illustrated in Figure 3. To reduce the probability of learning

suboptimal walking behavior, an offset is added to the network's outputted motor positions which makes an output of all zeros correspond to a neutral standing position.

2.4 Reward Design

Both groups of policies used a sequence of states captured from log data of a known-working controller walking forward at 1 m/s for one complete walking cycle, called a *reference trajectory*. The reward for both groups of policies was based around information obtained from this reference trajectory, similarly to Xie et al. [10]. From the reference trajectory, only the joint positions and spring positions are used, while the rest of the reward weighs factors like how well the robot is maintaining a straight-ahead orientation or whether it is matching a desired speed.

- q_{err} is an error term representing the difference in joint positions between the reference trajectory and the current joint positions.
- $\text{spring}_{\text{err}}$ represents the difference between the reference trajectory spring positions and actual spring positions.
- x_{err} is an error term denoting the difference between the forward position and desired forward position.
- $\dot{x}_{\text{err}}$ is an error term representing the difference between the desired forward velocity and actual forward velocity.
- $\dot{y}_{\text{err}}$ is an error term representing the difference between the desired sideways velocity and actual sideways velocity.
- $\text{orientation}_{\text{err}}$ which is the quaternion difference between the robot's orientation and an orientation which faces straight ahead.
- $\text{footfrc}_{\text{penalty}}$ measures the dot product of foot force and foot velocity, a term intended to ensure that large forces are applied only when the foot is stationary.
- $\text{ctrl}_{\text{penalty}}$ is a measure of the difference between the last action taken and the current action. This is intended to ensure that there are no large oscillations in applied torques which can cause shakiness on hardware.

The policies in Group A used the following reward function:

$$\begin{aligned} R = & 0.30 \cdot \exp(-\text{orient}_{\text{err}}) && +0.20 \cdot \exp(-\dot{x}_{\text{err}}) \\ & +0.20 \cdot \exp(-\dot{y}_{\text{err}}) && +0.10 \cdot \exp(-q_{\text{err}}) \\ & +0.10 \cdot \exp(-\text{footfrc}_{\text{penalty}}) && +0.05 \cdot \exp(-\text{spring}_{\text{err}}) \\ & +0.05 \cdot \exp(-\text{ctrl}_{\text{penalty}}) \end{aligned} \tag{1}$$

The policies in Group B used the following reward function:

$$\begin{aligned} R = &0.30 \cdot \exp(-\text{orient}_{\text{err}}) \quad +0.20 \cdot \exp(-q_{\text{err}}) \\ &+0.20 \cdot \exp(-\dot{x}_{\text{err}}) \quad +0.20 \cdot \exp(-\dot{y}_{\text{err}}) \\ &+0.05 \cdot \exp(-x_{\text{err}}) \quad +0.05 \cdot \exp(-\text{spring}_{\text{err}}) \end{aligned} \tag{2}$$

Since Group A was a later iteration of Group B, several insights taken from the earlier group were used to inform the reward function of Group A. The weighting of q_{err} (joint positional error) was reduced as it was found that allowing the policy to adhere less strictly to the reference motion allowed for more robust behaviors. A term representing the position of the robot (x_{err}) was also removed from the reward function after it was determined to be redundant when combined with the robot velocity. A sideways input reward term was added to allow the policy to learn sidestepping or sideways velocity matching behaviors.

2.5 Recurrent Proximal Policy Optimization

Proximal Policy Optimization (PPO) [13] is a popular model-free reinforcement learning algorithm. It has been used to train neural networks to control policies in the past [10] to great success. Several variants of PPO exist; in this work, the algorithm used features early-stopping once Kullback–Leibler divergence between the original policy and the updated policy grows past some threshold. Additionally, before training commences, the mean and standard deviation of the states is computed through random sampling and used to normalize states collected during training.

An additional modification to the algorithm must be made due to the fact that computing the gradient of an RNN necessitates the backpropagation through time (BPTT) algorithm. Normally, a batch of transitions is sampled from a so-called replay buffer, which stores a collection of transitions (where a transition is the state, action, reward, and critic value at a given timestep) from the simulated environment, and used to inform the gradient calculation for the policy. However, in the case of an RNN, a batch of trajectories of timesteps must be sampled due to the need to calculate a gradient over time. This is described by Sutton and Barto [11] as *trajectory sampling*.

2.6 Simulation

2.6.1 Dynamics Randomization

Ranges were chosen for each dynamics parameter based on whether or not robot behaviors seemed sensitive to that particular parameter. For the policies belonging to Group A, a set of 77 dynamics parameters are randomized, a description of which can be found in Table 1. The choice for the ranges and parameters themselves in Group A was informed by results obtained in Group B and trial-and-error discovery of what did and did not influence robustness on hardware. The policies belonging to Group B were trained with 61

randomized dynamics parameters. Descriptions of these parameters can be found in Table 2.

