Using Reverse Reinforcement Learning for Assembly Tasks

Niklas Widulle
Oliver Niggemann
Professorship for Computer Science in Mechanical Engineering
Helmut-Schmidt-University
Hamburg, Germany
{niklas.widulle,oliver.niggemann}@hsu-hh.de

Abstract

Using Reinforcement Learning (RL) for assembly tasks is of high interest, both because of the high complexity of the problem and the high economic potential in its solution. Alternatively, there are approaches summarized under Assembly Planning (AP), which use reasoning, search and / or optimization methods to find assembly sequences. One strategy used in AP is assembly by disassembly: Finding a solution to the easier disassembly problem and then reversing the solution to create an assembly sequence. We take inspiration from this approach and apply it to RL: We train the agent to solve the disassembly task and use this to simultaneously teach it the assembly solution. To demonstrate the potential of this approach, we developed a simulation that features the assembly of toy bricks. Using this simulation we can show that our Reverse Reinforcement Learning (RRL) approach speeds up training of assembly processes significantly. We see this as an example of how RL approaches can benefit from inspiration from the field of planning.

1 Introduction

Reinforcement Learning (RL) has been very successful since its inception (Barto, Sutton, and Anderson 1983). Despite its obvious success there are still some drawbacks to applying RL compared to the other two major directions of Machine Learning (ML), Supervised and Unsupervised Learning (SL, UL). While SL and UL just rely on data input, RL relies in its essence on interaction with an environment. This can be costly especially if it is a real environment rather than a simulation (Yu 2018). This is exacerbated by the fact that RL tries to solve two problems at once: Exploring the state space of a system and trying to optimize a policy to give the best possible return. The trade-off between the two is known as the exploration/exploitation dilemma.

These problems make RL especially difficult to employ in problems with a long decision horizon, since more interactions are needed to reach a solution. In some cases, the goal state is already known, the problem is just how to reach said goal position. Multiple methods have been proposed to incorporate knowledge of the goal state into the search for an optimal policy, such as creating virtual starting positions around the goal position (Florensa et al. 2017) or alternating between backwards and forwards search of a solution (Edwards, Downs, and Davidson 2018).

Assembly is a problem that has the aforementioned property: the goal state, the assembled product, is known. Another interesting property of assembly is that disassembling a product is easier than assembling it since at each step less knowledge is needed to remove a part than to install it. In the field of assembly planning, this property is exploited by using assembly by disassembly: The disassembly problem is solved and then the generated solution is inverted and used as a solution to the assembly problem (Hoffman 1989; Tian et al. 2022).

Assembly planning (AP) is itself a large field of research. The subdomain of assembly planning that is of most relevance here is assembly sequence planning (ASP). It relies on predicates created a priori such as geometric or mechanical feasibility to generate assembly sequences. This can be achieved through means of classical planning or with soft computing methods since those allow to include optimization goals such as assembly time or tool changes (Deepak et al. 2018).

These static planning methods, however, have some drawbacks: They can only account for effects which are known to the planner a priori. Therefore, reviews of the field often include the information of which effects, such as stability of solutions or gripper reachability, were included in the planning approach (Ghandi and Masehian 2015). Adapting to change in the environment as well as expanding the solution to other components such as a robotic gripper is difficult with planning approaches.

The natural combination of these two approaches is to use an assembly by disassembly strategy in RL. Using UL this has been done by Nair et al. (2020). They used a self-supervised method to learn the dynamics of disassembly and then used this model to create a control sequence which solves the assembly problem. This leaves the question whether and how RL can be employed directly to learn assembly from disassembly. Just solving the disassembly problem with RL is straightforward, but transferring this knowledge to assembly is not. Inverting the sequence can work, but this solution is brittle to changes in the system and the influence of non-reversible effects. Deformable objects or gravity may have a different influence on disassembly than on assembly and may make using the inverted sequence impossible. A policy learned from disassembly may still be able to adapt to the change using the observations, though this may require further training. As such, there should be a benefit in using RL in combination with the assembly by disassembly approach.

In order to enable directly learning an assembly policy from the disassembly process it is necessary to transform the observations, actions and rewards in such a way that the resulting policy works for the assembly process. This is not trivial since each of these is different depending on the execution direction. This leads to the first research question (RQ1): How can observations, rewards and actions from a disassembly process be used to learn how to perform an assembly process?

While this process can be treated as a symmetrical operation, as was discussed previously, less knowledge is required for a disassembly step than for an assembly step. This makes the process asymmetrical from an information perspective. This leads to RQ2: How can the difficulty disparity between assembly and disassembly be exploited?

In order to answer these research questions we propose a technique we call Reverse Reinforcement Learning (RRL). Similar to Hindsight Experience Replay (HER), it is an approach that can be combined with established RL algorithms (Andrychowicz et al. 2017).

