Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Final report: Hybrid Reinforcement Learning for Modular Robot Optimization

Yizhen Li, Silvester Rotenhan, Yingjie Xu

Practical Course *Robotics Intelligence* SS 2022

Advisor:	Jonathan Külz
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Submission:	31. August 2022

Hybrid Reinforcement Learning for Modular Robot Optimization

Yizhen Li, Silvester Rotenhan, Yingjie Xu
Technische Universität München
Email: yizhen.li@tum.de, ga42ziv@mytum.de, yingjie.xu@tum.de

***Abstract*—Modular robots are assembled by a set of links and joints. Currently, the task of optimizing modular robot configuration is mostly done manually. In this work, we investigate the adaptability of modular robots to a given task and environment through a self-learning process. We are implementing the P-DQN-algorithm and its variants into a self-developed environment for modular robots. The agent learns to synchronously optimize the discrete module type and continuous module parameters, resulting in a hybrid version of reinforcement learning. The performance of the optimization process is evaluated in different scenarios and finally, the P-DQN variants are compared.**

I. Introduction

As stated in most artificial intelligence research, the investigation of robot learning has a major focus on the cognitive system of robots, e.g. information processing and control theory. However, in recent years scientists are also exploring the area of morphological development. This topic deals with optimizing the configuration and parameters of a robot for its given task. Since the field of robotics originates in the human ambition to create artifacts in the image of life[1, Ch. 1.1], it is reasonable to argue that robots, too, need to adapt to their surroundings and given tasks.

The two pictures in Fig. 1 display an identical task environment for a modular robot: The green box on the top right is the target and the white spheres and box are obstacles. The robot is represented by the connected cylindrical bodies. The robot's base is on the bottom left and its goal is to reach the green target. Clearly, with only two short links the left robot is not well suited to its task, whereas the robot on the right has an extra link and thus is adapted to its surroundings.

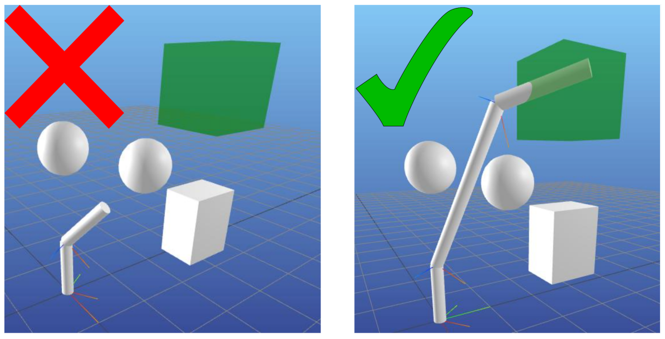


Fig. 1. Robot adaptation to the environment: left: robot cannot reach the goal; right: robot reaches goal

A robot sub-type that appears to be particularly suited for adaptation are modular robots. Their compartmentalization can be reconfigured and functional modules can be interchanged with some level of effort [1, Ch. 22.2]. Compared to traditional robots, modular robots are more flexible and can adapt their form to different tasks and environments. The adaption, namely the selection of modules is, however, in most cases done manually by heuristic or data-driven methods, which often requires professional experience or a large amount of data. For optimized and computationally more efficient results, reinforcement learning (RL) is introduced, which allows modular robots to self-learn their architecture.

In this work we

- give an overview of hybrid RL approaches,
- build a RL environment for a modular robot domain,
- implement RL agents (P-DQN [2] and MP-DQN [3]) that enable robots to adapt and self-learn their optimal configuration and parameters for a given task,
- compare the performance of several hybrid RL approaches, draw conclusions and give possible future research directions.

II. Background

In recent years, some contributions, which can serve as a guide and inspiration for our task have been made. In this section we look into the existing research in the field of modular robots and RL, starting with a modular design search.

A. *Modular robots*

In discrete modular robot design, Liu et al. propose an algorithm to find the optimal module composition, using performance metrics, considering energy and time efficiency, robotics' kinematic and dynamic characteristics, as well as the obstacle constraints [6]. Whitman et al. use RL to select modules from a discrete database [7]. Their approach creates a search heuristic that guides search on a tree of sequential modular arrangements.

However, with only fixed module types and sizes available these approaches become computationally infeasible when dealing with a large variety of module sizes. Liu also stated that by decreasing the number of modules with the same structure, but different sizes, and optimizing their module parameters instead, the computation time can further be reduced. Therefore we consider extending Whitman's approach by also optimizing the parameters of individual modules. Thus, in

