Learning Robot Skill Sequences with Reinforcement Learning for Off-Earth Assembly

Philip L. Møller, Valdas Druskinis, Frederik J. Christensen, Matej L. Debijadi

Department of Materials and Production, Aalborg University
Fibigerstraede 16, DK-9220 Aalborg East, Denmark
Email: pmalle18@student.aau.dk, vdrusk18@student.aau.dk, fjch18@student.aau.dk, mdebij21@student.aau.dk
Web page: http://www.mechman.mp.aau.dk/

Abstract
To enable Off-Earth manufacturing, such as on Mars or the moon, there is a great need for robots that can perform agile tasks as manual corrections and reprogramming may not be possible due to e.g. communication issues. This paper explores the possibilities for applying reinforcement learning (RL) to automate the reprogramming of robot skills, by replacing the common reliance on precedence charts with real-time collision detection of simulated robot trajectories. To prove the relevance for In-Space-Assembly (ISA), a power inverter is assembled, as stable generation and distribution of power is a predecessor to most other technologies relevant for colonization. The environment used for training an RL agent is simulated through Nvidia Isaac Gym. Experiments on trained agents are showing clear signs of learning, however, they are found to occasionally struggle with the planning of tool change. Assembly sequences from a trained agent are demonstrated and validated with a physical 7 DoF Panda robot manipulator developed by Franka Emika. This work proposes a standardized assembly board based on the power inverter while also incorporating realistic manufacturing processes.

Keywords: Robot skills, assembly sequence planning, reinforcement learning, automated reprogramming

1. Introduction

Traditional space operations have been focused around utilizing technologies and products that require little to no assembly after deployment, e.g., satellites are provided with a power source through unfolding solar panels. However, through recent years there has been an increased focus on developing technologies that allow for In-Space-Assembly (ISA) due to the increased flexibility of operations it provides. Xue et al.[1] outlines that the complexity of these technologies are proportional to the complexity of the required assembly, and therefore, technologies can be classified based on intended assembly products. Four classes, with increasing complexity, are suggested as: mating between elements, modular assembly, complex assembly and assembly from parts.

Any colonization on other planetary bodies, such as Mars or the Moon, will rely heavily on establishing In-Situ Resource Utilization (ISRU) to become self-sufficient, which has a high level of synergy with assembly-from-parts. Autonomous robotic systems are therefore highly relevant as this type of assembly will require complex and flexible technology.[2]

Optimization is generally central for all manufacturing processes, however, the problem of finding an optimal assembly sequence is an NP-hard (non-deterministic polynomial-time hard) problem, as the solution space grows with the factorial of the number of components in the assembly. This can lead to extensive computation time for an exhaustive search for the single optimal solution. By utilizing RL, an agent can be used to reduce this search time by building up experience through trial and error. [3]

2. Related Work

Previously, the problem of assembly sequence planning has been solved by systems based on some human input[4]. This human input comes in the form of information about precedence relations between objects or their constraints during the assembly[5]. Throughout the years, multiple methods (De Fazio and Whitney[6]; Sunil et al.[7]; Xu et al.[8]) have been developed to find optimal solutions based on this precedence information, however, the effectiveness of such systems would be significantly reduced for space related operations due

to e.g., communication issues preventing human input from reaching the system.

Research on automated robot skill sequence planning for assembly has also been conducted, where Rodríguez et al.[9] was able to automatically generate assembly sequences by providing product specifications to the system. This removed the necessity of precedence charts, however, the product specifications would still have to be input manually. Most recently, De Winter et al.[3] developed a system which did not require any human input related to precedence information or products, but instead required programming of the robot skills by demonstration to assemble a Cranfield benchmark. Precedence information to establish feasibility of individual assembly skills was here asserted through physics simulations.

One thing common to the works presented throughout this section is that they have been restricted in two ways. In autonomy, by reliance on precedence charts or programming-by-demonstration, or in generalization to realistic products, i.e., single process assemblies such as the Cranfield benchmark rarely occur in the industry. This work is aimed towards applying RL to a realistic assembly product, featuring more than one process, and allowing for fully autonomous sequence planning through real-time collision detection of manipulator trajectories.

In the following sections, we present the different components required for an overall solution. Section 3 will outline the background for this work, along with defining a relevant product featuring realistic processes, while Section 4 will outline the experimental setup related to these components. This will be followed by a presentation of results from experiments related to feasibility and optimality in Section 5 to which a conclusion will be given in Section 6. Some proposals for future work will then finally be outlined in Section 7.

3. Background

This section will outline the background for this work, which involves defining a relevant assembly product featuring realistic processes and the general RL framework, while alternatives to precedence charts will be discussed in the end.

