Learning Locomotion Policies for Tensegrity Robots

Brian Cera[1], Edward L. Zhu[1]

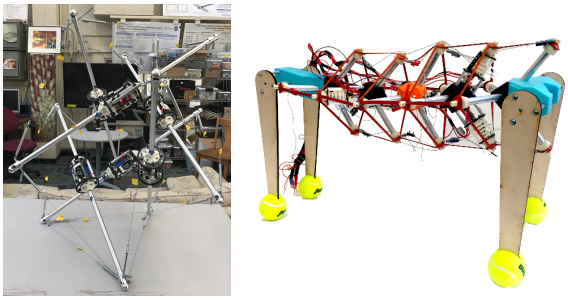

Fig. 1: The TT-5 spherical tensegrity robot (left) and Laika quadruped robot (right) from the Berkeley Emergent Space Tensegrities Lab

I. Introduction

Tensegrity structures and their application in the field of robotics has been a growing research area. This is largely due to the advantageous mechanical properties that they exhibit and how those properties translate into system-level robustness. Specifically, they are lightweight, compliant, and resistant to impact, making them an attractive candidate for applications in contact-rich environments [1].

Comprised of rigid rods suspended in a network of tensioned cables, these structures are free of the force amplifying lever arms commonly seen in traditional "hard" robots. In fact, it can be shown that in an ideal class-1 tensegrity system with isolated rigid elements (such as the 6-bar spherical tensegrity robot shown in Fig. 1), these rigid elements experience only axial loading and no bending moments [2]. Due to the aforementioned isolation of rigid elements, external disturbances acting upon the system are distributed globally throughout the tension network as opposed to being subject to magnification by lever arms and concentrated at joints [1]. Leveraging these advantages, tensegrity structures are being used as the basis for mobile robots capable of performing locomotion tasks.

However, the same characteristics which make tensegrity structures mechanically desirable, also add much complexity to the control of such systems due to their high dimensional state and action spaces. Thus, while simple, hand-engineered control schemes do exist [3], [4], they leave much to be desired in terms of performance and generalizability. On the other hand, recent work has seen model-based deep reinforcement learning (RL) techniques being applied to high dimensional complex control problems with much success [5], [6]. This motivates the use of model-based RL techniques to learn the gaits required for tensegrity robots to locomote effectively and efficiently.

Despite the high dimensionality of tensegrity robots, they can be accurately modeled using rigid body dynamics and models do exist in the form of either equations of motion derived through first principles [7], [8], or rigid-body physics simulators such as the Bullet based NASA Tensegrity Robotics Toolkit (NTRT) [1]. This motivates the use of model-based RL which has shown much better sample efficiency compared to model-free techniques. This is due to the fact that system models can be used in trajectory optimization to generate optimal state-action sequences which can provide efficient supervision during the training of the global neural network policy π_θ. As summarized in [5], trajectory optimization is commonly done offline as its computational cost is often prohibitive of real-time execution. For our work, the training time, which is on the order of hours, is dominated by the collection of sample trajectories and the subsequent optimization as opposed to the actual training of the neural network policy.

In this report, we present work on the learning of global locomotion policies for two tensegrity-based systems using model-based RL techniques. Specifically, we combine trajectory optimization with supervised learning to obtain neural network global policies. The first robot is a bio-inspired quadruped with tensegrity spine as seen in the right image of Fig. 1. In contrast to traditional rigid quadrupedal robots, this robot features a compliant body which can also be used for actuation by controlling the length of the cables. We hope to demonstrate that despite using simple 1 DoF legs, the robot can still perform locomotion by leveraging the flexibility of the spine. As is typical with tensegrity systems, this robot features high dimensional state and action spaces, which are of size 140 and 36 respectively. The state vector is comprised of the 3-D pose and velocities of the 9 rigid bodies (hips, shoulders, 3 vertebrae, and 4 legs) as well as the 32 cable control lengths. The action vector includes the 32 cable control lengths and torques to the 4 legs. The second robot is a 6-bar spherical tensegrity robot as seen in the left image of Fig. 1, which is able to change its shape by actuating the cables in its tension network. This robot has a 60-dimensional state space (rotations and angular velocities about the longitudinal axis of each rod are ignored) and a 24-dimensional action space of the control lengths of each cable.

