Continuous Design Variable Optimization in Modular Robot Design through Deep Reinforcement Learning

Max Asselmeier[1], Julian Whitman[2] and Howie Choset[2]

***Abstract*— Modular robots allow for a robust method of catering a robotic system to the task, or tasks, that it is to complete. However, many of the methods that develop ways to generate modular robot designs do so with a finite, discrete pool of modules to pick from. The methods that are able to handle continuous design parameters for these modular arms do not currently leverage the efficiency afforded by deep reinforcement learning algorithms. Continuous design variables would offer another level of versatility and customization with regards to the creation of modular robotic systems. Additionally, reinforcement learning is a computationally efficient way of evaluating modules that can be added to an existing modular arrangement, which is normally an exponentially complex problem. In this work, we offer forth a framework that allows for the combination of the discrete decision of selecting a module group to add to an arrangement through a Deep-Q Network with a continuous decision that optimizes the design variables for the given module group through the Soft Actor-Critic algorithm. We then provide results for the training of the Deep-Q Network on a set of finite modules along with the training of the Soft Actor-Critic algorithm on a relaxed constraint problem.**

***Index Terms*— Kinematics, Novel Deep Learning Methods, Reinforcement Learning, Task Planning**

I. INTRODUCTION

The modular design of robotic arms allows for the specialization of a robot to the task that it is to complete. However, this specialization hinges upon the ability to identify an optimal design, or the proper sequence of modules used to create a robotic arm. This process allows for an expansive amount of creative choices and a high degree of optimization with regards to the specific application of a robotic arm: the more modules that are available to use, the more types of arms that can be generated and the more types of tasks that can be completed. Human experts are capable of generating these optimal robotic arm designs, but this greatly restricts the types of users that can create these arms. Developing a tool capable of learning how to design a modular arm would allow a layperson to engage with this design process and potentially even create a design that an expert user could not conceive of. Additionally, this tool would be useful in situations where the task at hand changes frequently over time. Each unique task would require a re-design from an expert user whereas this tool would be able to produce a useful design in a much quicker fashion.

[1]Max Asselmeier is with the Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA ma53@illinois.edu

[2]Julian Whitman and Howie Choset are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA {jwhitman, choset}@cmu.edu

The recent advancements in deep learning have allowed for the utilization of reinforcement learning (RL) as a tool for facilitating search problems such as modular robot design [1], [2]. Computational complexity and time are severe limitations to how large search trees can be, and deep learning provides heuristics for solving these search problems. For modular robot design, the variations of the arrangements of modular arms grow exponentially with the number of different modules that can be added, but deep reinforcement learning provides ways to evaluate module arrangements and focus on those that may be of interest. With this being said, reinforcement learning tools like neural networks also have limitations on how they generate these evaluations. Prior work [1] on this project has employed the use of a Deep-Q Network (DQN), a popular tool in reinforcement learning which requires a finite output space. This type of network provides state-action values known as Q-values that function as scores for each possible action that can be taken from a given state. This works well for modular robot design, but only if we sample from a discrete pool of modules. If we want to adjust design parameters such as the length or mass of a link, then we will end up with an infinitely large action space with each action representing a link of an infinitesimally smaller or larger length or mass, and our problem quickly becomes intractable. This option to tweak design parameters for a robotic arm provides more freedom and flexibility for automated modular robot design, but a different approach that is specifically designed for continuous action spaces must be used.

In this paper, we build on prior methods of modular robot design [1] and develop a framework that would allow for the utilization of a continuous action space. We also initiate work on training a deep RL algorithm that can output the optimal continuous design variables for a certain task. Our framework involves a hierarchical structure of neural networks separating the task of adding a new module to an already existent arrangement of modules into two sub-tasks. A primary network would first choose the discrete type of module to be added to a modular arm such as a link or bracket. Then, a secondary network would set the module type's corresponding design variables. Setting these design variables could be a discrete decision such as choosing if a bracket will be pointed either towards or away from the robotic arm, or it could be a continuous decision such as setting the length of a link. The ability to set or alter these continuous design variables, and more so to do this in the same networks that set the discrete design variables, would allow for a much greater breadth of options when it comes

to which modules are being added to the robotic arm and what characteristics these modules possess.

