2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

# Sample-Efficient Learning for Industrial Assembly using Qgraph-bounded DDPG

Sabrina Hoppe[1,3], Markus Giftthaler[2], Robert Krug[1], Marc Toussaint[3]

Abstract— Recent progress in deep reinforcement learning has enabled agents to autonomously learn complex control strategies from scratch. Model-free approaches like Deep Deterministic Policy Gradients (DDPG) seem promising for applications with intricate dynamics, such as contact-rich manipulation tasks. However, these methods typically require large amounts of training data or meticulous hyperparameter tuning, limiting their usefulness for real-world robotics applications. In this paper, we evaluate and benchmark our recently proposed approach for improving model-free reinforcement learning with DDPG through Qgraph-based bounds in temporal difference learning. We directly apply the algorithm to a challenging real-world industrial insertion task and assess its performance (see https://youtu.be/Z_GcNbCWE-E). Empirical results show that the insertion task can be learned despite significant frictional forces and uncertainty, even in sparse-reward settings. We present an in-depth comparison based on a large number of experiments and demonstrate the advantages and performance of Qgraph-bounded DDPG: the learning process can be significantly sped up, robustified against bad choices of hyperparameters and runs with less memory requirements. Lastly, the presented results extend the current theoretical understanding of the link between data graph structure and soft divergence in DDPG.

## I. Introduction

There is a strong interest from industry to economically automate contact-rich assembly tasks such as shaft insertion or cable plugging for small and medium lot sizes. Based on our experience, classical solutions based on compliant control require prohibitive effort in designing appropriate manipulation strategies and tuning, even though the latter might be attenuated with black-box optimization strategies [1].

In this context, to devise generalizable and scalable solutions, we are interested in Deep Reinforcement Learning (RL). It enables autonomous agents to learn complex behaviors from experience. Examples include (simulated) continuous control tasks [2] or games [3]. Although often requiring significant amounts of data, model-free RL algorithms exhibit properties which seem particularly promising for robotics. As they neither require accurate dynamics models nor try to explicitly fit them, the disadvantages of identifying models and quantifying their trustworthiness [4] are alleviated. Recent work [5, 6] shows that model-free RL can indeed scale and perform well in robotics applications.

[1]Bosch Corporate Research, Renningen, Germany {sabrina.hoppe, robert.krug4}@de.bosch.com.
[2]Bosch Center for Artificial Intelligence, Renningen, Germany.
[3]Machine Learning and Robotics Lab, University of Stuttgart, Germany, first.last@ipvs.uni-stuttgart.de


Fig. 1. Left: the Franka Emika Panda robot employed for fitting an eBike motor shaft into a ball-bearing located in the motor housing. Right: a close-up view on the shaft.

On the algorithmic side, it is interesting to note that the theoretical understanding of RL algorithms can often not keep up with their practical success. For instance, deep Q-Learning [7] and DDPG [2] belong to the most popular model-free deep RL methods today and show excellent empirical performance. However, they are still not fully understood from a theoretical perspective [8]. In particular, Q-Learning has already been characterized as unstable even with linear function approximation in the 1990s [9]. These instabilities appear in the form of difficulties in reproducibility [10] or soft divergence, i.e. predicted Q-values outside of theoretically determined intervals [8]. These issues have recently attracted increased interest in the machine learning community [8, 11, 12]. In [12], we showed the link between the issue of soft divergence and the structure of the underlying data graph and introduced a novel method to stabilize DDPG through so-called Qgraph-based bounds.

### Contributions of This Paper

So far, we only evaluated our novel method on a small-scale, simulated peg-in-hole problem with trivial physics [12]. In this paper, we take a closer look at Qgraph-bounded DDPG and demonstrate its usefulness and advantages in an applied industrial robotics task (a video for illustration is available at https://youtu.be/Z_GcNbCWE-E). In particular, the contributions of this paper are as follows:

1) for the first time, the method from [12] is applied to a full-scale robot, which learns to perform a non-trivial insertion task using real industrial parts (Fig. 1).
2) we empirically evaluate the performance of Qgraph-bounded DDPG in extensive experiments and show

a detailed comparison to classical DDPG.
3) we extend the theoretical insights from [12] in one aspect, namely we demonstrate that soft divergence also occurs for indirectly linked transitions as the length of the path to the terminal state increases.

## II. Preliminaries

We consider a standard reinforcement learning setup in which an autonomous agent interacts with its environment. At each discrete time step $t$, the agent can observe the state $s_t$ and choose an action $a_t = \pi(s_t)$ which determines the next state $s_{t+1}$. After each action, the agent receives a reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. The environment is represented as a Markov Decision Process (MDP) with states, actions, (potentially stochastic) transition dynamics and a reward function.