Parameter	Unit	Range
Joint damping	Nms/rad	$[0.2, 6.0] \times$ default values
Joint mass	kg	$[0.2, 2.0] \times$ default values
Friction coef. (translational)	-	$[0.2, 1.4]$
Ground slope (pitch)	degrees	$[-4.6, 4.6]$
Ground slope (roll)	degrees	$[-4.6, 4.6]$
Joint encoder bias	degrees	$[-0.6, 0.6]$
Simulation delay	milliseconds	$[30, 36]$

Table 1: Ranges for dynamics parameters in Group A. Random biases are added to the joint positions reported by the encoders to account for any calibration error, and incline or decline the ground by several degrees so that resulting policies are more robust to slopes in the real world. Also aggressively randomized are damping and ground friction, to make policies robust to wear and tear on the robot and slippery or grippy surfaces.

Parameter	Unit	Range
Joint damping	Nms/rad	$[0.5, 1.5] \times$ default values
Joint mass	kg	$[0.7, 1.3] \times$ default values
Pelvis CoM (x)	cm	$[-25, 6]$ from origin
Pelvis CoM (y)	cm	$[-7, 7]$ from origin
Pelvis CoM (z)	cm	$[-4, 4]$ from origin

Table 2: Ranges for dynamics parameters in Group B. Initially, the pelvis center of mass was aggressively randomized because it seemed apparent that the robot's center of mass in simulation differed significantly from the center of mass on hardware, though this turned out later to be unlikely.

2.6.2 Simulation Robustness Test

Six types of policy from Group A were evaluated on their robustness to a variety of dynamics disturbances in simulation; LSTM networks trained with and without dynamics randomization, and conventional (feedforward) neural networks trained with and without dynamics randomization. The average time before falling down was recorded for each type of policy, and experiments that ran for more than 40 seconds were cut short for computational efficiency.

2.6.3 Parameter Inference

To perform online system-ID, the memory of the recurrent networks is treated as a latent encoding and used to train a decoder network to transform the memory back into the original dynamics parameters. The decoder networks are conventional feedforward networks taking the policy's memory as input, each with three hidden layers of sizes 256,

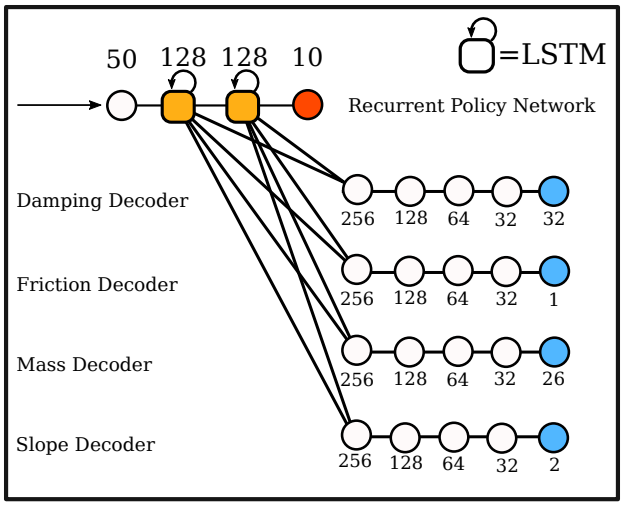

Figure 4: Two hidden LSTM layers (in orange) of a recurrent policy network form inputs to four decoder networks, which predict various dynamics parameters (predictions in blue).

128, and 64, and an output layer of size equal to that specific dynamics parameter. A diagram of the architecture of the policy and decoder networks can be seen in Fig. 4.

Training was done under a supervised learning paradigm, where 60,000 memory-label pairs were generated using the simulated Cassie environment. At a randomly chosen point during a simulated rollout, the policy networks' memory was sampled and stored along with the current dynamics parameter values to create the dataset. Afterwards, simple supervised learning was conducted with a split of 48,000 pairs in the training set and 12,000 pairs in a testing set using MSE loss and the Adam optimizer with a learning rate of $5 \cdot 10^{-5}$.

2.6.4 Principal Component Analysis

Interesting patterns emerge when principal component analysis is performed on the memory of the recurrent policies by collecting the state of the memory over the course of one rollout, then projecting each timestep's memory into a 2-dimensional or 3-dimensional point using PCA and visualizing the resulting scatterplot.

An example of the resulting curve can be seen in Fig. 5. I further explore the effects of dynamics randomization on the PCA projections by performing the analysis over multiple translational friction values, including some outside the range that policies were trained on. These values range from approximately the friction of wet concrete to the approximate slipperiness of slick ice and visualize the effect that these changes in the simulation dynamics have on the effect of the policy's PCA-projected memory. Also compared are PCA projections between memory taken from a recurrent policy trained with a clock input, and a PCA projection of recurrent policy memory trained with no clock input.