For the purpose of answering these question, a simple assembly simulation featuring toy bricks was created. In the simulation it is possible to reverse the order of operation, the same models can be assembled and disassembled. The simulation currently utilizes just discrete operations and does not incorporate a manipulator, though this could be added later. The focus currently lies on simplicity, as the simulation should demonstrate the impact of time reversal.

To summarize, this paper contains the following contributions: A simulation environment for assembly tasks. It introduces a novel way to utilize knowledge from time reversed execution in reinforcement learning. It presents an analysis of the benefit of time reversal in reinforcement learning.

This paper is structured as follows: After this introduction, an overview of relevant research regarding assembly planning and reinforcement learning is presented. The simulation environment is introduced in Section 3. The theoretical framework and the novel approach is described in Section 4. To analyse the benefit of reverse execution experiments were carried out and described in Section 5 and discussed in Section 6. The paper closes with a conclusion and outlook.

2 State of the Art

The chosen application for using RRL are assembly tasks. Planning and automating assembly tasks are active research fields on their own. ASP, sometimes also called assembly sequence generation or just AP, involves creating a feasible sequence of assembly actions. An assembly sequence can be infeasible due to different types of criteria, such as geometric, when installed components block other components, or stability, when an intermediate configuration is not stable. Other criteria include the reach of the used manipulators (Abdullah, Ab Rashid, and Ghazalli 2019). Additionally, there are soft criteria to consider which regard the efficiency of the chosen solution. Changing the used tool or reorienting the product increases costs and should be avoided. The goal can therefore be to find not just a feasible but an optimal solution (Deepak et al. 2018).

Historically ASP was a manual task in which an expert crafted a solution based on the provided information and his own experience. This manual process is time consuming and can lead to suboptimal solutions. Nowadays planning is still often done manually depending on the industry. The input for ASP can be the CAD files of the product to be assembled (Gu and Yan 1995). However, most approaches on ASP instead rely on intermediate information predicates, such as data on which parts are to be connected (liaison data), precedence constraints or tool data. This data is assumed to be available and needs to be either hand crafted or, if possible, extracted from the CAD or other information sources. Since large assemblies have numerous possible sequences, many of which can be feasible, this task can be very complex. The planning problem can be addressed by using approaches based on the Planning Domain Definition Language (PDDL) (Knepper et al. 2013). More popular however are soft computing methods since they are more focused on dealing with the optimization aspect. Among such approaches, Genetic Algorithms are the most used (Bahubalendruni and Biswal 2016; Deepak et al. 2018).

Creating the predicates which are the basis for ASP can be a manual effort or it can itself be automated. Automatic generation of precedence constraints is necessarily based on CAD files and employs some form of geometric reasoning. Morato et al. and Alfadhlani et al. use motion planning to search for a viable set of constraints (Morato, Kaipa, and Gupta 2013; Alfadhlani, Toha, and Samadhi 2019). Li et al. augment this approach with a block sequence based representation of the assembly task (Li et al. 2020).

2.1 ML for Assembly

Advances in ML have led to its application in assembly tasks. While traditional engineering approaches break down the assembly task into smaller, more approachable problems, ML based approaches usually try to solve the problem as a whole. This can include a robotic manipulator and vision systems. The most popular approaches for this include RL and Imitation Learning (IL). RL is an obvious choice since it captures the essence of the problem: An actor, the robot, uses observations, by the vision system, to achieve a goal, the assembled product. Since RL was introduced it has been significantly expanded with the use of neural networks instead of the previously used tables (Sutton and Barto 2018). This makes it possible to learn tasks even in very complex environments and observation spaces. Inspired by deep neural networks this is usually regarded as Deep Reinforcement Learning (DRL). Since RL requires a large amount of data to be successful, learning is usually done in a simulation environment. A great example of such an environment is by Lee et al., a Unity-based simulation of furniture assembly using robots (Lee, Hu, and Lim 2021). A recent survey of the use of RL in robotic assembly was created by Stan et al. (Stan, Nicolescu, and Pupăză 2020). The systematic drawback of using RL in assembly tasks is the long horizon of the task. The actor needs to perform a series of complex manipulations before the task can be successful. Intermediate rewards can mitigate the problem, but come with their own problems, i.e. setting suitable rewards at reasonable intermediate positions without compromising the solution space (Zhai et al. 2022).