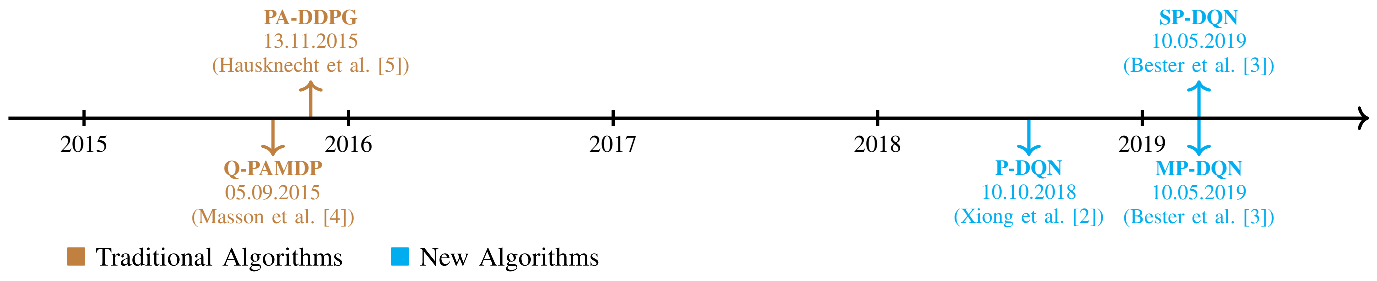


Fig. 2. Timeline of RL approaches in hybrid action domain

the next section, we take a closer look at RL for continuous parameters and hybrid domain.

B. RL

In RL, the environment is modeled as a Markov decision process if the agent can fully observe the environment. The environment state S_t at time t represents the observation an agent gets from the environment at time t. The agent performs an action A_t and then observes the next state S_{t+1}. Additionally, it receives a reward $R_t(S_t, A_t)$ indicating how well the agent is doing at step t. The policy $\pi_t = \mathbb{P}[A_t|S_t]$ defines the agent's behavior and the action-value function $Q_\pi(s,a) = \mathbb{E}_\pi[R_t|A_t = a, S_t = s]$ indicates the expected reward the agent will get starting from the state s, taking action a, and then following the policy π.

C. RL in continuous action space

Using RL in the work of Ha they train a version of an agent's design that is better suited for its task [8]. For a bipedal and also a four-legged walker their approach improves its performance when adapting the lengths and widths of its leg parts to a given terrain. Using a continuous action space to compute the parameters the agents can optimally adapt their legs to the environment. However, with only continuous parameters available this approach omits the choice of discrete parameters such as module number and module type. To enable full eligibility a hybrid approach appears to be most plausible.

D. RL in hybrid action space

RL algorithms in hybrid action space are divided into two categories: traditional algorithms and new algorithms.

1) Traditional algorithms: This category contains

a) *Separate Optimization for Discrete Actions and Continuous Parameters*. This means a discrete algorithm is adopted for discrete actions and a continuous algorithm is adopted for continuous actions. A representative of this type is Q-Parameterized Action Markov Decision Process (Q-PAMDP) [4] which alternatively optimizes discrete policy and continuous action parameters, using Sarsa (λ) [9] with the Fourier basis [10] and episodic Natural Actor-Critic (eNAC) [11] respectively.

b) *Discretization of continuous parameters* which means turning continuous action spaces into discrete spaces by binning or bucketing, and then discrete action algorithms e.g. deep Q-Network (DQN) [12] can be applied.

c) *Relaxation of discrete actions* which means mapping a discrete action space to a continuous action space using some transformations f, and then continuous action algorithms e.g. DDPG [5] can be used. With such a research orientation, Hausknecht and Stone extend the Deep Deterministic Policy Gradients (DDPG) algorithm. By applying to simulated RoboCup soccer, the action space contains a small set of discrete action types, each of which is parameterized with continuous variables. This algorithm is also called Parameterized Action Deep Deterministic Policy Gradient (PA-DDPG) in the MP-DQN paper [3].

However, there are many problems with these traditional straightforward solutions. For example, discretization/relaxation changes the original action space structure; and the separate optimization throws away the intrinsic connection between discrete actions and continuous parameters. Since there is generally some valuable prior between action type and action parameters, it can greatly affect the effectiveness of the agent's learning.

2) New algorithms: Thus new algorithms dedicated to hybrid action spaces were proposed. From the timeline in Fig. 2 we can also see that these algorithms emerged much later than the traditional algorithms. Xiong et al. combine the spirits of both DQN (dealing with discrete action space) and DDPG (dealing with continuous action space) and proposed a parametrized deep Q-Network (P-DQN) [2] framework that is designed for hybrid action space.

A year later a new method built on P-DQN called Multi-Pass Deep Q-Network (MP-DQN) [3] emerged, proposed by Bester et al. It modifies P-DQN to eliminate the correlation problem in P-DQN. From their experiment on Platform, Robot Soccer Goal, and Half Field Offense domain, MP-DQN outperforms P-DQN. It should also be mentioned that a naive solution called Separate Deep Q-Networks (SP-DQN) also appears in the same paper, which uses a separate Q-Network for each discrete action. This however significantly increases the computational and space complexity of the algorithm and slows it down.

III. Concept overview

It is known from the existing modular robot optimization literature that simultaneously optimizing modular robots with a mixture of discrete (module types) and continuous (parameters of each module) states can improve efficiency and results, as stated in Fig. 3. The column on the left shows the discrete space of module types. An agent can choose its desirable type and in the next step can choose the parameters from a continuous action space, shown on the right column.