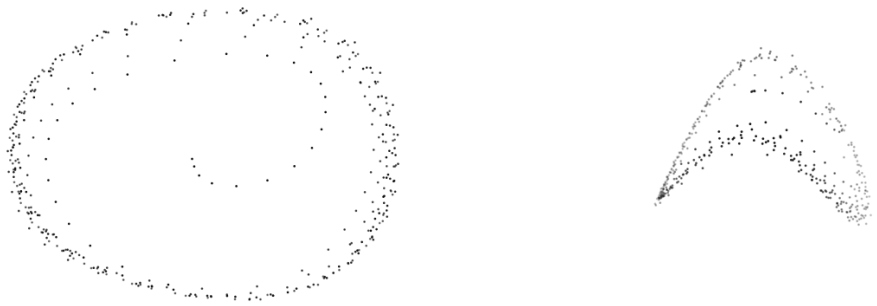

Figure 5: A plot of the 2-dimensional (left) and 3-dimensional (right) PCA-reduced state of a recurrent policy network's memory over the course of one rollout consisting of several gait cycles. In the 2D projection, points become lighter as a function of time. In the 3D projection, points become lighter as a function of depth.

2.7 Treadmill Test

The policies in Group B were evaluated on their ability to walk upright before falling over in the real world on a treadmill at a speed of 1 m/s. Experiments that ran for more than 40 seconds were cut off.

3 Results

3.1 Simulation Robustness

Recurrent LSTM networks and simple feedforward networks both with and without dynamics randomization were trained and compared; a graph of their reward curves when trained without dynamics randomization is shown in figure 6. LSTM networks achieved the highest reward, while feedforward networks failed to reach similar performance.

Also conducted was a simulation robustness test, which measured the time in seconds a policy was able to remain upright and walking forward at 1 m/s subject to the dynamics encountered under the set of dynamics parameters μ_i^A, sampled from the ranges for Group A described in Table 1. LSTM networks trained with dynamics randomization did not fall down under any of the conditions presented by a dynamics parameter μ_i^A, while every other group fell at least once or suffered from degraded performance. Despite achieving a higher reward as observed in Table 6, LSTM policies actually performed slightly worse than FF policies during the simulated robustness test. This is likely due to some sort of overfitting, as the computational capacity of the LSTM networks allows it to overfit to quirks of the simulator or static dynamics parameters.

The LSTM DR policies in Group A exhibited an encouraging range of robust behaviors.

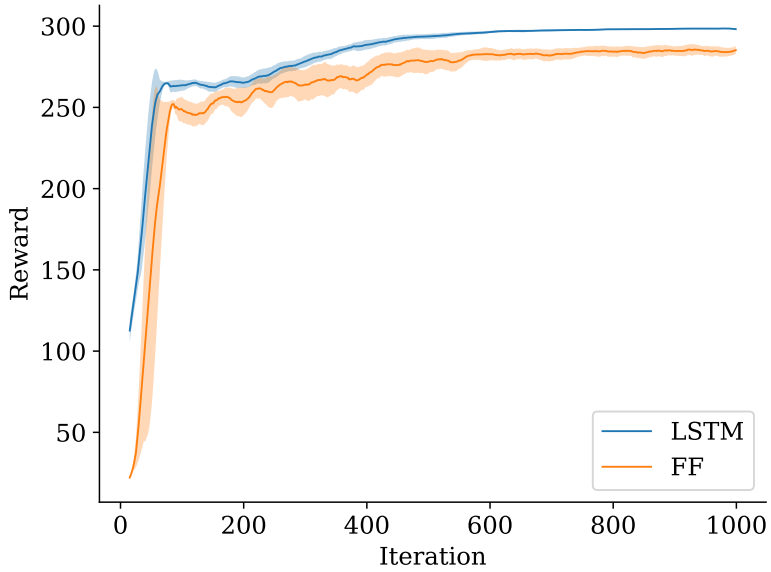

Figure 6: Averaged reward curves of LSTM and feedforward policies trained on the Cassie simulator without dynamics randomization. LSTM networks clearly outperform FF networks, but fail to do better on hardware as seen in Table 5 and do not outperform FF networks on a simulation robustness test as seen in Table 3. This could be because LSTM networks are more adept at overfitting to the simulation dynamics.

Parameter Set	LSTM DR	LSTM	FF DR	FF
μ_1^A	10s	1s	1s	1s
μ_2^A	40s	27s	33s	40s
μ_3^A	40s	1s	34s	1s
μ_4^A	40s	1s	40s	1s
μ_5^A	40s	1s	34s	2s
μ_6^A	40s	4s	40s	17s
μ_7^A	40s	1s	29s	3s
μ_8^A	40s	25s	24s	11s
μ_9^A	40s	9s	26s	1s
μ_{10}^A	40s	3s	15s	3s
Avg.	**37.0**	**7.4**	**27.6**	**8.1**

Table 3: Time (in seconds) that a policy resulting from a seed was able to walk in simulation subject to the dynamics imposed by μ_i^A. Each μ_i^A is a collection of 77 parameters sampled from the ranges described in Table 1.