*Research supported by NASA Early Stage Innovation grant NNX15AD74G

[1]Authors are with the University of California, Berkeley, Mechanical Engineering Department, CA, 94720, USA Contact email: brianmcera@berkeley.edu

[1]https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/

For the quadruped, we propose to use the guided policy search algorithm, which has achieved good results in recent work on a 6-bar tensegrity robot [9] where a fast locomotion policy was attained. However, as will be discussed later, we were not able to arrive at a meaningful policy possibly due to a poorly formulated cost function which could not properly indicate locomotion success in the context of this system. As for the 6-bar robot, we propose to use imitation learning with a model predictive control expert and were able to achieve stable locomotion.

In addition, we investigate the domain transfer of successful learned policies on the 6-bar robot to improve their robustness and applicability to uncertain systems. Specifically, we implemented a method which perturbs the model parameters during the generation of optimal trajectories. This is important when training in simulation using model-based RL methods where it is assumed that there is no model mismatch between the source and target domains. Obviously it is unlikely that this assumption actually holds, especially when the goal is to transfer a learned policy from simulation to real hardware. In this work we do not run our policies on real hardware, but simulate model mismatch to evaluate the success of policy transfer.

II. Locomotion for a Quadruped Robot with Tensegrity Spine

In this section we present the formulation for learning locomotion for the quadruped robot with tensegrity spine. As we were unable to attain a meaningful policy, reasons for the failure will be discussed.

A. Simulation Environment and Dynamics

Similar to [9], the model of the robot was built in the NTRT simulation environment and interacted with the learner using a ROS network. Functionality was built into the NTRT model to allow for single or multi-stepping of the physics simulation as well as resetting the robot to its initial condition. In order to allow for randomness in the initial state, the simulation was set to run a fixed number of steps upon reset and a randomly sampled action could be applied during the duration of this reset to effectively achieve a random starting condition. In our work, we chose this duration to be 500 steps, which corresponded to 0.5s with a simulation timestep of $\Delta T = 0.001$s. This also allowed the robot to settle as it is initialized slightly above the ground plane to avoid clipping of the geometry.

In addition, the cable control length inputs were mapped from the neural network outputs to valid cable length commands. In NTRT, actuated cables are modeled as a compliant section and a stiff section connected in series. Thus, as opposed to directly controlling the torque of an imaginary motor (or equivalently the tension in the cable), which would be difficult due to the global distribution of any local disturbances, the cable is controlled via the length of its stiff section (equivalent to spooling the cable in or out). This would allow the unconnected end of the compliant section to deflect freely in response to changes in tension within the cable network. As the cables control lengths were not allowed to take on negative values, the first 32 outputs of the global policy $\pi_\theta(u_t|x_t)$ corresponding to the cable control lengths needed to be remapped to valid inputs (the other 4 correspond to the torque being applied to the legs). This was done using the equation

$$\bar{u}_t^{1:32} = x_t^{109:140} + f(u_t^{1:32}) v_{max} \Delta T \tag{1}$$

Where the superscript indicates the subset of inputs and states that correspond to the cable control lengths and f is the sigmoid function scaled to be between [-1,1]. $\bar{u}_t$ is then applied to the model. Here v_{max} is the maximum velocity of the simulated actuator.

B. Guided Policy Search

Guided policy search algorithms seek to optimize a global policy $\pi_\theta(u_t|x_t)$ under an expected cost function which can be written as $J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} l(x_t, u_t)]$. This is done by training over trajectory samples which have been optimized using some local policy $p(u_t|x_t)$ while constraining the KL-divergence between local and global policies. The intuition here is that we use a teacher which exhibits locally optimal behavior (local policies) for the student (global policy) to learn from. In addition, the teacher tries to maintain a similar distribution of trajectories to the student in order to provide a good training set and to keep the learning process from diverging.

During local trajectory optimization, the system dynamics are required, but not known explicitly, so they can be attained by fitting models to the sampled trajectories. In our formulation, which is based on [10] and [11], we model the local dynamics as a time-varying linear Gaussian using linear regression over each sample trajectory $i \in [1, N]$. This results in dynamics of the form $p_i(x_{t+1}|x_t, u_t) \sim \mathcal{N}(A_t x_t + B_t u_t + c_t, F_t)$. In addition we use time-varying linear Gaussian controllers of the form $p_i(u_t|x_t) \sim \mathcal{N}(K_t x_t + k_t, C_t)$. Once the aforementioned dynamics have been fitted, we can solve for the controller gains using dynamic programming (DP), as is done in the LQR method [12]. Specifically, we solve the following constrained optimization problem for each sampled trajectory where P represents the space of all LQG controllers parametrized by K_t, k_t, and C_t, and $p_i(\tau)$ represents the distribution of trajectories resulting from the linear Gaussian controllers and fitted dynamics.