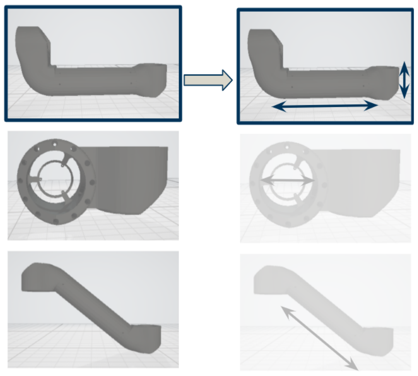


Fig. 3. Concept of hybrid optimization of a modular robot, left column: discrete optimization, right column: continuous optimization[1]

Furthermore, literature reveals that RL can be applied in the hybrid action space. Therefore we want to find a way to apply the RL hybrid method to modular robot optimization, similar to the application of Bester et al. in the Robot Soccer Goal.

It is the overall goal of the project to develop a P-DQN-based RL algorithm, which takes in the task requirement and outputs information of an optimal series of modules: the chosen modules' type and corresponding parameters (e.g. length of a cylinder) for each module. The concept can be divided into two parts:

- the creation of an environment, in which transformation between the task requirements/robot-related information (e.g. Kinematics) and state/action signal to the agent is performed, and
- the creation of agent training and testing process.

IV. Method

Knowing that the project requires optimizing both discrete module types and continuous module parameters, a RL method that can be applied to the hybrid action space is needed. From the literature research, it appears that P-DQN and MP-DQN stand out from the mentioned algorithms (section II-D). With an oriented and complete open source code, they can be used as references and adapted to the modular robot project. This section mainly describes the framework of P-DQN and MP-DQN method and their adaption to a modular robot project.

When adapting to the modular robot situation, the Agent is seen as a "Module selector". State S_t is settled as a set of the already attached module sequence, the robot's base position, and the task-related target information, E.g. target position in a point-to-point scenario. Action A_t refers to which module to choose and attach next plus the parameters of this module. Reward R_t indicates how well the modular robot with already attached modules performs in the given task and is modified during the training process for the best model result.

[1]modules' pictures generated from: https://gitlab.lrz.de/cps-robotics/robot-data/modrob-gen2

A. P-DQN and MP-DQN

The P-DQN method denotes the action-value function as $Q(a_t, s_t) = Q(s_t, k_t, x_{k_t})$, where k is the discrete action selected at time t and x_t is the set of assigned continuous parameters for the kth action. Then a deep neural network is applied to approximate Q-function [2]. The adaption of the P-DQN framework in modular robots can be seen in Fig. 4.

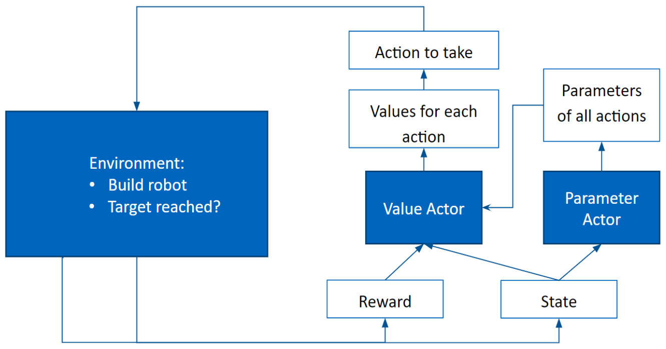
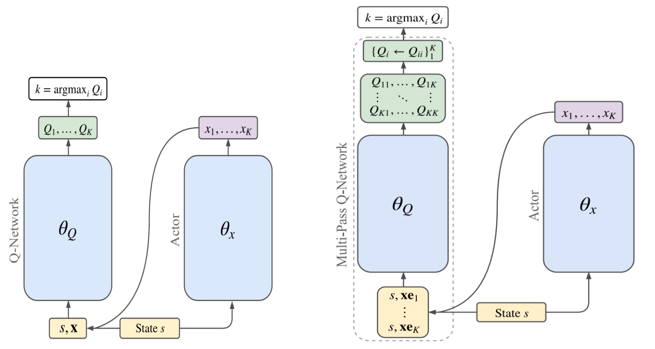


Fig. 4. Network P-DQN

After receiving action A_t, in the environment, a robot will be assembled based on previously already attached modules, which are addressed in S_t and the newly chosen action A_t. The robot is calculated and checked, if it can fulfill a given task, e.g. reach a given target or not. The current reward R_t indicates the "quality" of the new module.

In the agent part, two deep neural networks are used. The parameter actor first calculates action parameters $x_t(s_t, \theta_t)$ for every k_t, where θ denotes the parameters for the parameter actor network. In the next step the value actor $Q_t(s_t, k_t, x_{k_t}, \omega_t)$ is calculated for every k_t, where ω_t describes the parameters of the value actor network. Finally, the agents output action is calculated as $a_t = (k_t, x_{k_t})$, with $k_t = \arg\max_{k_t} Q(s_t, k_t, x_{k_t}, \omega_t)$ and the networks are optimized by estimating ω and θ by minimizing the mean-squared Bellman error via gradient descent [2].