The policies were able to walk with a varying stepping frequency, and at speeds of up to 1 m/s. Some of these behaviors, such as sidestepping, were demonstrated previously in the literature by Xie et al. [10], though each behavior was learned by distinct policies; e.g., one for sidestepping and one for walking forward. The policies presented here represent,

for the first time ever, learned controllers that are capable of walking at variable and even backwards speeds, sidestepping or strafing, and changing the stepping frequency on the fly.

3.2 Memory Introspection

3.2.1 Parameter Inference

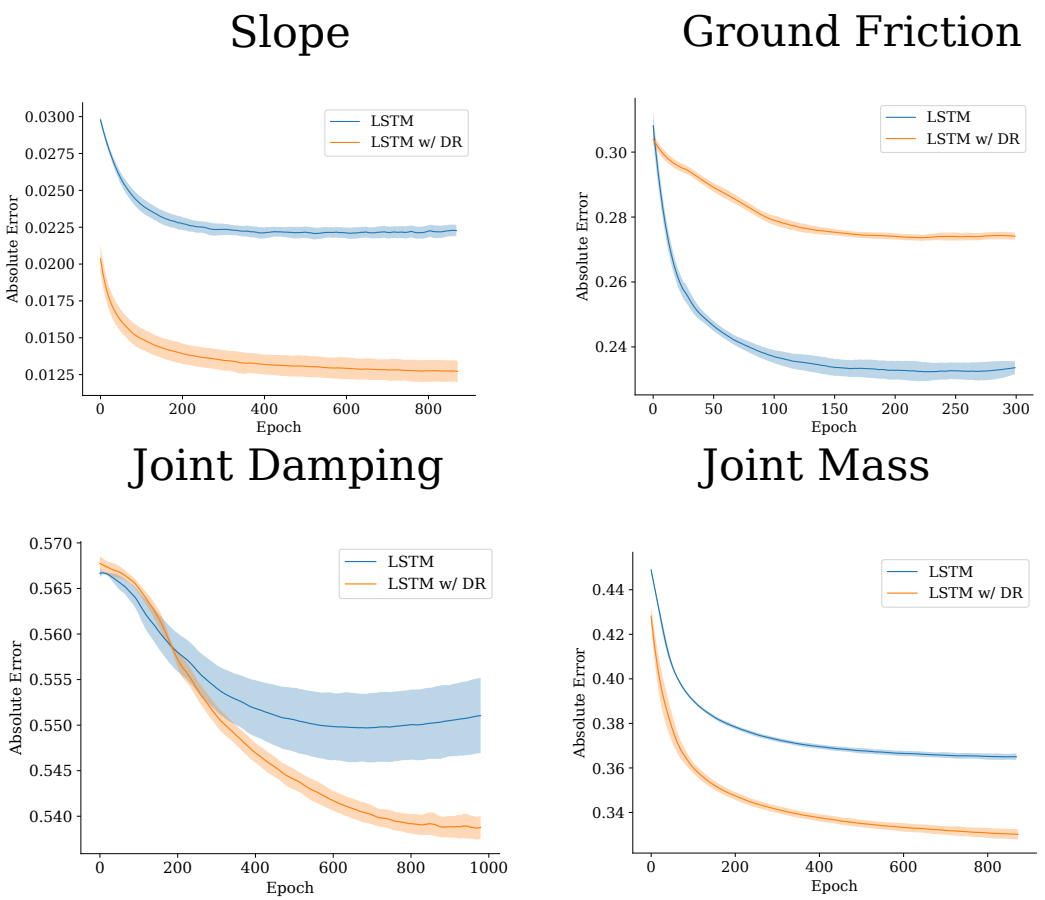

Figure 7: Absolute error on the test set over the course of training for the four decoder networks, each trying to predict a dynamics parameter using only a snapshot of memory sampled randomly from a policy rollout. Predictions conditioned on memory sampled from networks trained with dynamics randomization are labeled LSTM w/ DR, and those sampled from networks without dynamics randomization are labeled LSTM.

We compare dynamics parameter predictions from decoder networks trained to output dynamics parameter estimates conditioned on memory sampled from the recurrent policies trained with and without dynamics randomization. In general, the dynamics parameter predictions conditioned on memory sampled from networks trained with dynamics randomization outperformed predictions conditioned on memory sampled from networks trained

without dynamics randomization, with the exception of ground friction, as can be seen in Fig. 7. In addition, table 4 lists the percent improvement for each dynamics quantity, where the percent improvement is a quantity representing how much better the decoder networks trained on LSTM w/ DR memory did than the decoder networks trained on LSTM w/o DR. We use the performance of decoder networks trained on LSTM w/o DR as a baseline, since these networks did not encounter random dynamics parameters during training and so could not have learned to encode these quantities, meaning that any decoding of those memory states is simply the decoder networks using state history to make educated guesses about possible dynamics values.