The expected sum over discounted future rewards when following a policy $\pi$ from state $s$ is called return $R_s^\pi = \mathbb{E}\sum_{i=0}^{T} \gamma^{i-1} r_i$, where $\gamma \in [0, 1)$ is called discount factor. To incorporate off-policy behavior, i.e. actions not originating from the current policy $\pi$, a Q-value can be defined as the expected sum over future rewards when any action $a_t$ is executed at the current time step and $\pi$ is followed from then on: $Q^\pi(s_t, a_t) = \mathcal{R}(s_t, a_t, s_{t+1}) + \gamma R_{s_{t+1}}^\pi$.

The agent's goal is to find an optimal policy $\pi^*$ which chooses the action that will maximize the return for each state. One way to achieve such optimal behavior is to identify the Q-function for the environment at hand and select the action that maximizes $Q$ at each time step. A popular way to find the Q-function is temporal difference (TD) learning, in which the estimate of $Q(s, a)$ is updated using both experience and the current estimate of the Q-function for the succeeding state:

$$Q_{\text{target}}(s_t, a_t) = \begin{cases} r_t, & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \cdot Q(s_{t+1}, \pi(s_{t+1})) & \text{else.} \end{cases} \tag{1}$$

Neural networks are a popular choice for function approximators in TD-learning. In DDPG, an actor-critic architecture is created with two networks: the actor network represents the deterministic policy $\pi$, the critic network takes a state-action pair as input and predicts the associated Q-value [2]. The actor network can then be trained end-to-end to maximize the critic's output.

## III. Related Work

### A. Industrial Shaft Fitting Task

In this work, we consider a realistic industrial manufacturing step taken from a Bosch eBike motor plant. The task is to achieve a tight fit between a shaft and a ball bearing in a motor housing. It can be considered a variant of classical peg-in-hole insertion, which belongs to the most extensively studied assembly problems in robotics. A complete review of peg-in-hole insertion methods is beyond the scope of this paper, in the following we only provide a coarse overview.

Especially in the context of industrial robotics, much effort is spent on providing hardware solutions for reliably executing insertion tasks. Successful examples are the active and controllable remote center compliance element from [13] or the vibration device presented in [14].

Torque-controlled robots as well as manipulators equipped with force/torque sensors can be used to implement force-controlled approaches to peg insertion. Often, an analytic point of view is adopted, trying to model and understand the contact physics and then deriving control strategies [15, 16]. A crucial element of these methods is the accurate estimation of contact states, which is challenging but pivotal to the success of the insertion [17]. Once a contact has been established, compliant controllers are used to perform the insertion itself [18]. Most classical methods require the specification of a sequence of contact states and careful controller design. There have been efforts to lessen the manual engineering work using black-box optimization for controller tuning [1]. Nevertheless, our experience with these approaches indicated that they are typically not robust to variations of model parameters like (static) friction or force limits. Also, they still require intricate manual strategy design and significant tuning effort to work for specific instances of the problem.

The insertion task considered in this work requires both high accuracy and significant force when solved using classical machinery: High accuracy is required to precisely align the bearing and the shaft. High force is required to overcome significant resistance of the fitting process originating from the mechanical specification and static friction effects. The accompanying video (https://youtu.be/Z_GcNbCWE-E) shows the insertion as performed by a human, which requires force greater than 10 N and a determined push to reach the final configuration. Note that the motor housing is turned upside down for the videos to obtain a better field of view around the ball bearing.

Our objective is not to investigate the entire process including grasping the shaft and positioning it in the vicinity of the ball-bearing – these steps are beyond the scope of the paper. Instead, we focus on performing the insertion step: it starts in loose, randomly oriented contact with the ball bearing, requires significant interaction force and ends with the shaft being completely inserted with some predefined accuracy.

### B. Efficient Learning for Contact-Rich Manipulation

Recent progress in reinforcement learning techniques has enabled agents to autonomously learn complex behavior as long as enough data is available [19]. To prevent costly data collections, the robotics and machine learning communities have developed a variety of techniques to

improve sample efficiency.

Since supervised learning is more efficient than reinforcement learning [20], one promising approach is to exploit supervised learning to imitate model-based solutions [21, 22]. Finding a model accurate enough for contact-rich manipulation tasks however is challenging, even for data-driven approaches [23]. Residual networks, in which only an offset to an analytical model is learned, have been shown to be a particularly efficient solution [24].

Residual policies transfer this idea to reinforcement learning [25, 26]. We employ a residual formulation to infuse general knowledge about insertion tasks.