$$\begin{aligned} p_i(u_t|x_t) = & \underset{p(u_t|x_t)\in P}{\operatorname{argmin}} \sum_{t=1}^{T} \mathbb{E}_{p_i(\tau)}[l_i(x_t, u_t)] \\ & \text{s.t.} \quad \mathcal{D}_{KL}(p_i(\tau) \| \pi_\theta) \leq \epsilon \end{aligned} \tag{2}$$

This form of the optimization problem corresponds to a variation of GPS called mirror descent guided policy search (MDGPS) where instead of constraining each local policy to be similar to the previous one, we constrain the local policy against the global policy directly [10]. In order to use the LQR method here, we can rewrite the problem as its Lagrangian below.

$$\mathcal{L}(p_i(\tau), \eta) = \sum_{t=1}^{T} \mathbb{E}_{p_i(\tau)}[l_i(x_t, u_t)] + \eta(\mathcal{D}_{KL}(p_i(\tau) \| \pi_\theta) - \epsilon) \tag{3}$$

Now, we use dual gradient descent to alternate between minimizing the Lagrangian with respect to $p_i(\tau)$ by taking the fitted linear dynamics and the second order Taylor series (quadratic) expansion of the Lagrangian to solve for the controller gains and adjusting the dual variable η based on the amount of constraint violation. We do this until the KL-divergence is within 10% of ϵ [11]. Finally, a forward pass using the final controller gains is performed on each element of the sampled trajectories to provide the locally optimal state-action pairs. These will then be used as supervision for training the global neural network policy $\pi_\theta(u_t|x_t)$, which can be viewed as an approximate method for minimizing the KL-divergence between the global and local policies. The MDGPS algorithm used in this work is shown in Algorithm 1.

Algorithm 1 The MDGPS Algorithm

for iteration $k = 1 \dots K$ **do**
  Sample N trajectories $\{\tau_i\}$ using π_θ
  for $i = 1 \dots N$ **do**
    Fit model $p_i(x_{t+1}|x_t, u_t)$ on τ_i
    Solve for controller $p_i(u_t|x_t)$
    Do forward pass to get optimal action u_t^* for each x_t in τ_i
  end for
  Train π_θ using optimal state action pairs $\{x_t, u_t^*\}$
end for

In our work, we chose to run MDGPS for 12 iterations, and collected 25 sample trajectories per iteration using on-policy sampling (off-policy sampling using the local policies is valid as well, but has been shown to underperform compared to on-policy sampling [10]). Each of these trajectories are of horizon length 100 with a timestep of 0.05s, which is different than the NTRT timestep specified in the previous section as we only recorded one point for every 50 NTRT simulation steps. This was done in order to increase the overall length of the trajectory as it was unlikely for periodic gaits to emerge in short spans of time [9]. The neural network architecture used 2 hidden layers with 100 hidden units each and ReLU activation. Finally, many cost functions were experimented with, where the desired goal of locomotion was to encourage the robot to travel in the $+x$ direction and to maintain a high center of mass (i.e. keep the robot standing). However, as was mentioned at the beginning of this section, we were unable to achieve a meaningful locomotion policy.

C. Results and Analysis

Through our experimentation, we observed that while the robot was learning, as evidenced by the decreasing cost over iterations (an example of which can be seen in Fig 2), no stable locomotion policy was emerging. We believe that this could be due to the following reasons. Each of these reasons will be discussed in more detail below.