Dynamics Parameter	**Percent Improvement**
Damping	0.13%
Friction	-20%
Mass	10%
Slope	51%

Table 4: Percent improvement of decoder networks conditioned on memory from LSTMs trained with DR over decoder networks conditioned on memory from LSTMs not trained with DR.

One possible (counterintuitive) reason for the friction discrepancy could be that LSTM networks trained with dynamics randomization simply choose a stepping pattern which is robust to a wide range of ground frictions rather than inferring and accounting for varying ground friction. If this is the case, then it would be difficult to observe any sort of difference in the hidden states of the LSTM, as it is robust enough to friction that it does not need to invest any effort into changing its behavior or memory to account for it. Conversely, the LSTM networks trained without ground friction would be unable to be robust to the varying ground friction, and this would have a presumably large effect on not just the actions taken by the network, but also the internal state of the network. If these effects are simple enough, they could be easy for the decoder networks to use to inform a ground friction prediction. The reproducibility of this result and robustness of LSTM policies trained with DR to ground friction seems to imply that this is the case.

In the other curves, memory sampled from LSTM policies trained with dynamics randomization helps the decoder network learn faster and also settle at a lower minimum, implying that these memories contain more information about the dynamics disturbances.

3.2.2 Principal Component Analysis

Interesting patterns emerge when dimension reduction through PCA is performed on the memory of the recurrent policy networks as it changes over the course of a rollout. As can be observed in Figures 5, 8, and 9, the two-dimensional PCA projection of the memory is a circular shape, analogous to a central pattern generator [27]. Interestingly, all of the recurrent policies have a similarly elliptic 2-dimensional PCA reduced memory representation. The regularity of the shape of the reduced dimensional projection of the memory may indicate the importance of the clock in achieving a stable walking cycle.

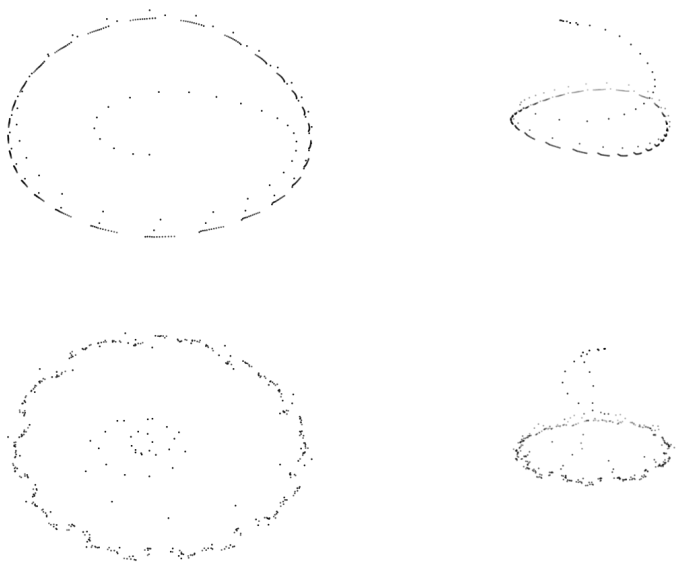

Figure 8: 2-dimensional (left) and 3-dimensional (right) PCA projections over the course of a rollout taken from memory of a policy trained with a clock input (top) and without a clock input (bottom).

In fact, policies not trained with a clock input appear to be missing the distinct 28-part segmentation that can be observed in policies trained with clock input (as can be seen in Fig. 8), and overall the projection is more stochastic in appearance; this is further evidence for the importance of the clock input, which appears to play a role in increasing the amount of variance in the memory which can be captured in a low-dimensional PCA projection. The saddle-like shape of the 3-dimensional projection is also extremely regular and appears in every recurrent policy evaluated, though it varies in steepness of shape from policy to policy. Notable in every projection is the apparent segmentation of the ellipsis, which features 28 distinct segments in keeping with the gait phase.

The shape of the ellipses in the reduced-dimension projection appears to decohere as the dynamics approach values not seen in training. This can be seen in Fig. 9 as the ellipses appear to fray at the edges and eventually lose their shape as a function of decreasing ground friction. The slipperiest ground friction visualized (0.25) is roughly equivalent to walking on slick ice. The PCA projection of the memory of the policy trained with DR more or less maintains its shape throughout the rollouts with low friction, while the projection from the policy not trained with DR is unable to find a consistent shape.