2.2 Usage of Time Reversal in Policy Learning

Other RL approaches also incorporate knowledge of the goal state into the training process. Florensa et al. have developed a method for a reverse curriculum generation for RL (Florensa et al. 2017). They aim to exploit knowledge of the goal state by creating a specific training curriculum. In this curriculum the agent is tasked with reaching the desired goal from increasingly further away starting positions, thus increasing the challenge while continuously building up knowledge about the task. The intermediate starting positions are automatically generated and thereby the whole curriculum. The approach is tested and succeeds in the class of tasks it is designed for. The approach is however still very different from the one presented here, since each sub-task in the curriculum is still solved forward in time, no real reversal is performed. This is different in the work by Edwards et al., which directly incorporates backwards reasoning (Edwards, Downs, and Davidson 2018). Here the agent uses imaginary backwards states additionally to the regular forward movements. The imaginary backwards states are generated from the learned model of state differences in the forward motion. As such they are initially unlikely to be accurate, as they do not capture the specifics of the goal state. But over time they can speed up the overall process since they already include the learned, inverted dynamics of the model and apply it to an unknown region. The experience gained from the virtual backwards motions is then incorporated into the regular rollout buffer for learning. The basic assumption of this work is that real backwards experiences cannot be obtained, which differs from our setup.

Nair et al. have used time reversal for assembly, similar to this work. The approach uses self-supervised learning to generate a dynamics model which can the be used to generate a policy. The learning takes place in a reversed setting, where random policies are used to destroy an assembly. This is based on the basic observation also present in this work: It is easier to destroy and disassemble than to build and assemble. While this work shares the same motivation for using time reversal in assembly, it does not try to use RL directly (Nair et al. 2020).

2.3 Imitation Learning

If a successful solution to a problem exists in the form of expert demonstrations it is still challenging to derive a policy from these. This kind of problem is known as learning from demonstrations or imitation learning. Approaches that try to achieve this typically fall either in the category of Behavioral Cloning (BC) or Inverse Reinforcement Learning (IRL) (Torabi, Warnell, and Stone 2018; Arora and Doshi 2021). In BC the policy of the agent is trained to mimic the demonstrations. BC suffers from the major problem that any deviation from the seen demonstrations results in unpredictable and inefficient behaviour, the training is described as ”brittle” (Bratko, Urbančič, and Sammut 1995). However, there are still numerous modern and successful implementations of BC, especially for behavioural cloning from observations (Torabi, Warnell, and Stone 2018; Fang et al. 2019). An improvement in learning performance can be achieved by training a policy and collecting observations similar to the standard RL setting, but choosing the actions based on suggestions by an expert. This algorithm is known as Dataset Aggregation (DAgger) (Ross, Gordon, and Bagnell 2011).

IRL takes a different approach such that it first tries to learn the reward function from the demonstrations. Using this reward function the problem can then be solved using RL techniques. IRL can be advantageous compared to BC in that it generalizes better to previously unseen data. While a policy learned with BC might be no longer useful, the reward function acquired by IRL likely remains relevant (Arora and Doshi 2021).

Finally it should be noted that using imitation learning can result in qualitatively different solutions to problems than using RL. This can be due to the nature of learning from iteratively improved trajectories in case of RL or just optimal policies in case of purely passive imitation learning (Gros et al. 2020).

3 Simulation

The environment consists of a simulation of the task of stacking toy bricks. The simulation was developed in Unity 3D. In order to create interoperability with Python the simulation uses Unity ML Agents, though the actual implementations of RL algorithms provided by Unity are not used (Juliani et al. 2020). The Unity environment is wrapped as an OpenAI Gym and, as a basis for the approach presented here, the algorithms implemented in Stable Baselines 3 were used (Brockman et al. 2016; Raffin et al. 2019).

The simulation allows connections between toy bricks. A target configuration of an assembly can be defined as a multidimensional array, it represents the graph of connections between bricks. These brick assemblies are currently two dimensional structures. The simulation can be run both forward and reverse with the difference being that in reverse the toy brick structure starts fully assembled. Currently the simulation is set up in a way that the bricks are not moved but teleported to the connection positions. This dramatically simplifies the underlying problem, since it removes the motion and path planning aspect, but it retains the core aspects of assembly. The task is thereby reduced to a discrete action space. In the future we plan to expand the simulation to movements and to incorporate grippers.

3.1 Properties of the Environment

The essential property of assembly tasks is that assembly is more difficult than disassembly since at each step more knowledge is needed to place a part than to remove it. For this environment, we define that an assembly operation requires knowledge of which brick should be stacked on which brick in which position. The size of the full action space is

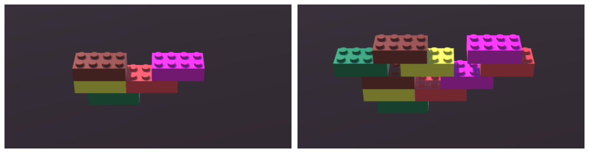
Figure 1: Brick structures inside the simulation environment. Pictured are structures with 5 bricks and 10 bricks.

therefore $n_B^2 \cdot n_p$ with n_B being the number of bricks and n_p the different assembly positions. The number of feasible actions decreases with the assembly progress since less bricks can still be connected. It is also generally infeasible to connect a brick to itself. In disassembly only the brick which should be removed needs to be selected and therefore the action space is smaller at just n_B.