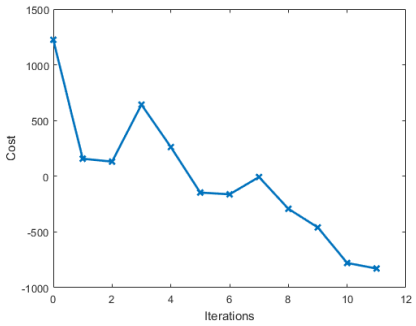

Fig. 2: The cost over iterations of MDGPS

- The cost functions used were too simple and were not expressive enough to indicate locomotion success
- Despite randomizing the initial condition, there was insufficient exploration of the robot's state and action spaces during training
- Highly coupled and non-linear dynamics resulted in small neighborhoods where linear approximations are accurate

While we iterated through many cost functions during our attempts at training, they were all focused on the average position and velocity of the robot. Specifically we formulated cost functions which would minimize the distance of the robot to a goal position or maximize the robot's velocity in a certain direction. In order to make the cost invariant to the global frame, thereby removing the policy's dependence on absolute linear position, we attempted to reference the body positions to the body center of mass and to reference the leg positions to either the positions of the hips or shoulders, depending on which one they were attached to. We also added terms in the cost which would penalize a low center of mass as well as downward body velocity in order to encourage the robot to stay upright. Most of these attempts resulted in a decreasing cost over training iterations similar to that seen in Fig 2, but what we observed is that the robot merely "learns" how to fall in the right direction and is unable to stand back up. Any further movement in the right direction is then caused by an inching/crawling motion which appeared to be rather arbitrary. These results suggest that the cost functions used were not sufficiently expressive to obtain meaningful supervision from trajectory optimization and that perhaps a more explicit definition of locomotion success was needed.

However, our observations run contrary to what has been demonstrated in the literature where stable and periodic locomotion gaits were able to arise from simple cost functions such as [9] and [5]. This then leads to the second point where we believe that another possible culprit is the larger action dimension of the robot used in this work. While previous works used robots with similar state dimension size, because they were rigid, the action space was limited to the number of joint torques that were being controlled. The quadruped

robot here, despite having simpler legs, has a flexible and controllable body where the relationship between body shape and locomotion performance is complex and not immediately apparent. This contributes to the difficulty of learning a policy due to the need to explore very high-dimensional state and action spaces. This, coupled with the fact that the robot in an upright standing position is an unstable equilibrium state, most likely resulted in inadequate exploration of states that were most important for it to develop a good policy, as most actions would result in the robot falling down. In addition, once the robot collapses, trying to get back up requires a complex coordination of actions between the legs and the spine, which further complicates training.

Ultimately, both of the issues above are likely related to the inherent dynamics of the robot, which are highly coupled due to the series-elastic cable tension network. Operating in a contact-rich environment also adds in discontinuities which cannot be well approximated using linear dynamics. Unfortunately, due to time constraints, we were not able to further investigate methods which may improve the process of trajectory optimization, which would result in better supervision for the global policy. A suggestion from the teaching staff was to use a model-free local optimizer which may be better suited for dealing with the non-linearities and discontinuities in the system. We plan on continuing work on this robot as research as we believe there is still much potential here.

III. Locomotion for a 6-bar Spherical Tensegrity Robot

A. Simulation Environment and Dynamics

In contrast to the approach we used for the quadruped robot, we instead used MATLAB to create a custom simulation environment for the spherical tensegrity robot. Using this environment, we were able to generate optimal trajectories in a relatively short amount of time without the additional delay caused by the overhead of ROS and (primarily) NTRT. Creating the MATLAB simulation environment entailed accurately modeling the rigid-body dynamics of the tensegrity system with careful attention to collision forces between rods as well as contact forces between the robot and the ground.

For accurate simulation of the agent with fast computation, we used MATLAB's ODE23 non-stiff differential solver with equations of motion derived using Lagrangian mechanics.

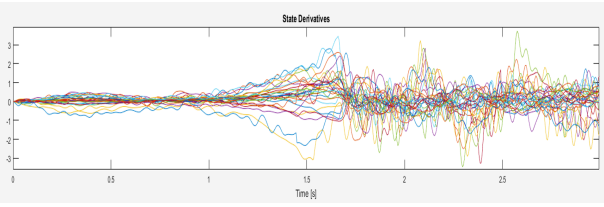

Fig. 3: Linear and angular velocities of the spherical tensegrity robot which is simulated in MATLAB. Note the high dimensional state-space and rapid changes due to contact with the ground at $t = 2.3s$

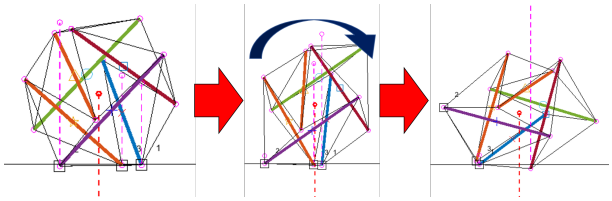

Fig. 4: Spherical tensegrity rolling motion in MATLAB simulation environment

Specifically, each rod is identified by its center of mass (CoM) position and a unit direction vector represented in polar and azimuthal spherical coordinates. Rotation about the axial direction of each rod is unconstrained and thus neglected. Each cable and rod element is modeled as an ideal two-force member in either pure compression or pure tension, making procedural generation of the nonlinear dynamics of the tensegrity system relatively straightforward given robot dimensions, physical parameters, and a connectivity matrix representing the interconnections between the individual rods and cables.