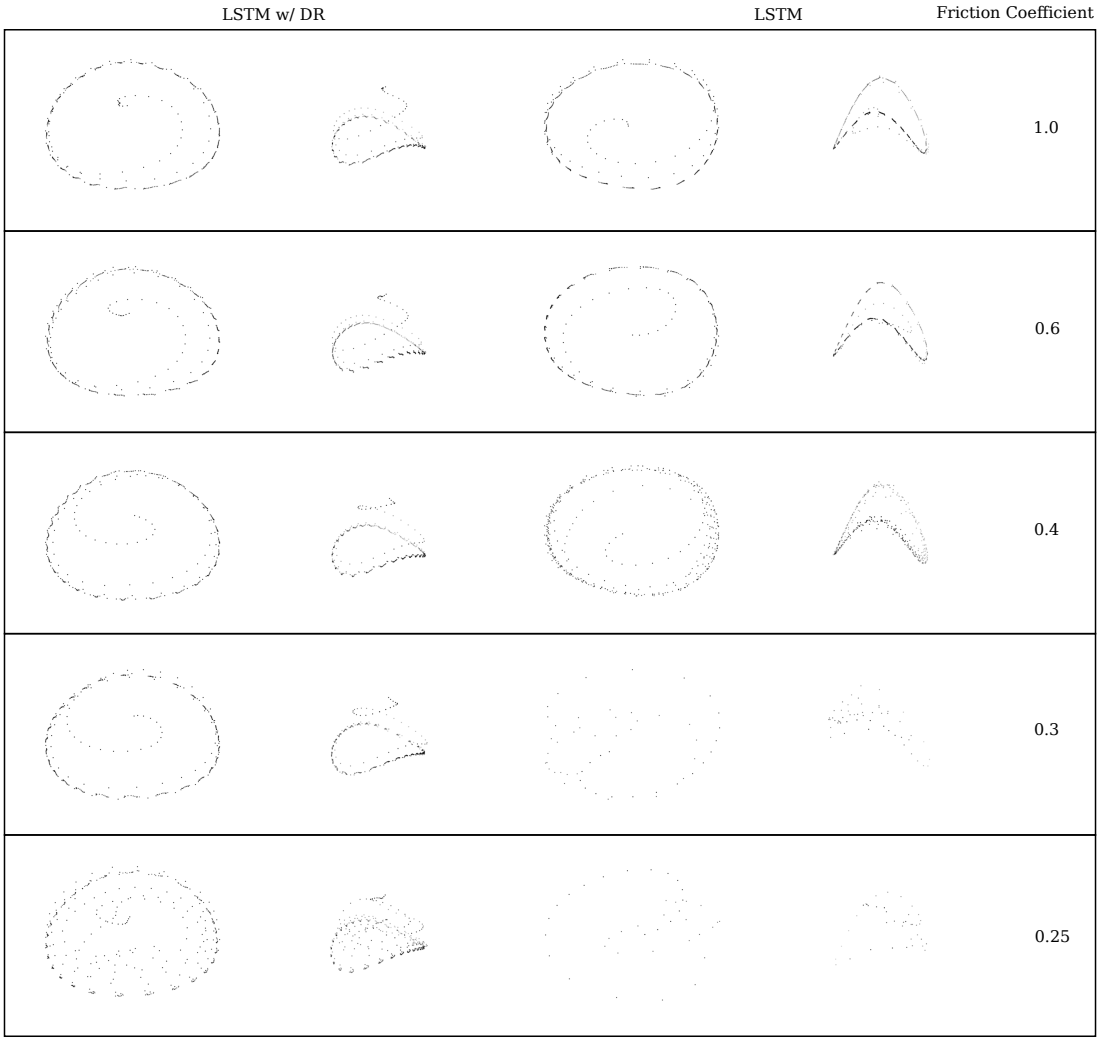

Figure 9: 2-dimensional and 3-dimensional PCA projections of a policy network trained with DR and without DR, over the course of several rollouts with varying ground friction coefficients.

3.3 Hardware Results

3.3.1 Treadmill Test

Before the Covid-19 pandemic, the policies in Group B were evaluated on a treadmill at a speed of 1 m/s, and the time walking upright before falling over was recorded. Policies that were able to walk for more than 40 seconds were cut short. The results can be seen in Table 5. None of the LSTM policies trained with dynamics randomization fell over before the end of the 40 second experiment, while only two of the LSTM policies trained without dynamics randomization were able to walk for more than 40 seconds. Feedforward policies performed roughly equivalently to LSTM policies when not trained with dynamics randomization despite achieving lower reward in simulation, which I attribute to the LSTM overfitting the simulation dynamics, much like the simulation robustness test.