The number of feasible actions during assembly is considerably smaller than the whole action space, as bricks cannot be connected with themselves and we can ignore connecting already connected bricks, as each model can be built one by one. Similarly, in disassembly the number of feasible actions shrink with each already removed brick. These criteria can also be used to simplify the RL problem using action masking, which is a standard technique that lets the agent only choose feasible actions (Huang and Ontañón 2020).

4 Approach

4.1 Short Introduction to RL

RL is motivated by finding an optimal solution for the Markov Decision Process defined as a tuple $M = (S, A, r, P)$. Here S is the state space, A is the action space and r denotes the reward function. $s_t \in S$ is a state at time step t, $a_t \in A$ is the action and $r_t = r(s_t, a_t, s_{t+1})$ is the reward for a specific state transition. $P(s_{t+1}|s_t, a_t)$ describes the transition probability between the states when executing actions.

Each episode is a sequence of states, actions and rewards $\tau = (s_0, a_0, r_0...s_T)$ which is called a trajectory. s_T is the final step of an episode with length T which therefore does not allow another action. A trajectory ends when either the maximum episode length is reached or a terminal state is encountered.

Each trajectory results in a return $R = \sum_0^T(r_t)$ which is the sum of the received rewards per episode. A policy $\pi(a_t|s_t)$ selects an action given a state. This allows to calculate the expected return when following a policy from a certain state $V^\pi(s_t) = \mathbb{E}[\sum_{t=1}^T(r_t)]$. V is called the state-value function as it assigns a single value for each state. It depends on the initially unknown transition probabilities P, the reward function r and the currently used policy π. Additionally we also define an action-value function $Q^\pi(s_t, a_t)$ which differs from V such that for the current state s_t the action a_t is chosen. Finally the advantage function $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ is defined as the benefit of choosing one action over the other options.

The objective lies in finding a policy $\pi^{opt}(a_t|s_t)$ which maximizes the expected return. In classic RL, the values of the current estimates of the value functions are stored in tables. However these methods struggle with generalizing from observations and were thus extended by using neural networks.

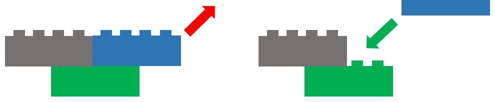
Figure 2: Illustration of the different states and actions at the time of removal and assembly of a brick. The state at time of assembly is different from the state at time of disassembly. The disassembly action is: $a_t = \{BB\}$. The assembly action: $\hat{a}_t = \{BB, GB, P\}$. BB and BG are the numbers of the blue and green bricks and P represents the desired position.

These neural networks can be employed to represent the value function or the action-value function. Their property of universal function approximators makes it possible to represent any scenario while also extending to previously unseen states. Problems arise due to instability, which can be addressed by using double NNs or other strategies (Van Hasselt, Guez, and Silver 2016; Anschel, Baram, and Shimkin 2017). The power of NNs also makes it possible to directly represent the policy as a network. The network is then updated according to the gradient of the expected return. This is the basis of the REINFORCE algorithm (Williams 1992). These policy gradient methods have then been combined with value-based methods, to create actor-critic approaches. In these, a policy is represented by one NN, the actor, and is valued according to another NN, the critic (Konda and Tsitsiklis 1999). These methods include Trust Region Policy Optimization (TRPO), Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) are among the currently most used RL algorithms (Schulman et al. 2015; Haarnoja et al. 2018; Schulman et al. 2017). PPO is adapted to RRL in this work.

4.2 Novel Approach

In assembly tasks, the amount of information needed to successfully perform a disassembly step is lower than for an assembly step. When placing a part in an assembly we need to know where exactly it should be placed, and with what other parts it should be connected. In disassembly, we only need to know what part should be removed. This makes disassembly a much easier task than assembly. The question is, how can this be exploited to simplify assembly operations?

The easiest solution to this question is to just invert the disassembly sequence and use it to assemble the product. There is, however, a substantial drawback to this approach: It cannot react to any perturbations or changes in the system. When inverting assembly tasks, some effects cannot be simply inverted: Gravity affects assembly different than it af-

Direct Training		5 bricks / 100000 timesteps	10 bricks / 250000 timesteps
Assembly	return	$0.91(\pm 0.12)$	$0.93(\pm 0.05)$
	episode length	$21.2(\pm 22.7)$	$628(\pm 480)$
RRL		5 bricks / 50000 timesteps	10 bricks / 50000 timesteps
Disassembly	return	$1.0(\pm 0.0)$	$0.96(\pm 0.05)$
	episode length	$5.0(\pm 0.0)$	$381(\pm 480)$
Assembly	return	$1.0(\pm 0.0)$	$0.82(\pm 0.3)$
	episode length	$4.0(\pm 0.0)$	$380(\pm 480)$

Table 1: The average returns and episode lengths after training. The top part shows direct assembly training with PPO. The lower part shows RRL, the performance for disassembly as well as assembly. Notice that training times were adjusted, since disassembly converges faster. The standard deviation is given in brackets. Averaged over 8 training runs, 10 evaluations per trained policy.