The simulation environment uses a discretized input with a zero-order hold and timesteps of $\Delta T = 0.005$s between each update of the controller input. Continuous dynamics of the system, however, are calculated in finer resolution within each timestep using an adaptive sub-step size with the differential solver mentioned above. This adaptive step size allows for more accurate simulation of complex interactions between the individual rods and the ground which is necessary for the contact-rich application of rolling locomotion.

B. Model Predictive Control and Imitation Learning

As an expert policy, model predictive control (MPC) was used to generate optimal state-action trajectories for imitation learning. This control schema iteratively solves a constrained optimization problem over a finite-time horizon at each timestep of the simulation and implements only the next control input. The primary advantage of MPC is the ability to predict future dynamical behavior while also complying with state and input constraints. However, the nonlinear dynamics of the tensegrity system introduce many complexities that make real-time calculation of the optimization problem intractable for real-time feedback control. For this reason, we utilize imitation learning to train a deep neural network policy for near-optimal online feedback control.

In order to reduce the computational effort required for optimal trajectory generation and make the optimization problem convex, the nonlinear dynamics of the tensegrity

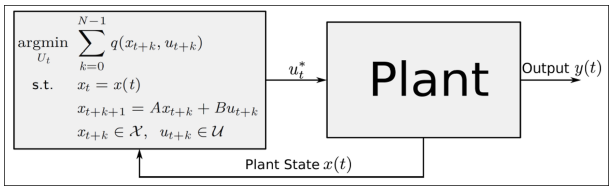

Fig. 5: Model predictive control

agent are linearized at each timestep by taking the Jacobian of the dynamics with respect to both state and input deviations from the linearization point. This linearized model is a deterministic model that can be represented as $\dot{x_t} = A_t \Delta x_t + B_t \Delta u_t + e_t$, where t is the MPC timestep, A_t is the Jacobian of the nonlinear dynamics with respect to state deviations, B_t is the Jacobian with respect to input deviations, and e_t is the constant term to account for dynamics at the non-equilibrium linearization point. These linearized system dynamics are used as linear constraints to form a convex quadratic program with the user-specified cost shown below in Eq. 4. This quadratic program is then solved at each trajectory timestep k using an optimization toolbox called GUROBI to find the optimal set of inputs over a short horizon of $N = 5$.

$$\min_{x_0, u_0, \ldots, x_N, u_N} \sum_{t=0}^{N} \gamma^t \Bigg(-\alpha \sum_{m=1}^{M} \mathcal{D}^\top v_{m,t}(z_{m,t} - \bar{z}_t) + (u_t - l_0)^\top R (u_t - l_0) \Bigg)$$

$$\begin{aligned} \text{s.t.} \quad & x_0 = x(k) \\ & \dot{x_t} = A_t \Delta x_t + B_t \Delta u_t + e_t \quad \forall t = 0 \ldots N \\ & x_{t+1} = x_t + \Delta T \cdot \dot{x_t} \quad \forall t = 0 \ldots N-1 \\ & (u_{t+1} - u_t) \leq \Delta T \cdot v_{max} \quad \forall t = 0 \ldots N-1 \end{aligned} \tag{4}$$

Here, M is the number of nodes (rod ends) on the robot, which for the 6-bar robot is equal to 12. $z_{m,t}$ represents the vertical position state of end node m and $\bar{z}_t$ is the center of mass height of the robot, calculated as the average heights of the 6 rods at the current timestep. γ is a discount factor on later rewards and α is a scaling factor for weighing the reward on nodal velocity $v_{m,t}$, in the desired direction $\mathcal{D}$. In addition, the velocity reward of each node is also weighted by the difference in height between itself and the robot's CoM. This was done in order to give a higher velocity reward to nodes farther from the CoM, for a ball-like rolling motion. Finally, l_0 are the unactuated pre-tensioned cable lengths for each cable when the robot is initially at rest and the quadratic cost matrix is $R = I_{24}$. This cost, summed over the finite horizon of the MPC quadratic optimization problem, helps guide the robot towards fluidly rolling by rewarding the nodes moving in a desired direction. In addition, the cable control length inputs were also penalized with respect to deviations from their initial lengths to avoid excessive shape-shifting. The constraints to the optimization problem are the linearized dynamics of the system, which are discretized using forward Euler integration, and the actuator limitations (e.g. the maximum cable extension/retraction at each timestep). The initial condition is also enforced using an equality constraint which sets the MPC initial state equal to the current state of the system at time k.