Only few model-free reinforcement learning approaches have addressed industrially relevant tasks and often make additional assumptions such as the availability of CAD models [22, 27, 28]. In [29], a peg insertion task is learned from discrete actions in a Q-learning formulation using LSTMs.

### C. Qgraph-bounded DDPG

At the core of our solution is DDPG, a model-free method that can handle continuous state and action spaces. Since even linear function approximators can easily diverge in TD learning settings [9], highly non-linear neural networks can lead to significant instabilities in the learning process [8]. Qgraph-bounded DDPG (QG) [12] suggests to counteract these instabilities using a so-called data graph, from which a Qgraph can be derived and associated lower bounds on the Q-values can be computed. Those bounds can be enforced during TD-learning, which in return stabilizes the training process.

Data from the agent's interaction with the environment is stored in a replay memory, which can be thought of as a list of transitions $(s_t, a_t, s_{t+1}, r_t)$ with state-action pairs, the state that is reached, and the reward that was received. Alternatively, this data can be represented as a graph where the nodes correspond to states and edges represent actions [30, 31]. In [12] it is shown that the structure of this data graph is directly linked to soft divergence in Q-learning: Transitions ending in terminal states are least likely to cause soft divergence, because in this case, TD learning reduces to supervised learning (cf. the first line in Eq. (1)). Similarly, transitions towards states that are indirectly connected to a terminal state, i.e. through few further transitions, are less likely to cause divergence than states which are not connected to a terminal state at all. Loose ends can, for instance, appear when an episode is finished due to a timeout but the agent has not reached a terminal state.

In QG, a subgraph of the data graph is extracted such that under the assumption of completeness, all associated Q-values can be computed analytically. This subgraph, annotated with its Q-values, is referred to as Qgraph. The Q-values in the Qgraph are exact for the subproblem derived from the data graph, but do not transfer directly to the original learning problem. Instead, they are lower bounds to the Q-values for the original MDP. In theory, this only holds for deterministic environments, but empirically the method has also been applied to stochastic settings in [12]. Further, it was shown that additional lower and upper bounds can be derived a priori, e.g. based on the domain of the reward function. In TD-learning, lower and upper bounds (LB/UB) can be enforced in the so-called Q-target from Eq. (1) by clipping as follows:

$$Q_{\text{target}}(s_t, a_t) = \min\left(\text{UB}, \max\left(\text{LB}, Q_{\text{target}}(s_t, a_t)\right)\right).$$

In many cases, the structure of the data graph can be artificially enhanced through zero actions. These are actions that do not change the robot state, e.g. by applying zero velocity. If such a zero action is known, it can be artificially added to each state the agent has encountered so far. This creates self-loops in the data graph and can be used to eliminate loose ends, i.e. non-terminal states from which the agent has never executed any action. Loose ends can for instance appear when an episode is finished due to a timeout but the agent has not reached a terminal state, and have been linked to soft divergence in [12]. Using zero actions, loose ends can be turned into disconnected states through self-loops and thus, new or potentially tighter lower bounds in the Qgraph can be derived which helps to stabilize learning.

## IV. Method

### A. Problem Formulation using a Residual Policy

One of the major drivers for sample complexity in many reinforcement learning problems is exploration [32]. In general, adding prior (possibly domain-specific) knowledge, can speed up this phase [33]. If the character of the task is known, e.g. in our case a shaft insertion with a dominant direction of force, this knowledge can be efficiently incorporated into the problem through residual policies [25, 26]: the agent then does not learn the full behavior from scratch but an addition to a fixed policy. In this spirit, we model the insertion task as an MDP with the following definitions.

1) States: a state is defined as $(\hat{f}_x, \hat{f}_y, \hat{\tau}_x, \hat{\tau}_y)$, which are the estimated contact forces and torques in X and Y direction in the endeffector frame, c.f. Fig. 1. Torques along the Z axis correspond to a rotation around the shaft's symmetry axis and are omitted.

2) Actions: the actions are formulated as task-space wrenches and consist of a constant and a residual policy component from the MDP:

- the constant part of the policy exerts a force $f_z = -15$ N in z-direction of the endeffector frame, c.f. Fig. 1.
- the residual actions as defined in the MDP consist of torques $[\tau_x^r,\ \tau_y^r]$ along the x and y axes of the endeffector frame and are computed directly as the

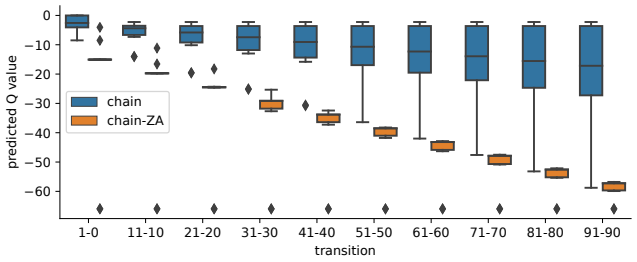



Fig. 2. Distributions of predicted Q-values by classical deep Q-learning on transitions in a chain of 100 states; including zero actions (orange) or not (blue). Transition 'X-Y' represents the transition from X to Y. Adding zero actions helps to reduce the variance significantly.

output of the actor network scaled to the interval $[-3, 3]$ Nm.