In P-DQN, the Q-value is a function of all parameters $(x_1, ...x_K)$ of different discrete actions, regardless of the action (k_t, x_{k_t}) it chose. Bester et al. solved this problem by stating a multi-forward pass layer, in which $\frac{\partial Q_k}{\partial x_j} = 0$ for $j \neq k$ guarantees that Q_k only depends on x_k. As stated in Fig. 5, the network structure between MP-DQN and P-DQN is similar and only contains a modification over the I/O scale. By splitting the input action parameters of the respectively correlated action to the Q-network using several passes, it solves the excessive parameter dependency problem of P-DQN.



(a) P-DQN (b) MP-DQN

Fig. 5. Comparison of P-DQN and MP-DQN

V. Implementation

The Implementation of RL for modular robots is divided into three parts: First, an environment is built, in which modules are selected according to the agent's feedback and a robot is assembled. The second part is the agent network. For a large part, the network stays the same as in the original P-DQN open source code from Bester et al. [3] and only the parameters are adapted to the modular robot project. Lastly, the training and evaluation process of the agent is implemented. To end this section, we introduce the module database used for the experiments.

A. Environment

The environment contains three major functions: *Render*, *Reset* and *Step*. In *Render* the result of the test process is visualized. In every episode, a new goal position is settled and a module sequence is generated. *Reset* is being called to initiate a new episode, with initializing the module sequence and target position (for a point-to-point task). The input of *Step* is action (k, x_k), with k as the chosen module type and x_k as the given parameters for the chosen module.

In *Step* a robot is generated with the already existing state, namely the module sequence stored for this episode, and the action it takes. It returns a tuple $(observation, reward, done, info)$, with $observation$ as state passing to the agent, $reward$ is defined as in Algorithm 1. It's proportional to the remaining distance between the target position and the end effector position, with a punishment of large mass and module numbers. EE refers to the end effector of a robot, $position(.)$ refers to the Euclidian Coordinate of the target/base/end effector, and $\#module$ states the number of modules in the current assembly. Once the new state of the environment has been computed, it's checked whether the state is terminal, i.e. whether the robot can complete the task or the maximum step for each episode has been reached, and $done$ is set accordingly. $info$ refers to additional information about the assembly, in the point-to-point task it is stated as the end effector location.

Algorithm 1 Reward for Modular Robot with Point-to-Point Task

```
if robot can reach target then
    $reward = 50 + \frac{300}{2^{\#module-2}}$
else
    $reward = \frac{-norm(position(target)-position(EE))}{norm(position(base)-position(target))}$
end if
$reward = reward - 2 * (\#module - 2)$
$reward = reward - mass * 5$
if collision then
    $reward = reward - 30$
end if
```

B. Agent

The same network structure as Bester et al. research is used. The parameter actor uses three fully connected layers with ReLU as the main path and is supplemented by a passthrough layer to avoid instability, with the hidden layer size being 128 and 64 respectively. It takes the state value as input. The output is (k_t, x_{k_t}) for all possible next actions k_t and the output is fed into the value actor. The value actor is simply a two-layer fully connected neural network with a hidden layer size of 128. The weights of the passthrough layer are initialized with a univariate Gaussian distribution of mean 1 and variance 1, and all the other weights and bias is initialized using Kaiming initialization [13]. The output of the agent is obtained by combining the output of the parameter actor and the value actor.

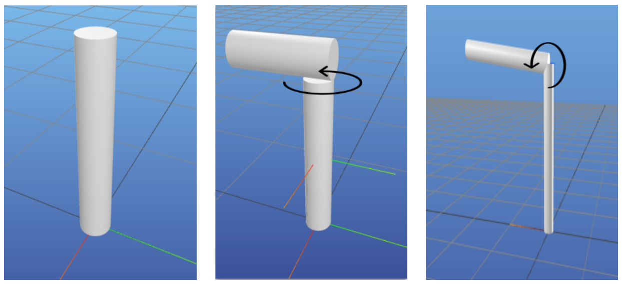
C. Training and evaluating process

In every episode, the task is initialized and a modular robot is generated. One episode contains several steps, within every step one module is selected and added to the current module instance sequence.

Experience replay is used to randomize the data and remove the correlation between the steps, which means we update Q-value through random sampling from the previous transitions during the training process. Here, the replay memory is set to 300,000 and the batch size to 128. After every 30 training episodes, 10 evaluating episodes are executed and the average will be taken as the evaluation value. Additionally, we use several environment wrappers to fix the input and output format of the action and the observation and scale to suit the agent.

D. Module database

In the implementation, we use the phrase *module database* to refer to a module library from which different types of modules can be selected. The module database used for the below experiments is made up of three modules: an I-shape cylinder module with no joints, an L-shape module with a rotational joint between the two cylinder body, which can rotate around the Z-axis, and another L-shape module that can hinge-like rotate around X-axis. The visualization of the module database is shown in Fig. 6.