In order to improve exploration of the state and action space during trajectory optimization, we started the agent in different orientations for each sample trial and randomly initialized the cable control lengths of the robot to be within 20% of the nominal control length. The nominal control lengths, when applied, would result in the robot taking the shape of a regular icosahedron when free from the effects of gravity. After adding in gravity and the perturbed initial control lengths, the initial equilibrium state $x(0)$ of the robot then needs to be found before running MPC. This is done using the dynamic relaxation (DR) method of form finding similar to that presented in [13]. We run DR until convergence within an error bound and use the resulting equilibrium state as the initial condition for MPC.

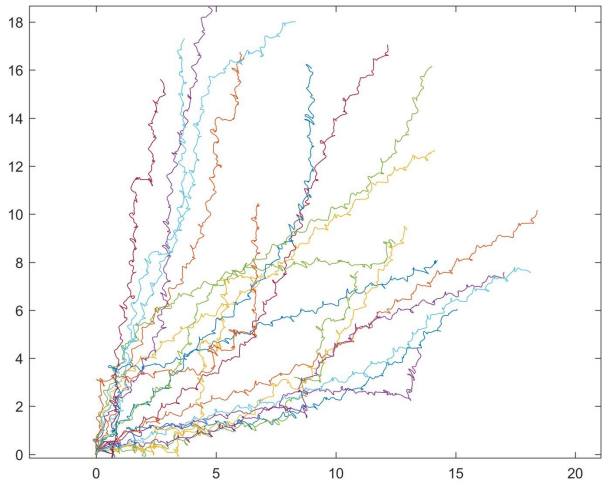

Fig. 6: Rolling trajectories over 75 seconds with the desired heading of [1,1] sampled using the MPC policy

The optimal state-action trajectories generated using the model predictive control approach described above are then used as training data pairs for supervised learning of a neural network policy. We evaluated two approaches to training - one where state-action pairs are weighted inversely proportional to the relative MPC cost and one where all data is equally weighted. From our initial preliminary results, the policies that were trained using unweighted supervised learning performed better and thus was the approach we used for all of our experiments.

C. Results and Analysis

Our goal was to evaluate how well a policy trained using simple imitation learning could emulate an optimal MPC expert policy for rolling motion in a straight line. As well, we were interested in seeing how robust the trained policy was to model-mismatch between the training and testing conditions and how much randomizing physical parameters at training time would affect the controller's performance.

For the first experiment, We generated 118 trajectories of 15,000 timesteps (75 seconds) each. The optimal trajectories were encouraged to move in the [1,1] x-y direction by the cost function, and some example trajectories can be seen in Fig. 6. We discovered that the rolling path of the robot is heavily dependent on its initial condition, with the triangular geometry of the faces on the outer surface of the robot playing a large role in determining how accurate the rolling motion can be completed in the desired heading; we believe that this is the main reason behind the variation in trajectories

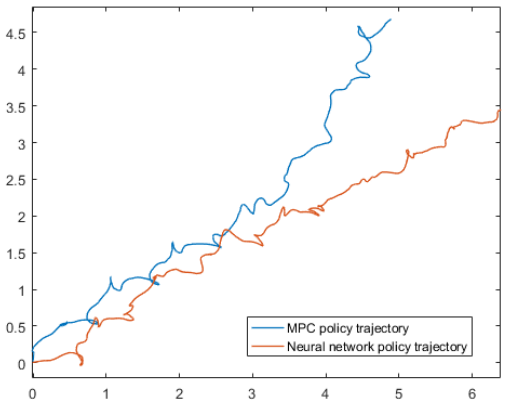

Fig. 7: Comparison between a trajectory generated using MPC and using the trained neural network policy