In this paper, Section II will detail literature and concepts that are related to the work discussed in this paper. Section III will explain the potential methodology of our hierarchical neural networks along with supplementary information for our tasks and solutions. Section IV will present our results, Section V will involve closing remarks on this research, and finally, Section VI will provide limitations of this work along with directions for future research.

II. BACKGROUND

A. Related Works

Previous implementations of modular design synthesis exist, but these methods typically utilize a discrete set of modules. Tools ranging from interactive design systems to best-first graph searches make use of libraries of standard modular components from which complete arrangements for modular robots are created [2], [3], [4]. Evolutionary algorithms are also used to compose robot designs from a finite set of modules [5]. However, these methods all select from a finite number of modules. Genetic algorithms have been able to optimize a mixture of both discrete and continuous design variables [6], but this work does not leverage the deep learning algorithms frequently used now.

Deep learning within robot design has been employed in other forms. For instance, deep learning tools have been utilized to jointly learn not only the structure or design of a robot, but also the robot's motion control policy [7], [8]. Also, prior work on this project has involved utilizing a Deep-Q Network to generate efficient designs for modular serial manipulators [1]. The synthesis of robot designs from a discrete pool of modules allows for the development of robotic systems that are customized towards their given tasks. However, confining modular design synthesis to a set of modules does enforce restrictions on both the types of robots that can be created and the types of tasks that can be accomplished. Continuous design variables would allow for much more freedom when it comes to creating modular robots, and multiple deep learning algorithms that are applicable to continuous action spaces have already been developed.

B. Deep learning in continuous output spaces

Several deep learning algorithms are able to be utilized within continuous output spaces. One way that these algorithms can produce continuous outputs is through methods such as using soft bounding functions to limit the regular outputs of a network's layer to the desired bounds of a given action. The Deep Deterministic Policy Gradients (DDPG) algorithm [9], the deep version of the previously developed Deterministic Policy Gradients [10]. DDPG is a model-free, off-policy algorithm that learns a deterministic policy by using the aforementioned bounding functions. The Twin Delayed Deep Deterministic (TD3) policy gradients algorithm is a modification to DDPG that also does this.

Algorithms can also generate these continuous outputs by producing the mean and standard deviation for a distribution and sampling from this distribution to obtain values. The Soft Actor-Critic (SAC) algorithm [11] is a model-free, off-policy algorithm that performs this idea of sampling from a distribution.

Prior work has used these deep learning algorithms within continuous action spaces to have robot platforms learn simple and compound abstract tasks [12] as well as learn complex manipulation tasks [13]. Deep RL has also been used to learn motion planners with continuous outputs for robotic systems as well. However, no prior work has incorporated continuous action spaces into modular robot design. Our work on this project has initiated the exploration of the optimization of continuous actions within modular design synthesis.

C. Deep learning for Modular Robot Design

Our modular design problem is treated as a finite Markov Decision Process where modules that are to be serially added to an arrangement are evaluated based on the current arrangement as well as the goal position of the episode. The module currently being added to the arrangement is connected to the module that was previously added.

Therefore, the state s_t of this problem is comprised of the active arrangement of the arm, or what modules are currently within the arm along with the goal position that is to be reached. An action a_t is referred to as the process of appending a new module onto the arrangement, and a reward r_t is obtained from the environment at each step based on both the state and the action that is taken. Further information on discrete Q-learning for modular robot design can be found in the previous work on this project [1].

The Soft Actor-Critic algorithm is employed in this project to begin work on the implementation of deep RL on continuous design variables in modular robot design. While most other algorithms only attempt to maximize the expected rewards achieved throughout training, SAC seeks to maximize the expected reward of the actor while also maximizing the entropy of the actor. This means that SAC attempts to have the actor succeed at the given task as frequently as possible while varying its actions as much as possible as well. This idea is evident when viewing the objective function used for the previously discussed algorithms such as DDPG or TD3

$$J(\pi) = \Sigma_t \mathbb{E}_{(s_t,a_t)\sim p_\pi}[r(s_t,a_t)], \tag{1}$$

where s_t, a_t, and $r(s_t,a_t)$ are our state, action, and reward respectively at a certain step, and p_π is our policy. This objective function can then be compared to the augmented objective function that is used for SAC

$$J(\pi) = \sum_{t=0}^{T-1} \mathbb{E}_{(s_t,a_t)\sim p_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))]. \tag{2}$$

We can see that an entropy term denoted by $\mathcal{H}$ is added to the reward, and the importance of the entropy term is decided by the temperature parameter α. If α is set to zero, then the traditional objective function based solely on expected rewards is recovered.