Combined, a feedforward wrench $\zeta^{\text{tool}} = [0\ 0\ f_z\ \tau_x^r\ \tau_y^r\ 0]$ is obtained and set as reference to a hybrid task-space force/impedance controller [34] together with the current endeffector pose. The controller position gains are constantly set to 250 N/m in X and Y direction and 0 N/m in Z. The orientation gain is set to be 4.0 Nm/rad in all directions.

When executing the policy, an action is considered completed once the robot's endeffector reaches a steady state with velocities below a predefined threshold. In essence, this leads to the robot applying constant force in z-direction of the shaft, while the residual policy allows to apply torque to the shaft and rotate the endeffector in space.

Lastly, the controller allows to impose a limit onto the orientation of the tool. In this work, we limit the maximum tool tilt to be equal to $\frac{\pi}{4}$ w.r.t. the horizontal ground plane, which essentially allows the learning algorithm to safely explore all end-effector orientations within a cone of $\frac{\pi}{2}$ opening angle w.r.t. the table.

3) Reward: We investigate two different reward functions:

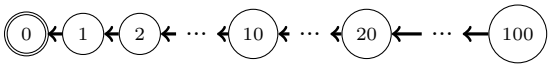
- Sparse reward: in this setting, we use a reward of $r = -1$ for each transition and $r = 0$ if the terminal state is reached.
- Dense reward: the reward is considered proportional to the distance error between the current endeffector pose and a target endeffector pose (corresponding to a fully inserted peg). The distance error in position $\Delta^P$ is computed as the $l^2$ norm of the Euclidean position difference vector. The distance error in orientation $\Delta^R$ is computed as the $l^2$ norm of the angle-axis error in x and y rotation, since the insertion task is invariant to z orientation. The combined reward is then computed as

$$r = r_P + r_R = \frac{1}{2}\left(\exp\left(-\frac{\Delta^R}{\sigma_R}\right) + \exp\left(-\frac{\Delta^P}{\sigma_P}\right)\right) - 1$$

with manually tuned scaling factors $\sigma_P = 0.015$ and $\sigma_R = 0.7$. This formulation guarantees that $r$ is always in $[-1, 0]$.

## B. Motivation for Bounded DDPG

Employing the residual policy, the agent reaches the goal in many episodes – so instead of loose ends or disconnected transitions, the predominant type of transitions in our problem are indirectly connected ones. In [12], those were evaluated in a toy example and did not cause significant divergence. In our setting, however, the path length between initial- and target state can easily reach high numbers, up to 1,000. After having observed soft divergence of classical DDPG frequently in these cases, we designed an extended toy example to demonstrate this type of soft divergence in a principled way. The example considers the following chain




in which 100 states are linked as a chain that leads to the terminal state zero. This is an abstraction of the states that would be added to the replay memory after a long trajectory which ended at the goal state. If zero actions are known, the graph could be enhanced by self-loops at each node. Assuming that each transition is associated with reward $-1$ and there is a fixed trivial policy following the chain transitions, we trained a DDPG-like critic to approximate Q-values based on these chain transitions only. We compare the original dataset ('chain') against the enhanced dataset ('ZA') in which self-loops exist in addition to each state in the chain. Note that we only changed the data, not the training algorithm.

As the results in Fig. 2 show, Q values predicted by classical deep Q-learning cover an increasing interval as the distance of states to the terminal state grows. Adding zero actions already helps to reduce this variance significantly. Yet, the predictions including zero actions are negatively biased: for instance, the true Q-value for the transition from state one to state zero is 0, but deep Q-learning predicts a value around $-15$.

We take these results as a motivation to apply QG to our industrial task and endorse the method's advantages in Section VI. More precisely, we apply the following items in the QG conditions:

1) Qgraph-based lower bounds, enforced in TD targets
2) a priori lower and upper bounds based on the minimum and maximum reward: $\frac{r_{\min}=-1}{1-\gamma} \leq Q \leq \frac{r_{\max}=0}{1-\gamma} = 0$,
3) zero actions which here correspond to zero residuals.