Algorithm 1: Reverse Reinforcement Learning

```
 1: initialize $\pi$, $\bar{\pi}$, $V$, $\bar{V}$, $s_0$        ▷ initialize the policies, starting state as well as the value functions
 2: initialize $t = 0$, $j = 0$
 3: **while** $t < MaxSteps$ **do**                               ▷ run training until the step limit is reached
 4:     **for** $j < BatchSize$ **do**                            ▷ train the policy in batches
 5:         $a_t \leftarrow \pi(s_t)$;                            ▷ choose action based on current policy
 6:         $v_t \leftarrow V(s_t)$;                              ▷ for value-based and actor/critic methods: estimate current value
 7:         $\bar{v}_t \leftarrow \bar{V}(s_t)$;                  ▷ the value of the reverse policy also needs to calculated
 8:         $s_{t+1}, r_t \leftarrow P(s_t|a_t)$;                 ▷ environment step
 9:         add $a_t, r_t, s_t, v_t$ to trajectories $\tau$       ▷ the trajectories here also include the values $v_t$ of each state
10:         add information to $a_t \rightarrow \hat{a}_t$       ▷ see: *Augmenting Action Information*
11:         add $\hat{a}_t, r_t, s_{t+1}, \bar{v}_t$ to trajectories $\bar{\tau}$
12:         $j \leftarrow j + 1$;                                 ▷ counts $j$ steps in one batch
13:         $t \leftarrow t + 1$;                                 ▷ counts total steps during training
14:     **end for**
15:     update $\pi, V$ using $\tau$                              ▷ using regular RL algorithm
16:     reverse $\bar{\tau}$ and sanitize states and rewards      ▷ see: *State Equivalence* and *Handling of Rewards*
17:     update $\bar{\pi}, \bar{V}$ using $\bar{\tau}$            ▷ again using regular RL algorithm
18: **end while**
```

fects disassembly. If parts are deformed in the process, this is also not invertible. Therefore it is crucial that not just a static inversion of the trajectory is used, but rather a policy. This should help accomplish two things: Generalize from observations, and thus be able to react dynamically to change, and be able to be retrained in order to adapt to effects that are different between assembly and disassembly.

Therefore it is crucial that a policy is generated to solve the assembly task. The disassembly task can be learned using RL and then the successfully trained disassembly policy can be used as the basis for an imitation learning approach, such as BC or IRL. While this can work, a downside lies in the training of the assembly policy: When using IRL or BC with just the successfully trained disassembly policy, a lot of knowledge from the less successful tries is wasted. Importantly they contain information about the less successful disassembly attempts and provide more knowledge about the overall dynamics of the system. It seems therefore advantageous to train the assembly policy using all knowledge from the disassembly process.

We therefore propose to train an assembly policy at the same time as the disassembly policy. More specifically, we use the disassembly trajectories to train a policy for assembly at the same time as training the disassembly policy. In contrast to standard imitation learning, we do not just rely on an optimal expert but rather use the knowledge from the disassembly trajectories while also training the disassembly policy. This is also different from DAgger, since we do not use the expert knowledge to interact with the environment but rather just rely on using trajectories gained from the disassembly task. To summarize, we use trajectories gained from training a policy on disassembly with standard RL and use these to train a policy for the assembly task. Since disassembly is easier, this is expected to speed up training significantly.

There are some major obstacles for enabling this approach. Directly using trajectories gained from disassembly to train a policy for the assembly task does not work. There are differences concerning the states, rewards and the actions themselves. Consider the disassembly of the structure in Figure 2. During the disassembly of the third brick the structure has three bricks, while during assembly of the same brick it has two. The observation is different, and it is different by the consequence of the action. The rewards have a similar problem, rewarding disassembly is counter-productive for an assembly task. The actions finally are also different. As stated before, the information needed for disassembly is lower. For the structure in Figure 2, the only information needed to remove the brick is which brick should be removed. For assembly, the information where to place the brick is missing. These issues make clear that using the disassembly trajectories to learn the assembly task is not trivial. The following sections describe how each of the previously outlined issues were addressed.

State Equivalence Consider a policy for a disassembly task $\pi(s_t)$ which chooses an action a_t at time step t. Executing the action results in reaching state s_{t+1}. For assembly, the sequence of visited states is inverted. Therefore an assembly action $\bar{a}_t$ is chosen at s_{t+1} leading to s_t. Therefore when training the assembly policy $\bar{\pi}$ the sequence of states in the trajectory needs to be reversed. Additional treatment is needed for the initial and the final state. The final state of RL is usually not visited and only the reward is collected. The final state is usually not of interest, since no further action choice depends on it. However, for the reversed task it is the initial state and very relevant. Therefore the final state of each trajectory needs to be included in order to make training the assembly task possible.