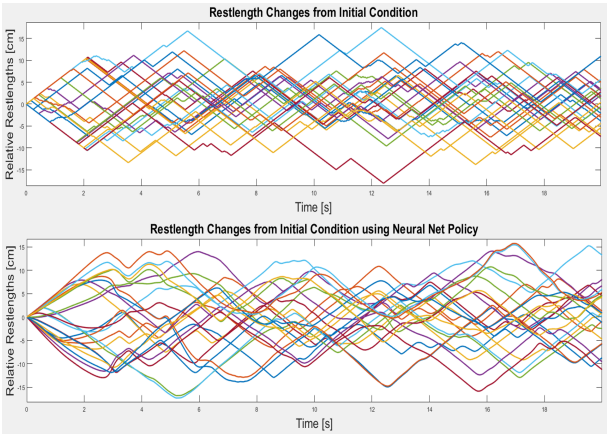

Fig. 8: Cable actuation inputs from the MPC policy (top) and neural network policy with input remapping (bottom) over 20 seconds

as seen in Fig. 6. Nevertheless, continuous rolling motion in the general desired direction was achieved for all 118 trials using the trained neural network policy comprised of two hidden layers of 64 nodes each with tanh activation functions. This policy was trained using supervised learning with input remapping from full-state information from MPC training data to only partial observability of the sensors readily available on the hardware robot - IMU data of the orientations and accelerations of each rod. One example of a rollout using the trained policy can be seen in Fig. 7. Fig. 8 shows the input trajectories for all cables found using MPC and the trained policy. It can be seen here that the neural network policy was able to learn from the MPC expert effectively and generalize the behavior seen from MPC to a policy that produces stable locomotion. Interestingly, the neural network policy seems to have a smoothing effect on the inputs where the transitions from cable extension to retraction and vice versa are more gradual compared to those observed in MPC.

Our second experiment involved investigating the effect

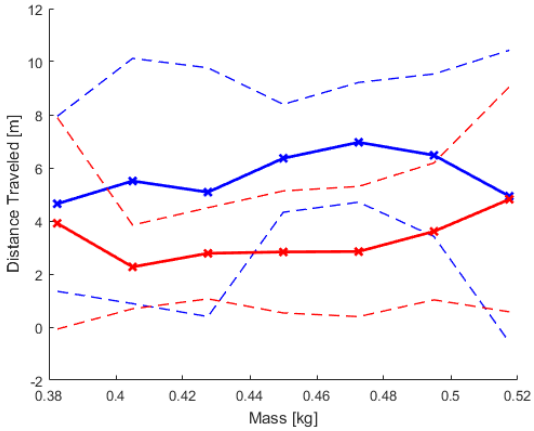

Fig. 9: Performance of the original policy (blue) and the randomized parameter policy (red). The solid lines represent the average over trials at each parameter value, the dashed lines represent performance two standard deviations from the mean

of randomizing system parameters during trajectory optimization to see if this would result in policies that were more robust to model mismatch. To do this, we sampled 100 trajectories each of length 3000 (15 seconds). Before training began, a system parameter was drawn from a normal distribution centered about the nominal value with a standard deviation of 10% of the nominal value. The rest of the trajectory optimization process remained unchanged. This results in trajectories which are optimized to succeed under various system models. As we used shorter trajectories here in an attempt to compensate for trajectory length and explore more system states, we also randomized the face on which the robot would start. Using this method, we investigated the randomization of rod mass and its effects on the trained policy. Similar to before, the neural network architecture used here has two hidden layers with 64 hidden units and tanh activation. After training, we evaluated locomotion performance based on the amount of distance that was traveled while varying the parameters from -85% to 115% of the nominal value in increments of 5%. Five trajectories were sampled at each parameter value for both the policy trained on randomized parameters and the original policy.

Fig. 9 shows the results when varying the mass of the robot's rods from the nominal value of 0.45kg. It can be seen here that the policy trained with randomized parameters actually underperforms compared to the original policy when model mismatch is introduced, but has lower variance for a wider range of parameter values. In addition, the performance trend of the original policy seems to match that presented in [14], where a similar study is performed but with a different approach. It can be seen that the best performance is achieved at the nominal parameter value. Around the nominal value, variance and average performance degrade slowly as parameter variation increases. On the other hand, performance of the policy trained with random parameters

seems to exhibit more consistent performance around the nominal value, but was not able to travel as far in distance. We may attribute the increase in variance at the two ends of the spectrum to the fact that fewer trajectories were available with those parameter values due to them being sampled from a normal distribution, and thus supervision was sparser in those regions.