With our objective function in mind, the soft Bellman Equation that is used to estimate the state-action value function Q^π can be formulated as:

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p_s}[V^\pi(s_{t+1})] \quad (3)$$

where the soft value $V^\pi(s_t)$ is equal to

$$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi}[Q^\pi(s_t, a_t) - \alpha \log \pi(a_t|s_t)]. \quad (4)$$

Here it can be seen that the entropy term for SAC is defined as the negative log of the current policy.

SAC adapts the double-Q learning trick used in TD3 where two critic networks are trained. Once an action is taken, both critics will evaluate the action and return their respective Q-values. When the networks are being trained, the lesser of the two Q-values between the two critics is taken to reduce overestimation bias. SAC also uses target networks [14] and experience replay [15] to facilitate and stabilize the training of the networks. Furthermore, we use Hindsight Experience Replay (HER) [16] which is a data augmentation technique applied to the replay buffer during training to help with the sparse reward function that is used as part of this work.

III. METHODS

This project adapts much of its framework from prior work on this project [1]. We previously utilized a DQN to approximate the state-action values for a set of modules that can be added to a robotic arm. However, since one of the core ideas of this project is the implementation of continuous action spaces which are incompatible with DQNs, the framework that we developed plans on using our DQN in a slightly different way. For this framework, our DQN would instead select the *type* of module to be added. For instance, instead of selecting a link with predetermined design variables, the DQN selects the overall module group of "link", and the variables would be set later on in the architecture of the problem. This combination of discrete and continuous decisions was not achieved during this project, and plans to implement it are detailed in section VI. The desired framework for this hierarchy is detailed below.

The DQN and SAC networks will still be used to design an optimal modular arm design given a goal position in space. The inputs to the networks would aim to represent the current state of the arm along with the goal position that the arm is to reach. The outputs of these networks would aim to select the best possible module that could be appended to the arm in order to reach the goal position. A reward is given to the networks if the arm is able to get within a certain distance to the goal position, and the networks selects modules to maximize both the amount of rewards obtained along with the randomness of the modules chosen.

The task space for this problem is limited to one position $p \in \mathbb{R}^3$ where p = [p_x,p_y,p_z]. This means that during each episode of training, inverse kinematics (IK) is performed on the end-effector of the arrangement with the respective point p of that episode as the goal position. Inverse kinematics is solved through the module PyBullet [17] with the Damped Least Squares method [18]. A tolerance ϵ_p is set so that a reachability function for an arrangement $A \in \mathbb{R}^{N_{max} \times N_m}$ where N_{max} is the maximum number of modules allowed in an arrangement and N_m is the number of modules to choose from and target position $T \in \mathbb{R}^3$ can be defined as follows:

$$reach(A, T) = \begin{cases} 1 & || \, p - p_{EE} \, || < \epsilon_p \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Where p_{EE} is the location of the end-effector of the arrangement in space after forward kinematics has been performed on the arrangement using the angles obtained through inverse kinematics. These modular arrangements are also evaluated on other non-terminal conditions such as the mass and complexity of the arrangement. The mass of the arrangement $M(A)$ is simply calculated by summing up the masses of the individual modules in the arrangement, and the complexity of the arrangement is represented by the number of actuated joints in the arrangement $N_J(A)$. An objective function for these conditions can be defined as

$$F(A, T) = -w_J N_J(A) - w_M M(A) + reach(A, T). \quad (6)$$

Weights w_J and w_M are determined by the user based on how important the mass and complexity of the arrangements are. Therefore, it follows that an optimal arrangement is capable of maximizing this function for its singular goal position:

$$A^* = \arg\max_A F(A, T) \quad (7)$$

Now, we outline the proposed way to train these networks to discover and select these optimal arrangements.

A. DQN for module group selection

The actions from our DQN are chosen from a set of four options: actuators, brackets, links, and end-effectors. As of now, the links are the only module group that support continuous design variables. For the other three module types, the DQN simply would select from a discrete pool of modules with predetermined design variables.

The arrangement of each modular arm would be represented through a list of one-hot vectors such that each index of a vector indicates either the discrete module chosen for the arrangement or the module group selected in the case of a link.