# V. Experimental Setup

All experiments in Sec. VI followed the same setup: The training phase consisted of 40 episodes. To start an episode, the endeffector was manually set to one out of eight initial poses with different inclinations (see video

attachment). Each episode was stopped when either the target state or a maximum of 1000 steps was reached. We implemented a pose-based heuristic to verify whether the target state is reached, but always confirmed this by detailed inspection and manual feedback because of possible slippage between shaft and gripper fingers. The network was trained after every cycle consisting of 20 steps; the number of training iterations after each cycle is one of the hyperparameters investigated in Section VI-B.

The test phase consisted of eight more episodes also covering all initial positions. A test episode was stopped after 200 steps if the target is not reached, the remaining setup remained unchanged to the training phase.

### A. Network Details

All networks were implemented in Tensorflow. Both the actor and critic network consist of three fully connected layers, where the two hidden layers contain 100 nodes. The actor network has tanh-activations on all layers and a two-dimensional output; all weights were initialized from a Glorot uniform distribution. The critic network has ReLU activations on the first two layers and no non-linearity on the one-dimensional output; the weights are initialized from a He uniform distribution. The forces and torques which served as state descriptors to the critic network were linearly scaled such that all values were in $[-1, +1]$. For optimization, the Adam optimizer was used – different learning rates and the number of training iterations per cycle were tuned on a grid of hyperparameters as described in Sec. VI-B. Following the argumentation in [8, 12], no target networks were used since they are known to delay but not prevent divergence.

### B. Robot Control

All experiments were performed on a Franka Emika Panda CoBot, where we controlled the joint torques at 1 kHz using a custom control toolchain. Throughout this work, for rigid body kinematics and dynamics we employed the open-source library 'Pinocchio' [35]. The required inertial parameters of our Panda robot were identified using an LMI-approach as presented in [36]. The end-effector contact wrench was estimated at 1 kHz real-time using an Extendend Kalman Filter-based disturbance observer implementation taken from [37]. The gripper was controlled to grasp the shaft with a constant gripping force. In order to ensure a safe grasp, we used custom-printed finger tips shaped such that a variety of cylindrical objects can be centered and grasped robustly (see Fig. 1).

## VI. Results

To validate the theoretical claims from [12] on our industrial force-fitting task, we evaluated the following aspects of the learning system: random baselines, robustness to hyperparameters, reproducibility of results, sparse rewards, limited replay memory capacity and generalization capability.

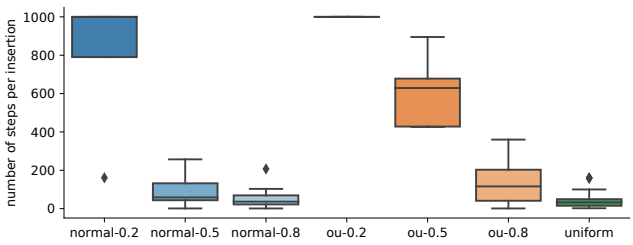



Fig. 3. Random baseline performance: distribution of the number of steps per episode for different random actions. Each episode was stopped after 1000 steps if not successful and the experiment consisted of up to 3000 steps.

Due to the stochasticity of the experiments, we ran each experiment three times with different random seeds. All plots illustrating these results show the mean as a solid line surrounded by a shaded area representing the standard deviation of the mean estimator, i.e. $\frac{\sigma}{\sqrt{n=3}}$. One experiment took between 30 and 180 minutes, in total the results in the following sections sum up to approximately 60 hours of real world interaction.

### A. Task Difficulty and Baselines

As mentioned in Sec. III-A, significant force and precision is needed to fit our shaft into the ball bearing (see video attachment). On the other hand, the constant part of our residual policy serves as a strong prior for this type of task. We therefore first evaluated a number of random baselines: instead of the actor net output, the residual policy consists of randomly sampled actions in the same output range. We compare uniform sampling and two of the standard noise processes for exploration in reinforcement learning [32, 33], namely Gaussian noise ('normal') and Ornstein-Uhlenbeck ('ou'), both with different $\sigma$[1].

As Fig. 3 shows, the uniformly sampled random actions show the best performance and solve the task in 32 steps on average.

### B. Sample Efficiency and Robustness to Hyperparameters

To obtain a broad overview of learning performance, we tuned those hyperparameters that are most related to sample efficiency on a grid: learning rates for actor and critic networks, as well as the number of training iterations per cycle.

We tested learning rates in $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$ for the critic and used one tenth of this learning rate for the actor. In prestudies a smaller learning rate for the actor seemed advantageous and confirms results from [12]. We further used either 10 or 50 training iterations per cycle.