(a) I-shape module (b) L-shape module (c) Hinge-like L-shape module

Fig. 6. Visualization of module database: 6(a) with no joint, 6(b) with rotational joint around the length of downside cylinder, pointing vertically to the horizontal plane of the visualization space, 6(c) with rotational joint around the diameter of downside cylinder, pointing outside paper

VI. Result

For the prototype testing process, the task is chosen as point-to-point: the robot's base point is fixed to $(0, 0, 0)$ in the 3D space, and the given task is for the end effector to reach the target point. The performance of the P-DQN algorithm is tested in three different scenarios: with a fixed target, a random target in a settled range, and a random target with obstacles. The below-given results are all using the P-DQN method. A comparison of P-DQN and MP-DQN method results can be found in section VII-C. Our experiment run on following hardware: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 13GB system memory on Ubuntu 18.04.4. It is implemented in python, using PyTorch [14] and OpenAI Gym [15].

Because the ϵ-greedy policy is used for choosing actions during the training process, in every step agent will pick an action randomly (with probability ϵ) or follow the policy (with probability $1 - \epsilon$) to guarantee enough exploration for sampling training data. Thus the original reward curve has a strong divergence. The probability ϵ will decrease during the training process, making the agent more dependent on policy. During the evaluating process, the ϵ-greedy algorithm will be deactivated and actions will be selected solely based on policy. The epsilon-episode curve in random target circumstance in section VI-B is shown in Fig. 7. To make the graphs more readable, training/evaluating curves in section VI are smoothed using Tensorboard. The transparent blue/light pink curves in the background are the original curves for the training/evaluating process.

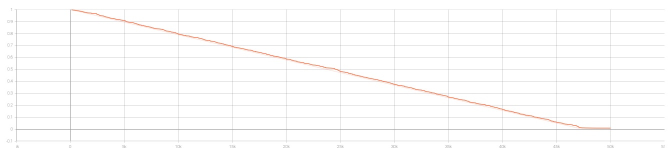


Fig. 7. Epsilon decrease along the training process in the random target circumstance: when epsilon decrease, the agent will rely more on policy. When $\epsilon = 0$ there will be no random choice at all. In our case, the ϵ value uniformly decreases from 1 at the beginning to 0.01 in episode 47000.

A. Fixed target

In this scenario, the target point is settled as the same for every training episode. A visualization of the generated example is shown in Fig. 8. The green point refers to the fixed target, and finally, the algorithm can find a globally optimal solution to reach the target, which is a single Hinge-like L-shape module with a small mass.

The reward-episode curve is shown in Fig. 11. It's trained for a complete 50,000 episodes process and applied with the early stop method, which means the training process is terminated after 300 consecutive evaluations with the evaluation reward not exceeding the previous best performance. The maximum reward (orange curve) is the biggest reward it can achieve in the best circumstance, namely when the target can be reached in one step with one module and without redundant shape (not going beyond the target and then turning back). The light blue curve shows the reward when not using the RL agent, instead randomly picking modules and parameters. From the evaluating curve, it can be seen that after around 2,000 episodes agent starts to choose one-step solutions, and after 6,000 episodes it will always find the maximum reward solution as shown in Fig. 8.

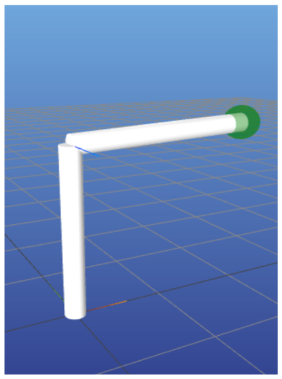


Fig. 8. Visualization of a robot with fixed target result

B. Random target

In this scenario, the target point is settled differently for every training episode. It is chosen randomly from a space set. The goal of setting up this experiment is to test whether the algorithm can learn to find the optimal solution for any target position in a fixed region. A visualization of the generated example is shown in Fig. 9. The green box refers to the region of choosing the target, and finally, the algorithm can find a locally optimal solution to reach the target, which is a two-module Hinge-like L-shape module, while the global optimal solution will be a one-module solution like in the fixed target scenario.

The reward-episode curve is shown in Fig. 11. It's trained for a complete 50,000 episodes process and applied with the early stop method. For comparison, we did two experiments, one with a small target region $(0.6m^3)$ and one with a large range $(2m^3)$. From the evaluating curve, it can be seen that in both experiments after around 1,000 episodes agent failed into local minimal, namely chosen two-step solution as shown in Fig. 9.