The design variables for the arrangement would be represented through a list of vectors where each index of the vectors represents a singular design variable for the module that is at the position in the arrangement of the corresponding index of the vector within the list. If a module within the arrangement were to be chosen from a discrete set and therefore not possess any continuous design variables, then its vector of design variables would simply consist of all zeroes. Also, if an arrangement would be to end with less than the maximum number of modules, then the vectors occupying the empty indices of the list would also contain all zeroes.

The design variables would be passed through a pre-processing layer for the module type that is to be added so

that even if the numbers of design variables for two types of modules are different, the outputs of the pre-processing layer are still the same size. For instance, a pre-processing layer for links would accept as its input the design variables for a link, a pre-processing layer for brackets would accept as its input the design variables for a bracket, and both of these layers would have the same size outputs. All of these outputs can be appended for a given arrangement so that the processed design variable list is the same size for all arrangements. The structure of the networks is further detailed in Figure 1.

At each step within an episode, the DQN would either select a discrete module or a module group. If a discrete module is chosen, then this module would just be appended to the arrangement and there would be no more activity from the networks for the duration of this step. However, if a module group were to be chosen, then the state of the arrangement along with the module group to be added would be passed to the SAC networks.

The reward function would be identical for all arrangements and it would consist of non-terminal and terminal components. The non-terminal penalties would come from the mass and complexity of the module m that is to be added to the arrangement:

$$r(m) = -w_J N_J(m) - w_M M(m) \tag{8}$$

If the module that is chosen by the DQN were to be an end-effector, then the action as well as the next state of the arrangement would be terminal, and a terminal reward would be returned. The terminal reward function evaluates two features of the arrangement. If the maximum number of modules for an arrangement is reached without an end-effector being added to the arm, then a terminal reward of -1 would be returned. Otherwise, if an end-effector were to be added at any point along the arrangement, then the previously detailed reachability function for the arrangement would be evaluated

$$r_{terminal} = \begin{cases} -1 & \text{length(A') == } N_{max} \\ & \text{and } m \text{ is not an EE} \\ reach(A', T) & m \text{ is an EE} \end{cases} \tag{9}$$

where A' is the arrangement after the action has been taken and EE is an end-effector.

B. SAC for design variable selection

SAC consists of an actor-critic framework where a single actor network learns a stochastic Gaussian policy by outputting mean and standard deviation values that can be used to create a Gaussian distribution. Actions are then sampled from this distribution. Two critic networks evaluate and return Q-values for the actions taken by the actor network. The actor network accepts the state as its input and outputs actions, and the critic networks accept the state as well as the current actions and outputs a Q-value that evaluates the action taken.