Seed	LSTM^B	LSTM DR^B	FF^B
1	7s	$>$ 40s	0s
2	7s	$>$ 40s	16s
3	$>$ 40s	$>$ 40s	$> 40s$
4	11s	$>$ 40s	9s
5	3s	$>$ 40s	33s
6	3s	$>$ 40s	0s
7	2s	$>$ 40s	5s
8	0s	$>$ 40s	3s
9	$>$ 40s	$>$ 40s	7s
10	4s	$>$ 40s	10s
Avg.	**11.7s**	**40s**	**12.3s**

Table 5: Time (in seconds) that a policy resulting from a seed was able to walk in the real world. Experiments that ran for more than 40 seconds were cut off.

Feedforward policies trained with dynamics randomization were not able to walk due to unsafe oscillations in the output, and so only LSTM, LSTM DR, and FF were evaluated on hardware. Though several policies from Group A were tested on the treadmill, no comprehensive or rigorous experiments were done and so quantitative hardware results only for the older Group B are presented.

All policies, whether trained with dynamics randomization or not, exhibited a curious 'falling backward' behavior when policies were commanded to step in place, marked by at first flat-foot walking, then heel-walking and slowly drifting backwards, and finally falling over backwards. The randomization of pelvis center of mass was a proposed solution which sought to address a hypothesized cause of this behavior, which we initially thought to be a discrepancy between the simulation robot center of mass and true hardware center of mass, since this behavior was not present in simulation. Randomizing pelvis center of mass greatly improved the robustness of policies to falling backwards (policies in some cases were able to walk in place for up to thirty seconds), but the issue persisted regardless of the range of center of mass.

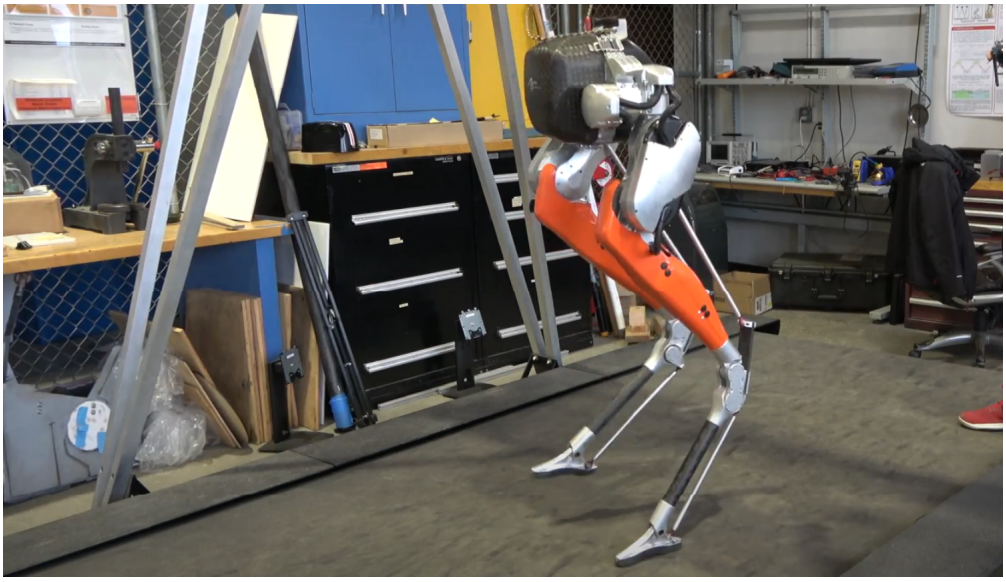

Figure 10: A recurrent policy being evaluated on the physical Cassie robot.

Walking forward was not an issue, and recurrent policies trained with dynamics randomization were generally able to walk forward while even some recurrent policies not trained with dynamics randomization were able to complete the walking forward task. This could possibly be because walking at speed is more dynamically stable and easier than stepping in place. The training regimen for Group A was to a large degree informed by attempts to stamp out the falling backwards behavior observed in Group B.

4 Conclusion

In this work, it has been shown that recurrent policy networks are a compelling choice of agent for learning robotic bipedal locomotion due to their high level of robustness and the fact that they are theoretically more sound for complicated, nonlinear and discontinuous problems like bipedal locomotion. They are capable of learning an wide range of all-in-one expressive behaviors which have not yet been demonstrated using a single learned controller. This bodes well for various other challenging robotics problems suffering from partial observability which can be solved by learned controllers. Additionally, the resulting shape of the PCA-projected memory over time heavily implies that some sort of clock input is important to achieving stable, robust walking behavior analogous to central pattern penerators. Furthermore, the ability of learned, memory-based controllers to embed indirectly observed information such as parameters of the dynamics (though the information can currently only be retrieved in a lossy way), has consequences for all sim-to-real tasks. One can imagine using these networks to collect data and infer information about the real world, and use the inferred information to inform values in simulation; or, at the very least, using them to account for disturbances to the expected physics dynamics in real-time.

References

[1] Miomir Vukobratovic and Branislav Borovac. Zero-Moment Point - Thirty Five Years of its Life. *I. J. Humanoid Robotics*, 1:157–173, 03 2004. doi: 10.1142/S0219843604000083. URL https://www.worldscientific.com/doi/abs/10.1142/S0219843604000083.