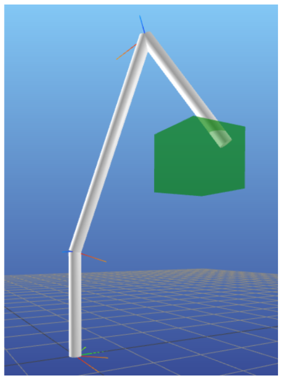


Fig. 9. Visualization of a robot with random target result

C. Random target with obstacles

In this scenario, the target point is also settled randomly, but we add obstacles to the environment to test if the agent can learn to avoid obstacles. A visualization of the generated example is shown in Fig. 10. The green box refers to the region of a chosen target, and the spheres and box are obstacles along the passing way of the fixed-target one-step solution to the target so the agent should avoid using a brutal one-step solution now. Obstacles are settled as the same for every episode. From the visualization, it can be seen that the agent choose to use a two Hinge-like L-shape module assemble to avoid obstacles and reach the target.

The reward-episode curve is shown in Fig. 13. The obstacles are settled as the target can still be reached in one step with narrowed down choices range for parameters, So the maximum reward (orange curve) is still the biggest reward it can achieve in the best one-step circumstance. From the evaluating curve, it can be seen that the result still failed into local minimal, the same as in the random without obstacles scenario.

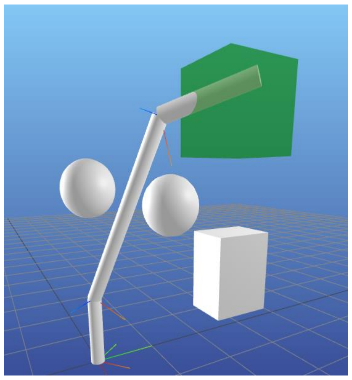


Fig. 10. Visualization of a robot with random target and obstacles in the environment

VII. Evaluation

In this section, some phenomena and possible causes of the results are mainly discussed, including local minima, comparison between different ranges of random target results, and the comparison of P-DQN, MP-DQN, and other methods.

A. Local minima

The experimental results fall into local optima in the random objective case. We believe the reason may be that there is not enough exploration of the optimal solution parameters that can support policy learning in the continuous parameter case. The optimal solution, i.e., the parameter set when a single module is available, is limited to a small number of pairs. In contrast the local optimum, i.e., the two hinge-like module case has more parameter set solutions that can reach the objective, and the possibility of being explored and learned is greater.

It can be seen from the data that the optimal solution is rarely explored before 2,000 episodes in the fixed objective case, and the corresponding policy is adopted only after 2,000 episodes. The part before the 2,000 episodes is similar to a local optimum case. Presumably, it is also possible that the optimal solution can also be reached with a random target in a sufficient amount of training time, but the geometric multiplication of the target space in the random state leads to the fact that the time required for learning may tend to infinity.

B. Comparison of random target scenarios with different ranges

Although the two results differ very little in the final evaluation reward, the training process still shows differences. In the narrow objective range, the training reward shows peaks where the optimal case was taken, while no optimal solution was explored in 20,000 episodes in the wide objective range. Possible explanations for this is because the parameter value exploration is random and when in each episode target point is updated randomly, it is difficult to fetch the parameter value of the optimal case for the particular episode's target, especially in the continuous state space and with the morphological limitations of robot module. At this point, if the range of random target points is reduced, the possibility of fetching the optimal parameters for the corresponding target will be greater, thus increasing the likelihood of obtaining a globally optimal solution.

C. Comparison of P-DQN, MP-DQN and other methods

We again use the fixed target scenario to compare the performance of P-DQN, MP-DQN, SP-DQN, and one of the traditional methods PA-DDPG. All agents use the same settings as in the MP-DQN paper [3]. The results are shown in table I. Here, we still use the same training process with the early stop method and add 1000 episodes of final evaluation after the training is stopped to obtain the final performance of the agent. We can obtain that PA-DDPG, the representative method of the traditional approach, does not lead to a globally optimal solution in this scenario, but is stuck in a local