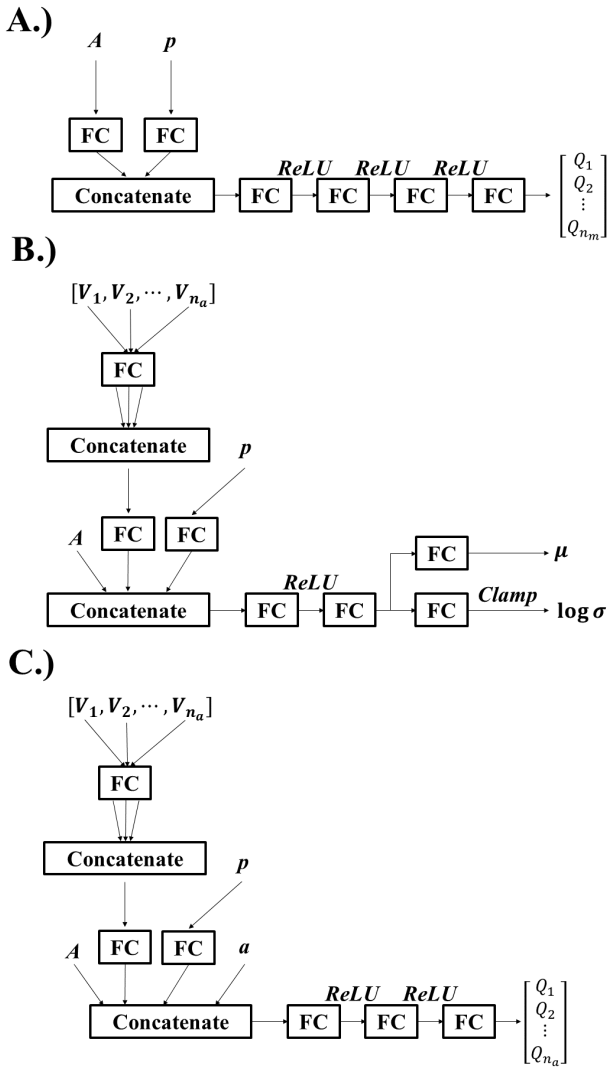

Fig. 1. The structures that we used for our DQN and SAC networks. A.) shows our DQN, B.) shows the actor network for SAC, and C.) shows the critic network for SAC. These networks use fully connected (FC) layers, concatenations, rectified linear units (ReLU), and clamps between minimum and maximum values. The DQN accepts an arrangement A and a target position p as its input and outputs a Q-value for each discrete module or module group. The Q-values span the number of modules or module groups N_m. The SAC actor takes A and p as inputs as well, but also takes in our processed design variables V of length n_a, the maximum number of design variables in an arrangement. The actor outputs a mean μ and a logarithm of a standard deviation $\log \sigma$ which are both used to calculate the action. The critic for SAC takes in the same inputs as the actor while also taking in the action a chosen by the actor. The critic outputs Q-values for each design variable set by the action.

The state for the SAC algorithm would consist of the same elements as the DQN: the current arrangement of the arm along with the goal position that the arm is to reach, but then we would also incorporate the design variables that are set for each module.

In our work, the SAC networks were not configured to train on the IK problems, and they were instead trained on a variable range problem where the sum of the lengths as well as the sum of the twists of all the links in the arrangement

had to fall within a randomly generated range. The twist of a link is the angular difference between the two joint axes at both ends of the link.

Since our SAC networks have only been adapted to links within our arrangements, the reward function only consists of one non-terminal component which penalizes for the mass of a link. The mass is penalized to encourage lighter and cheaper arrangements.

$$r(m) = -w_M M(m) \tag{10}$$

C. Training the neural network architecture

For this project, the DQN was trained on a finite module set to determine that the network functioned properly. At the beginning of each episode, the target position is generated from a random uniform distribution. The X and Y indices of the target position are generated from the range [-0.5, 0.5] whereas the Z index is generated from the range [0.0, 0.5].

As an episode progresses, the arrangement grows by appending a module after each step. At each step, the DQN outputs Q-values for each possible discrete module. The proposed continuous framework would have the DQN instead output a Q-value for each module group. The masking of certain actions is performed to ensure that each type of module can only connect to a certain subset of modules. For example, two actuators cannot be connected and two non-actuators can be connected. Q-values are learned for all actions, but only the Q-values for valid actions are evaluated when selecting an action. The Boltzmann exploration strategy is employed for the DQN in order to handle exploration and exploitation [19]. Often times when building an arrangement for a modular robot, multiple modules represent valid additions that can lead to high-reward states. With this in mind, a method such as ϵ-greedy fails to account for the exploration of multiple valuable actions since ϵ-greedy will either pick an action at random or choose the single most valuable action. Boltzmann exploration makes the process of choosing an action stochastic by creating a probability distribution across all valid actions. This allows higher value actions to still be selected more often while also ensuring that no single action is repeatedly exploited. A temperature parameter can also be adjusted to make the probability distribution more or less skewed towards higher value actions.

For the SAC networks, the actions are bounded by predefined action limits. The length of a link is restricted to the range of [0.0, 0.75] and the twist of a link is restricted to the range of [0.0, 2π]. Exploration is encouraged through the entropy term that is added to the objective function of the SAC networks, and exploration is automatically added through the random sampling of the Gaussian policy that is generated by the actor network.

While training the SAC networks, actions are sampled randomly from a uniform distribution for the first predefined number of steps. This is done to ensure a proper amount of initial exploration and to also allow for more uniformity with respect to the initial weights of the network across separate trials.

A replay buffer is utilized to allow for off-policy learning. At each step, the state, action, reward, and next state are all added to the replay buffer along with a variable signaling if the action taken was a terminal one. Since the reward function for our IK-based tasks is quite sparse, hindsight experience replay (HER) is implemented [16] to ensure that high-reward states are always existent within the buffer that training batches are sampled from. If an episode terminates and the end-effector of the arrangement is not within the distance threshold ϵ_p of the goal position to earn the terminal reward of one, then the same exact tuples of the state, action, reward, next state, and terminal variable are added to the replay buffer again, but now with their goal position as the point in space that the end-effector ended up reaching. However, if the final position of the end-effector lies outside of our original goal ranges, then the tuples are not added to the replay buffer.