3.1 Assembly Product

Various products are relevant to colonization of other planetary bodies, however, power generation and infrastructure is a general predecessor to most other necessary technologies. To safely permit both generation and distribution of power, inverters are needed to invert produced DC power into usable AC power, and are thus crucial components. Inverters also feature general assembly processes that apply to a wide variety of common products. Therefore, a simplified inverter, as seen in Figure 1, will be used as a standardized assembly board, which consists of eight components: a PCB, two MOSFET switches with heat sinks, a clock pulse generator, a transformer and a brace, requiring two fastenings with screws, to fix the transformer.

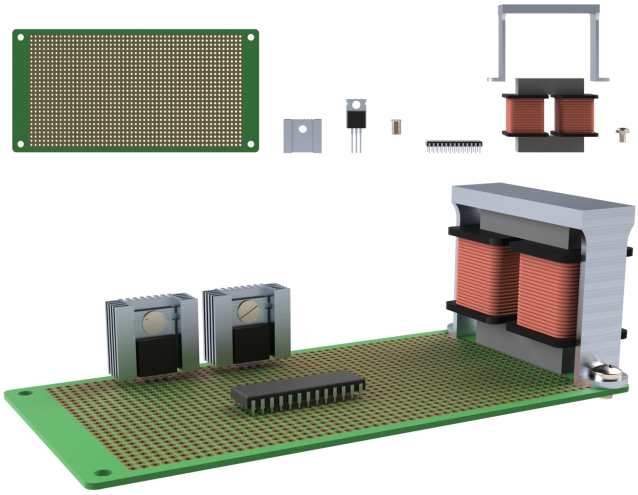

Fig. 1 Simplified inverter assembly.

The proposed standardized assembly board can also serve as a benchmark for related works, as it is paramount to have a common basis to reasonably compare performance of RL algorithms.

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique that revolves around allowing models to learn through trial and error. Through this learning process, an agent is trained to select actions that yields the greatest reward given an observed state of a given environment. The way rewards are determined for given actions can be used to encourage various behaviors of the agent. RL problems are mathematically formalized as Markov Decision Processes, defined by a tuple of the following elements (S, A, T, γ, R)[3], where:

- S is a set of all states
- A is a set of all possible actions, in this project actions will be referred to as skills

- T is a probability transition function from current state to next state when performing the action
- γ discount factor $\gamma \in [0, 1]$, measure of importance of future rewards
- R is a reward function

Probabilities completely characterize the dynamics of the environment in a Markov decision process. That is, the probability of any potential state and reward at any given time is solely determined by the state and action that came before it.

Various algorithms can be used to solve MDP's, where one of the most common is Q learning. In the Q learning algorithm, a lookup table called a Q-table is formed, which stores information of all of the State-Action values and maps them to a quality value, or Q-value, which is determined as the maximum expected future rewards for actions in a given state. All Q-values are initially set to 0 and as the agent explores the environment, the Q-values in the table will be continuously updated, with more iterations providing better approximations. In the complex environment where the number of states and action is high the Q function, approximated with the neural networks, is used to map state and action pairs to Q-values.[10]

3.3 Robot Skill Portfolio

To allow for the RL agent to perform general actions related to assembly, they can be defined as robot skills following the standard robot skill model as outlined by Bøgh et al.[11], visualized in Figure 2.

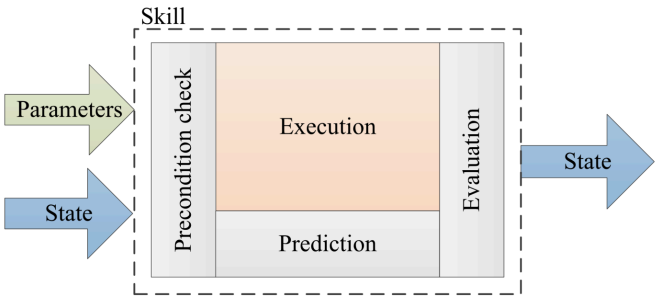

Fig. 2 Visualization of a general robot skill model. Parameters defining the skill along with the current state of an environment is used to evaluate whether an action is feasible through e.g., pre- and postconditions, which ensures that skills are successfully executed.[11]

The assembly of the proposed power inverter features four processes: pick, place, mechanical fastening, and soldering. Soldering is difficult to combine with the other processes within the same work cell, therefore, the assembly of the inverter will not include the final soldering of components to the PCB board. Thus, a portfolio of the skills necessary to perform the assembly can be summarized as in Table I.

Tab. I Robot skill portfolio.