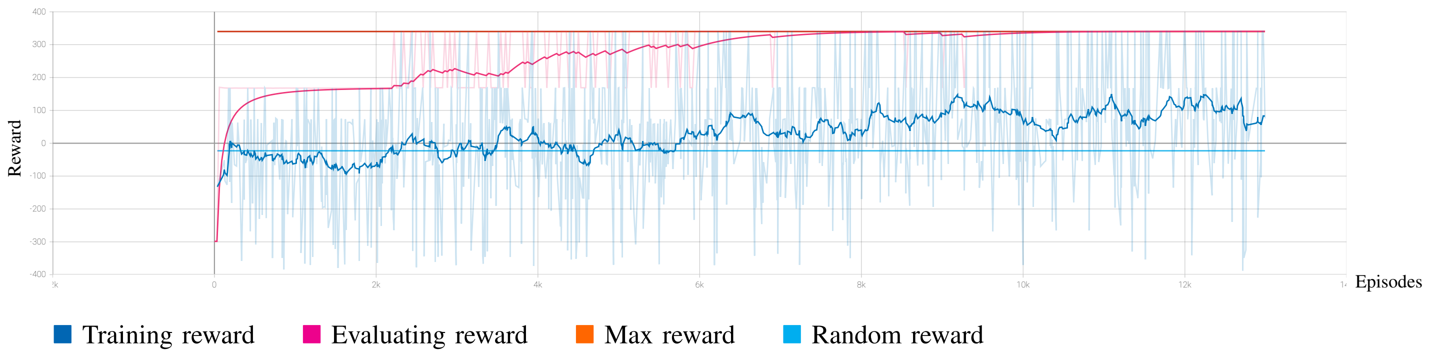
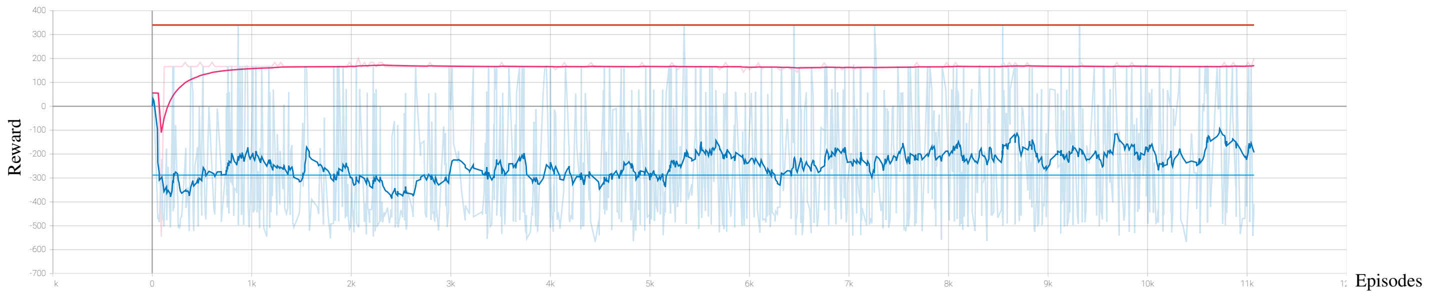
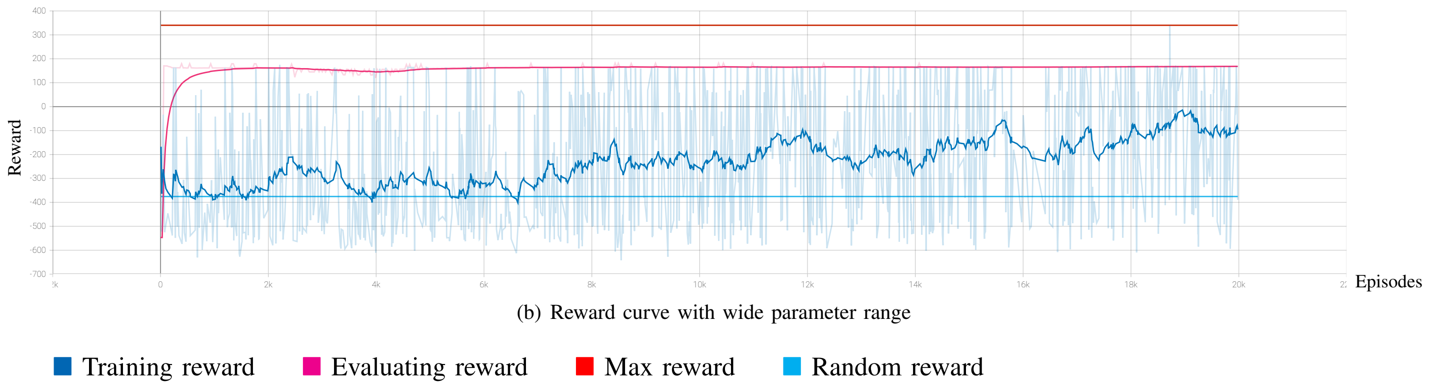


Fig. 11. Reward curves for fixed target



(a) Reward curve with narrow parameter range



(b) Reward curve with wide parameter range

Fig. 12. Reward curves for random target

minimum which can only get the evaluation reward value of 168.0. In addition, it runs extremely slowly, about twice as long as the new methods.

The new approaches can all reach the global optimum with different performances. The slight difference in the average reward indicates the difference in mass of the final robot obtained, which shows that MP-DQN can find the best solution by decoupling the action parameters. For training episodes till stabilization, however, SP-DQN shows the least training demand, slightly outperforming MP-DQN, but both greatly outperforming P-DQN. In terms of running speed, P-DQN and MP-DQN are almost identical and outperform SP-DQN by a large margin. So by combining the analysis, we can conclude that MP-DQN is the most suitable algorithm for our modular robot optimization scenario.

VIII. Conclusion

In this project, we explore RL optimization methods for hybrid action spaces on modular robots. We conduct experiments in simple scenarios and it is clear from the results that convergence and optimal results can be achieved with training at fixed target points. With a random target, the results can also converge, but fall into local optima. We also compare the performance of several RL methods hybrid action space in the simplest scenario and analyze the most suitable method for this domain.

A. Future work

To further explore the conditions and causes of the results, some possible future works could be performed, including finding better network structure and initialization, improving visualization, and adding complex scenarios.