Validation checks are also made at certain intervals of training to determine how the policy is being updated throughout training. These validation checks are made by simply having the policy construct an arrangement with solely exploitative actions.

IV. RESULTS

The results of this project contain results obtained from training the DQN on a finite set of modules as well as results from training the SAC networks on a relaxed problem that requires the design variables of the links within an arrangement to fall within certain goal thresholds. Both of these training series occurred separately, and the DQN and SAC networks were not trained together.

Figure 2 shows the total rewards earned from the DQN arrangements during training. These results were tabulated over three trials. It can be seen that for roughly the first 500 episodes of training, the total rewards are negative. This is due to the fact that at the beginning of our training, our Boltzmann exploration tends to select actions uniformly, so modules that are less helpful for reaching a goal position will be selected more often. Also, the network has only just started to train, and it has not seen enough high-reward states to learn what arrangements are able to reach the goal positions.

Figure 3 demonstrates the total rewards earned for the simplified SAC training. These results were also obtained over three trials, and similar trends to the rewards for the DQN training can be observed. The total rewards are negative for roughly the first 500 episodes for the same reasons provided for the DQN training, and soon after this the networks are able to select states that earn positive rewards which leads to a positive, linear increase in total rewards for the remainder of the training.

Figures 4 - 8 demonstrate the evolution of arrangements throughout the validation checks. Earlier arrangements are frequently much too short to reach goals, and as more training episodes are done, these arrangements tend to have the proper number of modules to allow the end-effector to reach the goal. Later on in training, the DQN also learns to

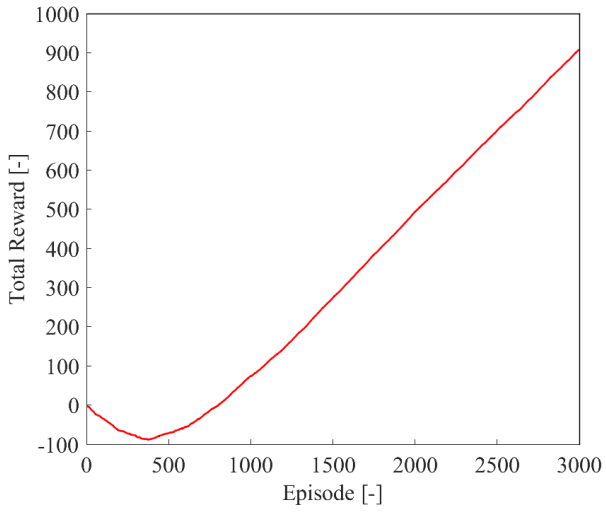

Fig. 2. A plot of the total rewards earned from the DQN over all of the training episodes. These results are averaged over three training trials, and a reward of one was given if the arrangement was able to get within two centimeters of the goal position while minor penalties were given for the mass and complexity of the arrangement at all steps.

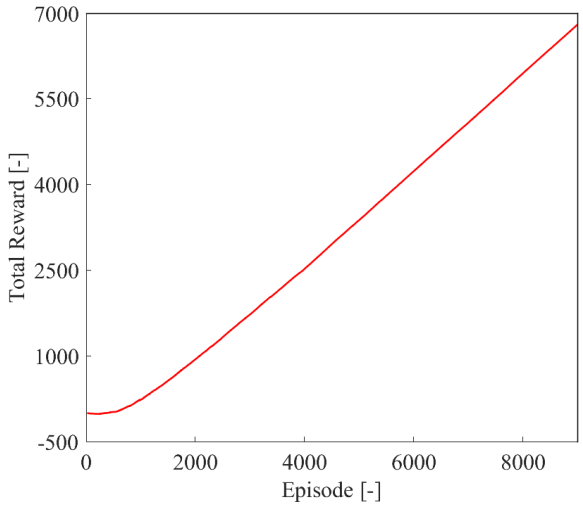

Fig. 3. A plot of the total rewards earned from the SAC networks over all of the training episodes. These results are averaged over three training trials, and a reward of one was given if both the length and twist totals fell within the randomized goal thresholds while a minor penalty was given for the mass of the arrangement at all steps.

select efficient arrangements by choosing arrangements that reach goals with the minimum amount of modules required.