[2] Y. Haikawa K. Hirai, M. Hirose and T. Takenaka. The development of honda humanoid robot. In *1998 IEEE International Conference on Robotics and Automation*, volume 2, pages 1321–1326, May 1998.

[3] Robin Deits Kanako Miura Russ Tedrake, Scott Kuindersma. A closed-form solution for real-time zmp gait generation and feedback stabilization. In *2015 IEEE-RAS International Conference on Humanoid Robots*, volume 2, 2015.

[4] Eric R Westervelt, Jessy W Grizzle, and Daniel E Koditschek. Hybrid zero dynamics of planar biped walkers. *IEEE transactions on automatic control*, 48(1):42–56, 2003. URL https://ieeexplore.ieee.org/document/1166523.

[5] Salman Faraji, Soha Pouya, Christopher G Atkeson, and Auke Jan Ijspeert. Versatile and robust 3d walking with a simulated humanoid robot (atlas): A model predictive control approach. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1943–1950. IEEE, 2014.

[6] Siavash Rezazadeh, Christian Hubicki, Mikhail Jones, Andrew Peekema, Johnathan Van Why, Andy Abate, and Jonathan Hurst. Spring-Mass Walking With ATRIAS in 3D: Robust Gait Control Spanning Zero to 4.3 KPH on a Heavily Underactuated Bipedal Robot. Dynamic Systems and Control Conference, 10 2015. doi: 10.1115/DSCC2015-9899. URL https://doi.org/10.1115/DSCC2015-9899. V001T04A003.

[7] Mingon Kim, Jung Hoon Kim, Sanghyun Kim, Jaehoon Sim, and Jaeheung Park. Disturbance observer based linear feedback controller for compliant motion of humanoid robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 403–410. IEEE, 2018. URL https://ieeexplore.ieee.org/document/8460618.

[8] Nicholas Paine, Joshua S Mehling, James Holley, Nicolaus A Radford, Gwendolyn Johnson, Chien-Liang Fok, and Luis Sentis. Actuator control for the NASA-JSC valkyrie humanoid robot: A decoupled dynamics approach for torque control of series elastic robots. *Journal of Field Robotics*, 32(3):378–396, 2015. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21556.

[9] Taylor Apgar, Patrick Clary, Kevin Green, Alan Fern, and Jonathan W. Hurst. Fast online trajectory optimization for the bipedal robot cassie. In *Robotics: Science and Systems*, 2018.

[10] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan Hurst, and Michiel van de Panne. Learning Locomotion Skills for Cassie: Iterative Design and Sim-to-Real. In *3rd Conference on Robotic Learning (CORL)*, 2019. URL https://zhaomingxie.github.io/publications/corl_2019.pdf.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[12] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io/lil-log*, 2018. URL https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[14] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[15] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, May 2018. doi: 10.1109/ICRA.2018.8460528. URL https://ieeexplore.ieee.org/document/8460528.

[16] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's Cube with a Robot Hand. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1910.07113.

[17] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. doi: 10.1177/0278364919887447. URL https://doi.org/10.1177/0278364919887447.

[18] 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. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.010. URL http://www.roboticsproceedings.org/rss14/p10.html.

[19] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[20] Qinglong Wang, Kaixuan Zhang, Alexander G. Ororbia II, Xinyu Xing, Xue Liu, and C. Lee Giles. A comparison of rule extraction for different recurrent neural network models and grammatical complexity. *CoRR*, abs/1801.05420, 2018. URL http://arxiv.org/abs/1801.05420.

[21] Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *CoRR*, abs/1512.04455, 2015. URL http://arxiv.org/abs/1512.04455.

[22] Heikki Hyotyniemi. Turing Machines are Recurrent Neural Networks, 1996. URL http://users.ics.aalto.fi/tho/stes/step96/hyotyniemi1/.

[23] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the Turing Completeness of Modern Neural Network Architectures. In *International Conference on Learning Representations*, 2019. URL https://iclr.cc/Conferences/2019/Schedule?showEvent=707.

[24] A. Singla, S. Padakandla, and S. Bhatnagar. Memory-Based Deep Reinforcement Learning for Obstacle Avoidance in UAV With Limited Environment Knowledge. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12, 2019. ISSN 1558-0016. doi: 10.1109/TITS.2019.2954952. URL https://ieeexplore.ieee.org/document/8917687.

[25] Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*, November 2015. URL http://www.cs.utexas.edu/~pstone/Papers/bib2html/b2hd-SDMIA15-Hausknecht.html.

[26] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010. URL https://academic.oup.com/jigpal/article/18/5/620/751594.

[27] Scott L Hooper. Central pattern generators. *Embryonic ELS*, 2001.