Handling of Rewards The reward also needs consideration. In most RL tasks, choosing the reward function is a design choice. For assembly two variations (and intermediaries) are plausible, getting a reward for each successful (dis-)assembly action or getting a reward for the completed (dis-)assembly. When using the first strategy the rewards of actions can be reversed similar to the state trajectories. However, when rewards are given for completing the disassembly, the reward needs to be shifted to completing the assembly. This means, that in this case, the reward is not reversed, as it needs to remain at the end of the episode.

Augmenting Action Information While disassembling, the number of choices presented are lower than during assembling. In the proposed environment, when disconnecting a brick we do not need to know from which brick we are disconnecting or from which exact position. For assembly, this information is vital. However, since the information is implicitly present during disassembly, the information is added to the trajectories which are used to train the assembly policy. To summarize, the action space of the assembly policy is larger, but each disassembly action a_t can be augmented with information gathered during disassembly to get $\hat{a}_t$, the assembly action which includes the necessary information.

4.3 Description of the Algorithm

RRL can be used by augmenting standard RL algorithms, PPO is used in this work. While RRL can be used regardless of the algorithm, there are still concepts used which need to be adapted. The pseudo code for the general case is described in Algorithm 1. Variables with a bar are for the assembly task, while variables without the bar are for the disassembly task. Note that, when using value-based or actor-critic methods, the current values v_t and $\bar{v}_t$ need to be calculated for both the disassembly as well as the assembly direction. Since the trained policies are different, their estimated value functions, V, $\bar{V}$, will also differ. Since the updates for policies and value functions are based on the gradient, it is necessary to calculate the current values based on the associated value functions. For most RL approaches, training is done in batches both for stability as well as efficient execution. PPO, which is used in this work, is regarded as an on-policy approach, meaning that the policy used for exploration is the same that is improved to solve the task. When using RRL the assembly policy is still trained off-policy as it is not used to generate exploration trajectories.

5 Experiments

All experiments have been performed on a workstation with an Nvidia Geforce RTX 3090 and an Intel Core I9-10900K. The simulation environment as well as the used Python environment is encapsulated in a Docker container for reproducibility as well as parallelisation. The code of simulation environment, the implemented algorithm, the dockerfile and the surrounding scripts are publicly available on Github[1].

In order to establish a baseline the simulated assembly task can be tackled by using the standard PPO algorithm. Figure 3 shows the training progress for 5 and 10 bricks both for assembly as well as disassembly with the achievable reward being normalized to 1. The agent is rewarded for each successful assembly action.

Table 1 shows the results after training for the different configurations. First the baseline results for using PPO to learn the assembly task directly are given. Each training was done 8 times and after training the policies were evaluated 10 times. The training durations were 100000 timesteps for 5 bricks and 250000 timesteps for 10 bricks respectively. Given are both the achieved normalized returns as well as the average episode lengths. The achieved episode lengths depend on the maximum episode length which was set to 50 for 5 bricks and 1000 for 10 bricks. The large difference between the optimal and maximum episode length results in the large standard deviation.

With RRL the disassembly task is used for training and the resulting policies are then evaluated on disassembly tasks as well as assembly tasks. The results are also given in Table 1. The achievable minimum episode length is different for disassembly than for assembly, since the disassembly environment has to include the final step as laid out in Section 4.2.

6 Discussion

The baselines for assembly and disassembly in Figure 3 show clearly that for this task, disassembly requires substantially less training duration to achieve the maximum reward. However, Figures 3 (a) and (d) also show a potential problem: With longer training times the solution can degrade in