V. CONCLUSIONS

Through this project, we have initiated work towards deep learning in continuous action spaces for modular robot design. We adapted a previous framework for deep RL from this project [1] and developed a plan for a hierarchical structure of neural networks that segments the process of adding a module to a current arrangement of an arm into a primary discrete section and a secondary continuous section.

We developed a potential structure for designing a modular arm arrangement where we would use a Deep-Q Network (DQN) to select either a discrete module to be added or a module group that is to be optimized for the arrangement. If a module group is selected, then the Soft Actor-Critic (SAC) algorithm would then be employed to set the continuous design variables for the type of module that is to be added to the arm.

We found that the Soft Actor-Critic (SAC) algorithm allowed us to successfully optimize continuous design variables when trained on a simplified version of our inverse kinematics tasks, and this algorithm has also led to promising results for our inverse kinematics tasks as well.

While the training of these hierarchical networks is not entirely finished, we plan on continuing to work on this project in the future. In Section 6, we go into more detail about current limitations for this work as well as our plans for this project in the future.

VI. LIMITATIONS AND FUTURE WORK

One limitation for our current work is that the SAC networks have only been trained on the length and twist variables for links. It is quite possible that different design variables that possess different action limits will require different amounts of training, and training our SAC networks on more of these design variables could help us fine tune our training. For instance, design variables could potentially be introduced to our brackets to allow for another module group that can be implemented into our SAC networks.

Looking forward, we want to optimize our SAC networks to work for the inverse kinematics tasks that we utilize for our modular robots. Once these networks would be able to return optimal design variables for a single, constrained arrangement, we would then want to integrate our SAC networks into our DQN and simultaneously train the two groups of networks at the same time. Being able to train both the DQN and SAC networks would then allow us to construct modular arrangements by selecting a module or module group through our DQN and then having our SAC networks optimize the design variables for the module that is to be added.

Beyond this, we also want to incorporate target orientations into our target positions to allow for more challenging or selective tasks. Having our modular arms satisfy both position and orientation requirements allows our arms to complete more tasks than if they were to just satisfy position constraints. For even a simple pick-and-place task, the orientation of the end-effector of a gripper or robotic arm is critical when it comes to properly picking up and setting down objects. We also want to incorporate obstacles into our environments to allow our networks to be more realistic and versatile. Forcing our modular arms to satisfy position and orientation constraints while also avoiding obstacles allows for the more robust modular designs to stand out and obtain higher rewards while also penalizing the more rigid designs that cannot adapt to obstacles.

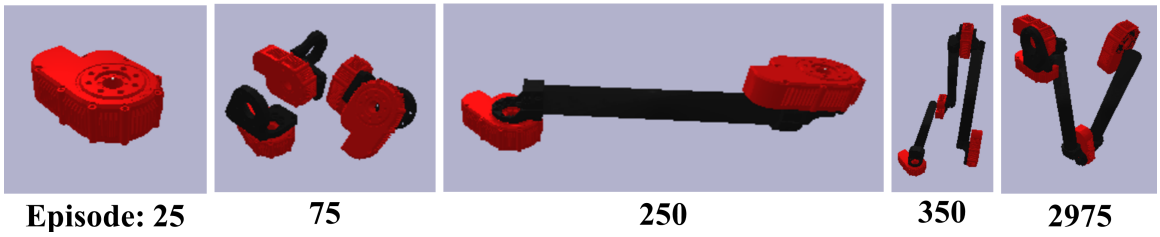

Fig. 4. Validation arrangements for the goal [0.1, 0.1, 0.1]. The numbers below the arrangements are the episodes at which these arrangements were made. Notice how at the beginning of training, the network simply outputs a single actuator which is unable to reach any positions in space. However, after only 250 episodes of training, the network is able to produce a more pragmatic design.

Fig. 5. Validation arrangements for the goal [0.2, 0.2, 0.2]. By episode 475, the network is able to produce an arrangement that can reach the desired goal position. However, by the end of the training the network has learned that only two links are needed to reach the goal instead of three.