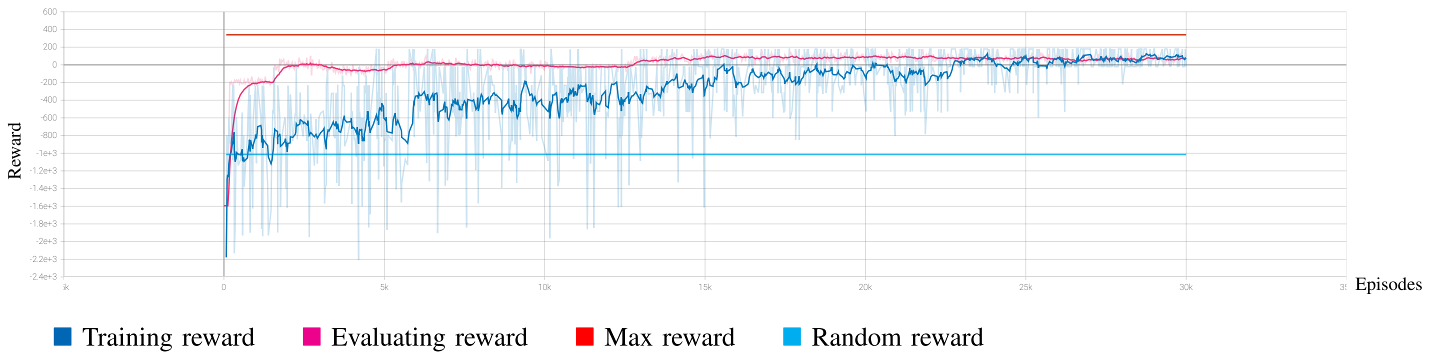


Fig. 13. Reward curves for random target with obstacles

TABLE I
COMPARISON OF DIFFERENT METHODS IN FIXED TARGET SCENARIO

Method	Reachability of Global Optimum	Average Reward of Final Evaluation	Training Episodes till Stabilization	Running Time of 1000 Episodes
PA-DDPG [5]	No	168.0	-	522.82s
P-DQN [2]	Yes	340.2	9451	***258.87s***
SP-DQN [3]	Yes	339.9	***3331***	296.53s
MP-DQN [3]	Yes	***340.5***	3991	262.23s

The table compares the performance of all the mentioned methods in the fixed target scenario. The reachability of the global optimum means the agent can give a stable one-step solution before the early stop, where the stabilization is defined as giving a one-step solution for ten consecutive intermediate evaluations (without random exploration), i.e. the reward reaches its maximum. The training episodes till stabilization then indicate the episode that the first time of ten such consecutive intermediate evaluations appears. Running time of 1000 episodes refers to 1000 training episodes without intermediate evaluations, so the run speed of each method is compared intuitively. The best performing algorithm in each column is indicated by a number in bold italics.

Using a simple database and scenario, the optimal solution consists of only one or two modules and does not reflect much of the final reward differences between similar algorithms such as P-DQN and MP-DQN. In experiments, we only consider point-to-point tasks. To include complexity, multi-goal and trajectory planning tasks can be introduced. The environment can also be set with more complicated obstacles or moving obstacles. Another possible change is to expand the module database or create an interface to existing discrete modular robot databases. Our database has only three simple modules. From the experiments, the more complex the database is, the more volatile the training process is.

REFERENCES

[1] B. Siciliano and O. Khatib, *Springer handbook of robotics*, 2016.

[2] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, “Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space,” *arXiv preprint arXiv:1810.06394*, 2018.

[3] C. J. Bester, S. D. James, and G. D. Konidaris, “Multi-pass q-networks for deep reinforcement learning with parameterised action spaces,” *arXiv preprint arXiv:1905.04388*, 2019.

[4] W. Masson, P. Ranchod, and G. D. Konidaris, “Reinforcement learning with parameterized actions,” *ArXiv*, vol. abs/1509.01644, 2016.

[5] M. Hausknecht and P. Stone, “Deep reinforcement learning in parameterized action space,” *arXiv preprint arXiv:1511.04143*, 2015.

[6] S. B. Liu and M. Althoff, “Optimizing performance in automation through modular robots,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4044–4050.

[7] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, “Modular robot design synthesis with deep reinforcement learning,” *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*, 2020.

[8] D. Ha, “Reinforcement learning for improving agent design,” *arXiv preprint arXiv:1810.03779v3*, 2019.

[9] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” *IEEE Transactions on Neural Networks*, vol. 16, pp. 285–286, 2005.

[10] G. D. Konidaris, S. Osentoski, and P. S. Thomas, “Value function approximation in reinforcement learning using the fourier basis,” in *AAAI*, 2011.

[11] J. Peters, S. Vijayakumar, and S. Schaal, “Natural actor-critic,” *Neurocomputing*, vol. 71, pp. 1180–1190, 2008.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *ArXiv*, vol. abs/1606.01540, 2016.

[2]This report is done by the team, with each individual contributing to the writing and revision of all sections.