[1]https://github.com/imb-hsu/reverse-rl

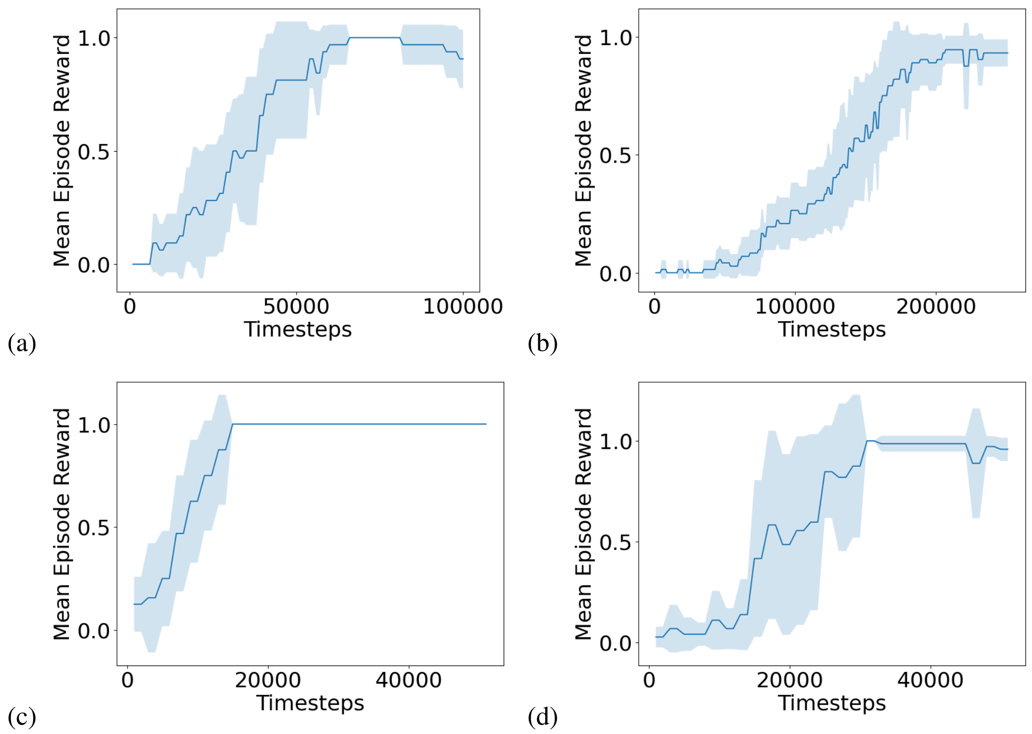

Figure 3: Training of the assembly task compared with training of the disassembly task using PPO. Each task was run 8 times, the standard deviation is overlaid over the graph. The achievable return or cumulative reward per episode is normalized to 1. (a) shows the return during training of the assembly task with 5 bricks, (b) with 10 bricks. (c) shows training of the disassembly task with 5 bricks (d) with 10 bricks

quality as the agent tries to search for other potential maxima in the reward function. One potential solution is to just save the best achieved model or to stop training once the maximum possible reward is reached. Overall this shows the large potential of using disassembly for training rather than assembly.

This potential can be levaraged using RRL, as shown in Table 1. Using RRL, it is possible to train an assembly policy using just the knowledge gained from the disassembly task. This is especially made clear by the result for 5 bricks. The result for 10 bricks is less ideal though. In disassembly, the agent reaches a reward close to that of direct training for assembly while the result for assembly is worse. This could be the result of the effect seen in Figure 3 (d), that the training already deviated from the optimum. Since assembly is the more difficult task, it is more affected by any deviation from the optimal policy. While this is a possible explanation, it is also possible that other effects are influencing this outcome. It is still clear, though, that, using RRL the training times are substantially lower than direct assembly training.

While the benefits of RRL are therefore clear, there are still pitfalls that may have an impact. The assembly policy is trained concurrently with the disassembly policy, but there is no guarantee that it converges at the same time as the disassembly policy. Due to the different complexities of the tasks it makes sense to assume that they do not converge at the same time. While additional training of the assembly policy using an optimal disassembly policy can alleviate the problem, this could take away from one of the benefits of RRL: That experience is not just gained from successful examples.

Some question could not be answered by these results, such as how the approach scales with more complex tasks. Another core question is how the approach reacts to effects that are not reversible between assembly and disassembly, such as deformable objects. Currently the approach is implemented in a discrete action space. A continuous action space, as for a example movements by a robotic gripper, will behave very differently though. In such an environment the difference in task complexity between assembly and disassembly is no longer a direct property of the action space but rather an implicit property of the environment. Whether RRL is beneficial under those conditions remains an open question.

Another open question is the performance of RRL in conjunction with other RL approaches. In this work PPO was used as the basis, which is an on-policy RL approach. Since training of the assembly policy is by nature off-policy, it would be very interesting if the performance of RRL would improve with an off-policy algorithm such as SAC.

7 Conclusion and Future Work

In this work we presented RRL: A novel approach for RL in assembly which is inspired by the technique of assembly by disassembly from the field of assembly planning. RRL uses trajectories from training of a disassembly task to simultaneously train an assembly policy. In order to make this possible the trajectories need to be adapted. This is described in detail in Section 4 and answers RQ1.

RRL was evaluated on a custom assembly simulation featuring toy bricks. The simulation clearly showed that disassembly is easier than assembly, since it requires less knowledge at each step. Using RRL we could exploit this to speed up training of assembly policies, and with that answer RQ2.

We see this work as an example of how RL can benefit from inspiration taken from planning approaches. However, it is clear that this work can only be the first step. In the future we plan to further evaluate the performance of RRL with regard to larger assemblies and more complex tasks. Especially interesting is the performance of RRL with regard to deformable objects or other non-reversible effects. We wish to bring RRL closer to actual real world use by testing it on real assemblies rather than toy bricks. For this, we also wish to test the performance of RRL with a continuous rather than discrete action space and make gripper movements possible. Finally, since RRL is used in conjunction with other RL algorithms, we want to test its performance using different approaches.