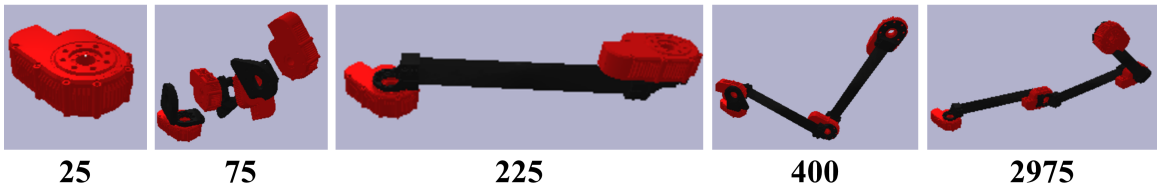

Fig. 6. Validation arrangements for the goal [0.3, 0.3, 0.3]. The arrangements for this goal actually follow a similar trajectory as those for the first validation goal. However, the arrangements for this goal end with three links instead of two for the first goal.

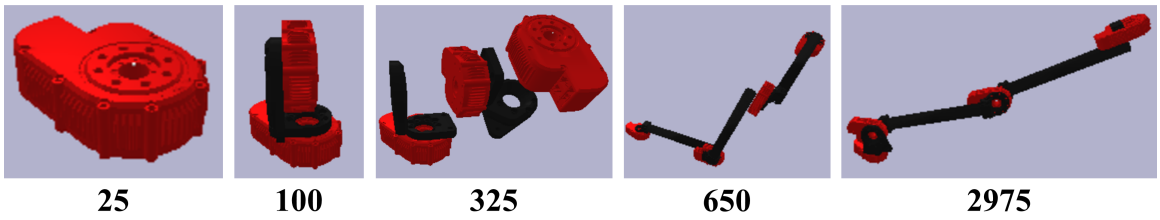

Fig. 7. Validation arrangements for the goal [0.4, 0.4, 0.4]. The first few arrangements for this goal strictly use brackets which are not very effective for reaching positions in space all by themselves. However, later on in the training the networks are able to use both links and brackets to reach the desired goal position.

Fig. 8. Validation arrangements for the goal [0.5, 0.5, 0.5]. The network appears to learn at an early point in the training that several links will be required to reach this goal position. This can be seen in the longer arrangement shown at episode 75. By the end of the training, the network still uses the same number of links as it did at episode 75, but it has moved the bracket to be earlier on in the arrangement which allows the arrangement to be more flexible.

ACKNOWLEDGMENT

I would like to thank the Biorobotics Lab for hosting me for this project and allowing me to engage with such fascinating research. I would also like to thank the Robotics Institute Summer Scholars program as well as specifically Ms. Rachel Burcin and Dr. John Dolan for working so hard

to keep this program alive for this summer. Finally, I would like to thank my home university, the University of Illinois at Urbana-Champaign, for helping me to earn the opportunity to participate in the RISS program. This material is based upon work supported by the National Science Foundation under Grant No. 1659774

REFERENCES

[1] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, "Modular robot design synthesis with deep reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2020.

[2] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," in *In Conference on Robot Learning*, 2017, pp. 271–280.

[3] R. Desai, Y. Yuan, and S. Coros, "Computational abstractions for interactive design of robotic devices," in *In IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1196–1203.

[4] R. Desai, M. Safonova, K. Muelling, and S. Coros, "Automatic design for task-specific robotic arms," in *ICRA Workshop on Autonomous Robot Design*, 2018.

[5] E. Icer, H. A. Hassan, K. El-Ayat, and M. Althoff, "Evolutionary cost-optimal composition synthesis of modular robots considering a given task," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3562–3568.

[6] Z. M. Bi and W. J. Zhang, "Concurrent optimal design of modular robotic configuration," *Journal of Robotic Systems*, vol. 18, no. 2, pp. 77–87, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-4563%28200102%2918%3A2%3C77%3A%3AAID-ROB1007%3E3.0.CO%3B2-A

[7] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, "Jointly learning to construct and control agents using deep reinforcement learning," 2018.

[8] D. Ha, "Reinforcement learning for improving agent design," *Artificial Life*, vol. 25, no. 4, p. 352–365, Nov 2019. [Online]. Available: http://dx.doi.org/10.1162/artl_a_00301

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[10] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. I–387–I–395.

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

[12] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5174–5184, 2018.

[13] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016.

[14] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[15] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005*, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328.

[16] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2017.

[17] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2020.

[18] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[19] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1, pp. 81 – 138, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/000437029400011O

[20] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[21] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf

[22] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International conference on machine learning*, 2015, pp. 1312–1320.

[23] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," 2016.