[1]for Ornstein-Uhlenbeck noise, we use $\theta = 1$ and dt $= 0.01$

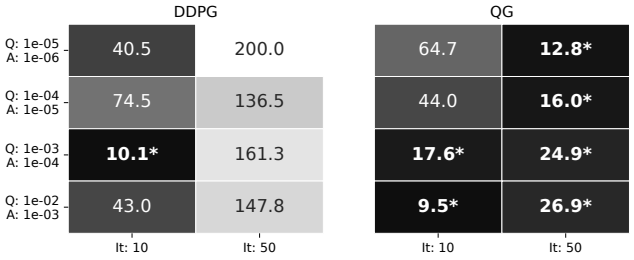



Fig. 4. Performance comparison on full grid of hyperparameters. Lower (darker) is better, entries beating all random baselines are highlighted by *. While classical DDPG outperforms the random baseline in just one out of eight cases, QG-DDPG achieves this in six cases.

Each of these eight combinations of hyperparameters was tested with 3 random seeds and both algorithms, leading to robot interactions of approximately 48 hours for this particular experiment. Fig. 4 shows the mean number of steps needed to successfully complete the task at test time for each combination of hyperparameters. Bearing in mind that the best baseline from Sec VI-A solved the task within 32 episodes, one can see that vanilla DDPG outperforms uniformly sampled action under only one particular set of hyperparameters. QG, however, performs better than any random baseline in 6 out of 8 cases.

For closer inspection, Fig. 5 depicts learning curves for the most favorable and unfavorable hyperparameters for both algorithms and plots the development of train and test performance. The shaded area represents the standard deviation of the mean estimator for performance during training episodes, the intervals on the right show the same confidence interval for test time results.

For the best case hyperparameters, we can see that both algorithms' test time performances are quite close, which is in line with the findings in [12]. Interestingly, the variance during training is lower for QG, potentially indicating higher reliability and reproducibility.

For the worst case hyperparameters, one can observe two effects: First, DDPG does not solve the task even once (at 200, the episodes were stopped if not successful). QG also decreases in performance but still solves the task.

To verify whether the continuous decrease in performance for the DDPG worst case is the effect of soft divergence, we analyzed the mean Q-value over time. The result in Fig. 6 shows that only for DDPG and unfavorable hyperparameters the Q-values diverge over time, while even bad trials of QG do not lead to divergence.

The dotted green line illustrates anecdotal results from a single run of QG in an extended setting where not only the initial shaft orientation was changed, but also the ball bearing orientation changed iteration after 4 episodes (and every second episode at test time). This evaluation is shown in the video attachment (https://youtu.be/Z_GcNbCWE-E).

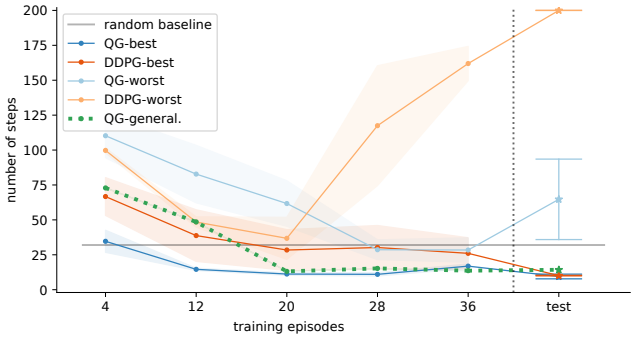



Fig. 5. Best and worst case performance for vanilla DDPG and Qgraph-bounded DDPG (QG). The x-axis shows the number of episodes and at each tick, the performance of 8 episodes has been averaged. The y-axis extends to 200, which is the worst possible test time performance in our setting. The green dotted line illustrates QG's performance on a more general and time-consuming task, where the orientation of the motor housing is changed for every fourth training episode and every second test episode.

## C. Robustness to Sparse Rewards and Limited Memory

To successfully apply reinforcement learning in practice, robustness is not only desirable w.r.t. hyperparameters but also regarding other design choices. Exemplarily, we here assess a drastic change in the reward function to sparse rewards, and a replay memory buffer that is limited to only 300 transitions. Sparse rewards are a natural formulation for our setting because they reflect more precisely our evaluation criterion (the number of steps) and at the same time circumvent all issues related to reward shaping because the endeffector pose of the robot only partially reflects the shaft pose. Limited memory availability is particularly interesting in industrial robotics as it creates a setting that is closer to the requirements of embedded AI.