Acknowledgements

This research as part of the project KIPro is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU.

References

Abdullah, M. A.; Ab Rashid, M. F. F.; and Ghazalli, Z. 2019. Optimization of assembly sequence planning using soft computing approaches: a review. *Archives of Computational Methods in Engineering*, 26: 461–474.

Alfadhlani, A. M.; Toha, I. S.; and Samadhi, T. A. 2019. Automatic precedence constraint generation for assembly sequence planning using a three-dimensional solid model. *International Journal of Technology*, 10(2): 339–350.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Anschel, O.; Baram, N.; and Shimkin, N. 2017. Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, 176–185. PMLR.

Arora, S.; and Doshi, P. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297: 103500.

Bahubalendruni, M. R.; and Biswal, B. B. 2016. A review on assembly sequence generation and its automation. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 230(5): 824–838.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5): 834–846.

Bratko, I.; Urbančič, T.; and Sammut, C. 1995. Behavioural cloning: Phenomena, results and problems. *IFAC Proceedings Volumes*, 28(21): 143–149.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. arXiv:1606.01540.

Deepak, B.; Gunji, B.; Bahubalendruni, M. V. A. R.; and Biswal, B. 2018. Assembly sequence planning using soft computing methods: a review. *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, 233.

Edwards, A. D.; Downs, L.; and Davidson, J. C. 2018. Forward-backward reinforcement learning. arXiv:1803.10227.

Fang, B.; Jia, S.; Guo, D.; Xu, M.; Wen, S.; and Sun, F. 2019. Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, 3: 362–369.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, 482–495. PMLR.

Ghandi, S.; and Masehian, E. 2015. Review and taxonomies of assembly and disassembly path planning problems and approaches. *Computer-Aided Design*, 67–68: 58–86.

Gros, T. P.; Höller, D.; Hoffmann, J.; and Wolf, V. 2020. Tracking the race between deep reinforcement learning and imitation learning. In *Quantitative Evaluation of Systems: 17th International Conference, QEST 2020, Vienna, Austria, August 31–September 3, 2020, Proceedings 17*, 11–17. Springer.

Gu, P.; and Yan, X. 1995. CAD-directed automatic assembly sequence planning. *International Journal of Production Research*, 33(11): 3069–3100.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Hoffman, R. 1989. Automated assembly in a CSG domain. In *1989 IEEE International Conference on Robotics and Automation*, 210–211. IEEE Computer Society.

Huang, S.; and Ontañón, S. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171.*

Juliani, A.; Berges, V.-P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2020. Unity: a general platform for intelligent agents. arXiv:1809.02627.

Knepper, R. A.; Layton, T.; Romanishin, J.; and Rus, D. 2013. IKEAbot: an autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE International conference on robotics and automation*, 855–862. IEEE.

Konda, V.; and Tsitsiklis, J. 1999. Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Lee, Y.; Hu, E. S.; and Lim, J. J. 2021. IKEA Furniture assembly environment for long-horizon complex manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 6343–6349.

Li, Z.; Wang, J.; Anwar, M. S.; and Zheng, Z. 2020. An efficient method for generating assembly precedence constraints on 3D models based on a block sequence structure. *Computer-Aided Design*, 118: 102773.

Morato, C.; Kaipa, K. N.; and Gupta, S. K. 2013. Improving assembly precedence constraint generation by utilizing motion planning and part interaction Clusters. *Computer-Aided Design*, 45(11): 1349–1364.

Nair, S.; Babaeizadeh, M.; Finn, C.; Levine, S.; and Kumar, V. 2020. TRASS: Time reversal as self-supervision. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 115–121.

Raffin, A.; Hill, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; and Dormann, N. 2019. Stable Baselines3.

Ross, S.; Gordon, G.; and Bagnell, D. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 627–635. JMLR Workshop and Conference Proceedings.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International conference on machine learning*, 1889–1897. PMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Stan, L.; Nicolescu, A. F.; and Pupăză, C. 2020. Reinforcement learning for assembly robots: a review. *Proceedings in Manufacturing Systems*, 15(3): 135–146.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: an introduction*. MIT press.

Tian, Y.; Xu, J.; Li, Y.; Luo, J.; Sueda, S.; Li, H.; Willis, K. D.; and Matusik, W. 2022. Assemble them all: physics-based planning for generalizable assembly by disassembly. *ACM Transactions on Graphics (TOG)*, 41(6): 1–11.

Torabi, F.; Warnell, G.; and Stone, P. 2018. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, 5–32.

Yu, Y. 2018. Towards sample efficient reinforcement learning. In *IJCAI*, 5739–5743.

Zhai, Y.; Baek, C.; Zhou, Z.; Jiao, J.; and Ma, Y. 2022. Computational benefits of intermediate rewards for goal-reaching policy learning. *Journal of Artificial Intelligence Research*, 73: 847–896.