It should be made clear here that there is an important discrepancy in the number of samples used to train the original policy and the one with randomized parameters. Where the original policy was trained with almost 1.8 million data points, when training the policy with randomized parameters, only 300,000 data points were used. This was due to a limitation in the time available to generate sample trajectories. By randomizing model parameters, we are effectively creating an ensemble of models to use as experts. This is especially problematic as arguably more samples are needed in the case of additional model degrees of freedom to sufficiently populate the state and action spaces across the model space with expert behavior. We believe that better robust performance will arise from generating more trajectories using MPC, and we will continue exploring this topic as we continue our research.

IV. Conclusion

In this report, we presented our work on learning locomotion policies using model-based RL methods for a quadruped robot with tensegrity spine and a 6-bar spherical tensegrity robot. While we were not able to achieve stable locomotion for the spine robot, we did learn a lot about the intricacies of doing learning for a high dimensional system by reflecting on and analyzing the shortcomings of our method. The lessons that we learned here will be useful towards our future work with this robot as it remains a challenging and interesting problem. Something that we did not discuss in the report but would like to include here is that we also ran into some issues with the NTRT simulation environment. Specifically, there were stability issues where we saw the robot sliding on the ground after it had collapsed. In addition, the way we built the agent was not very conducive to arbitrary initialization of the robot, which was an important aspect of previous work that used a similar method. Because of this, we hope to transition away from NTRT to Mujoco, which will provide better fidelity in our future simulations.

As for the 6-bar robot, it was able to learn a meaningful locomotion policy using trajectory optimization through MPC and imitation learning. We were also able to investigate the effects of parameter randomization on the robustness of the trained policy. Here we found that while unable to match the average performance of the original policy, the policy trained with perturbed rod mass appeared to show lower variance across a wider range of mass values. We are aware that the conclusions reached here may change upon the introduction of more samples and will be continuing on with this experiment to hopefully achieve better results.

References

[1] R. E. Skelton and M. C. de Oliveira, *Tensegrity Systems*. Springer, Boston, MA: Springer-Verlag, 2009.

[2] R. E. Skelton, R. Adhikari, J. P. Pinaud, W. Chan, and J. W. Helton, "An introduction to the mechanics of tensegrity structures," in *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, vol. 5, 2001, pp. 4254–4259 vol.5.

[3] L. H. Chen, K. Kim, E. Tang, K. Li, R. House, E. Zhu, K. Fountain, A. M. Agogino, A. K. Agogino, and V. SunSpiral, "Soft spherical tensegrity robot design using rod-centered actuation and control," *Journal of Mechanisms and Robotics*, vol. 9, no. 2, 2017.

[4] L.-H. Chen, B. Cera, E. L. Zhu, R. Edmunds, F. Rice, A. Bronars, E. Tang, S. R. Malekshahi, O. Romero, A. K. Agogino, and A. M. Agogino, "Inclined surface locomotion strategies for spherical tensegrity robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9 2017.

[5] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3132–3140.

[6] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," *CoRR*, vol. abs/1708.02596, 2017. [Online]. Available: http://arxiv.org/abs/1708.02596

[7] C. Sultan, "Modeling, design, and control of tensegrity structures with applications," Ph.D. dissertation, Purdue University, 1999.

[8] R. E. Skelton, J.-P. Pinaud, and D. Mingori, "Dynamics of the shell class of tensegrity structures," *Journal of the Franklin Institute*, vol. 338, no. 2-3, pp. 255–320, 2001.

[9] M. Zhang, X. Geng, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel, and S. Levine, "Deep reinforcement learning for tensegrity robot locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 5 2017, pp. 634–641.

[10] W. Montgomery and S. Levine, "Guided policy search as approximate mirror descent," *CoRR*, vol. abs/1607.04614, 2016. [Online]. Available: http://arxiv.org/abs/1607.04614

[11] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1071–1079.

[12] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *ICINCO*, 2004.

[13] K. Kim, A. K. Agogino, A. Toghyan, D. Moon, L. Taneja, and A. M. Agogino, "Robust learning of tensegrity robot control for locomotion through form-finding," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9 2015, pp. 5824–5831.

[14] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *CoRR*, vol. abs/1610.01283, 2016. [Online]. Available: http://arxiv.org/abs/1610.01283