Fig. 7 summarizes the results for both sparse rewards and limited memory capacity under their respective best hyperparameter configurations. We can observe that QG still performs better than random on average for both settings while DDPG does not. Additionally, QG keeps the relatively low variance in performance, while the variance for DDPG increases significantly compared to its peak performance as in Fig. 5.

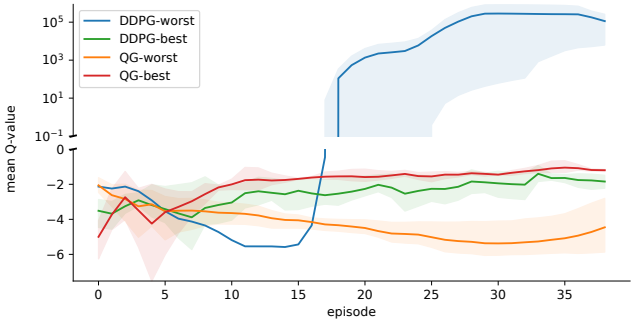



Fig. 6. Evolution of mean Q-values over training episodes for the same hyperparameters as in Fig. 5. Only DDPG diverges under bad hyperparameters while QG-DDPG is robust against those. The line represents the mean over all trials, the shaded area spans the full range between minimum and maximum.

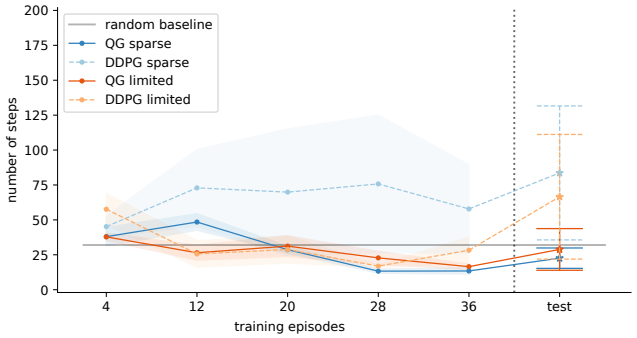



Fig. 7. Robustness to changes in the learning setting: sparse rewards (blue) and limited memory capacity (orange) for vanilla DDPG (dashed) and QG (solid lines). Both axes are scaled as in Fig. 5 for comparison.

## VII. Conclusion

Targeting an industrial force fitting task, we have chosen an efficient formulation of a reinforcement learning problem using residual policies. Applying classical DDPG in this setting tends to lead to soft divergence. We therefore extended the toy example introduced in [12] to accommodate the specific types of data graph structures originating from residual policies and could demonstrate a novel case of soft divergence. With more than 60 hours of real-world interaction, we have provided empirical evidence that Qgraph-bounded DDPG (QG) can indeed prevent this type of soft divergence and exhibits a series of further advantages: A grid search over hyperparameters has revealed that QG reaches good performance six times more often than vanilla DDPG. Results for one given set of hyperparameters differ less for QG, indicating improved reproducibility. QG is also able to deal more gracefully with further changes in the learning setup, such as sparse rewards and limited replay memory. We showcase the generalization capabilities of our approach to varying ball bearing orientations, for which the algorithm autonomously discovers an appropriate insertion strategy.

Interesting questions for future work arise in several directions: For a real-world industrial assembly process, the shaft insertion may have to be integrated with other steps such as grasping the shaft or positioning the motor housing. In particular it would be interesting to explore the limits of uncertainty from imprecise grasps that the system can learn to deal with. It is an open question to which degree the results in this paper generalize across different assembly processes; and whether manual tuning for the transfer between processes can be reduced. Data graphs as used here work on the level of raw observations, but may be extended in the future to integrate state aggregation or different levels of hierarchy. Our understanding of convergence in model-free off-policy deep reinforcement learning is still limited – for instance how the specific MDP formulation interacts with which hyperparameters and convergence properties. This lack of theoretically grounded insights in turn clearly leads to increased tuning efforts for practitioners. We also leave the comparison of QG-DDPG to further model-free methods for future work.

## VIII. Acknowledgements

The authors would like to thank Bosch eBike Systems for providing the motor components required for the evaluation of the industrial use case.

## References

[1] L. Johannsmeier, M. Gerchow, and S. Haddadin, "A framework for robot manipulation: Skill formalism, meta learning and adaptive control," in Proc. IEEE ICRA, 2019, pp. 5844–5850.

[2] 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," arXiv preprint arXiv:1509.02971, 2015.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013, NIPS Deep Learning Workshop 2013.

[4] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," arXiv preprint arXiv:1906.08253, 2019.

[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel et al., "Soft actor-critic algorithms and applications," arXiv preprint arXiv:1812.05905, 2018.

[6] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," Science Robotics, vol. 4, no. 26, 2019.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. 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. 529–33, 02 2015.

[8] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," arXiv preprint arXiv:1812.02648, 2018.

[9] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in Machine Learning Proceedings 1995. Elsevier, 1995, pp. 30–37.

[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[11] J. Achiam, E. Knight, and P. Abbeel, "Towards characterizing divergence in deep q-learning," arXiv preprint arXiv:1903.08894, 2019.

[12] S. Hoppe and M. Toussaint, "Qgraph-bounded q-learning: Stabilizing model-free off-policy deep reinforcement learning," arXiv preprint arXiv:2007.07582, 2020.

[13] A. Rueb, "Compensating device for a handling unit and handling unit comprising the compensating device," Patent US2018207811 (AA), 2018.

[14] S. Kilikevičius and B. Bakšys, "Experimental investigation of vibratory peg-in-hole insertion for robotic assembly," in Vibration Problems ICOVP 2011, J. Náprstek, J. Horáček, M. Okrouhlík, B. Marvalová, F. Verhulst, and J. T. Sawicki, Eds. Dordrecht: Springer Netherlands, 2011, pp. 621–627.

[15] H. Bruyninckx, S. Dutre, and J. De Schutter, "Peg-on-hole: a model based solution to peg and hole alignment," in Proceedings of 1995 IEEE International Conference on Robotics and Automation, vol. 2. IEEE, 1995, pp. 1919–1924.

[16] Y. Li, "Hybrid control approach to the peg-in hole problem," IEEE Robotics & Automation Magazine, vol. 4, no. 2, pp. 52–60, 1997.

[17] Y. Fei and X. Zhao, "An assembly process modeling and analysis for robotic multiple peg-in-hole," Journal of Intelligent and Robotic Systems, vol. 36, no. 2, pp. 175–189, 2003.

[18] T. Lefebvre, J. Xiao, H. Bruyninckx, and G. De Gersem, "Active compliant motion: a survey," Advanced Robotics, vol. 19, no. 5, pp. 479–499, 2005.

[19] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," The International Journal of Robotics Research, vol. 37, no. 4-5, pp. 421–436, 2018.

[20] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274, 2013.

[21] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1334–1373, 2016.

[22] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, "Learning robotic assembly from cad," in Proc. IEEE ICRA, 2018, pp. 1–9.

[23] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Learning data-efficient rigid-body contact models: Case study of planar impact," arXiv preprint arXiv:1710.05947, 2017.

[24] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," arXiv preprint arXiv:1710.04102, 2017.

[25] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," arXiv preprint arXiv:1812.06298, 2018.

[26] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in Proc. IEEE ICRA. IEEE, 2019, pp. 6023–6029.

[27] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," arXiv preprint arXiv:1906.05841, 2019.

[28] F. Wirnshofer, P. S. Schmitt, W. Feiten, G. v. Wichert, and W. Burgard, "Robust, compliant assembly via optimal belief space planning," in Proc. IEEE ICRA. IEEE, 2018, pp. 1–5.

[29] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 819–825.

[30] H. Dong, J. Mao, X. Cui, and L. Li, "Explicit recall for efficient exploration," 2018.

[31] G. Farquhar, T. Rocktaeschel, M. Igl, and S. Whiteson, "TreeQN and ATreec: Differentiable tree planning for deep reinforcement learning," in International Conference on Learning Representations, 2018. [Online]. Available: https://openreview.net/forum?id=H1dh6Ax0Z

[32] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," arXiv preprint arXiv:1706.01905, 2017.

[33] S. Hoppe, Z. Lou, D. Hennes, and M. Toussaint, "Planning approximate exploration trajectories for model-free reinforcement learning in contact-rich manipulation," IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 4042–4047, 2019.

[34] R. J. Anderson and M. W. Spong, "Hybrid impedance control of robotic manipulators," IEEE Journal on Robotics and Automation, vol. 4, no. 5, pp. 549–556, Oct 1988.

[35] J. Carpentier, F. Valenza, N. Mansard et al., "Pinocchio: fast forward and inverse dynamics for poly-articulated systems," https://stack-of-tasks.github.io/pinocchio, 2015–2019.

[36] C. D. Sousa and R. Cortesão, "Inertia tensor properties in robot dynamics identification: A linear matrix inequality approach," IEEE/ASME Transactions on Mechatronics, vol. 24, no. 1, pp. 406–411, Feb 2019.

[37] M. Giftthaler, M. Neunert, M. Stäuble, and J. Buchli, "The control toolbox — an open-source c++ library for robotics, optimal and model predictive control," 2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), pp. 123–129, 2018.