Skill	Description
Pick	Pick X up
Place	Place X at Y
Load	Attach X to manipulator
Offload	Return X to holder
Fasten	Fasten component X and Y at hole Z

As defined by Figure 2, to verify the feasibility of a skill, the execution of each skill is governed by a set of pre- and postconditions while the execution itself is continuously evaluated through predictions. Pre- and postconditions are evaluated with a set of input parameters, i.e. the desired skill, and the environment state. If a skill fails to comply with any of these, it will not be considered feasible in that state.

Preconditions includes verifying compliance with the current environment for a desired action, e.g. the correct tool is attached or an object to be picked has not already been placed. After having asserted compliance with the environment, the skill can safely be executed. As the overall goal is to output an assembly sequence, the skills must carried out in a feasible sequence, i.e., no collisions are allowed throughout their execution. Assuming no collisions are present throughout execution, postcondition will then be applied to verify that the skill resulted in the expected outcome, i.e., after a tool change the desired tool is attached or after fastening a certain torque was achieved.

3.4 Simulation of Collisions

The execution of a skill in traditional sequence planning methods is ensured to be feasible through a hand-made precedence chart, however, to make a fully autonomous assembly sequence planner these potential collisions must be detected through simulations. Assembly itself is, however, inherently based on performing controlled collisions of components, and therefore, a robust system must be defined to distinguish between desired collisions, e.g. contact when placing a component, and undesired, actual, collisions.

Physical collision detection often involves analysis of forces that are being applied to the objects or the robot, which can be simulated in a physics-driven simulated environment, where assets are commonly defined as rigid bodies. The simulation environment should be

configured to accurately detect collision and rigid body dynamics, as it should replicate the real world behavior of the robot and the environment to a degree that is useful.

However, a highly realistic environment would lead to a long computation time, which is not desirable in a context of reinforcement learning. To mitigate the trade-off between the computational complexity and the accuracy of the simulation, the rigid body objects are represented as meshed objects. The computational complexity can therefore be reduced by simulating only one rigid body at a time.

This trade-off also largely affects the collision detection between the objects, as for example two flat objects can not be stacked on top of each other, as the contact points will move to different locations and make the objects shake or sometimes even explode away from each other due to surface penetration. A workaround to avoid surface penetration in the simulation, is by dropping the objects relatively close to the assembly point. [3]

4. Experimental Setup

The architecture of the system can be seen summarized in Figure 3. A DQN agent is trained to output an action, which is interpreted by the trajectory planner as a sequence of via points and end-effector manipulations based on which skill is indicated. Throughout this trajectory, the forces applied to the assembly components is tracked in real-time and fed into a collision detection algorithm. If a collision is detected, a flag is raised within the simulated environment which will affect the reward signal returned to the DQN agent and cause the environment to reset to an initial state. If no collisions are detected, a new state is computed and another iteration is carried out.

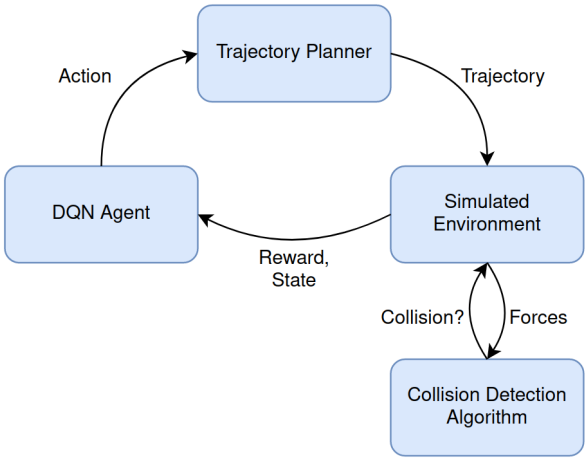

Fig. 3 Summary of system architecture.

The trajectory planner is running within the simulated environment such that defined pre- and postconditions for the skills can be evaluated. For a more robust system, these should be checked through sensors in the environment, as specifically postconditions will rely on external observations; However, this has not been a focus for this work. The project repository can be accessed via this hyperlink:https://github.com/P8-RL-Project/main-rl.

4.1 Physics Simulation Environment

The simulated environment is defined through Isaac Gym[12], where its physics engine is used to enable collision detection throughout the execution of a skill. Isaac Gym supports non-parallel, multi-environment training which can be used to increase training efficiency and convergence rates. To enable physics simulation of assets, a digital representation of a Panda manipulator is used while geometrically simplified assembly components have been developed. An overview of the simulated environment can be seen in Figure 4.

The observable state of the environment is presented to the agent as a sequence of 17 bits, where each bit is related to either pre- or postconditions of individual skills or the current state of the robot. This can be seen summarized in Figure 5.

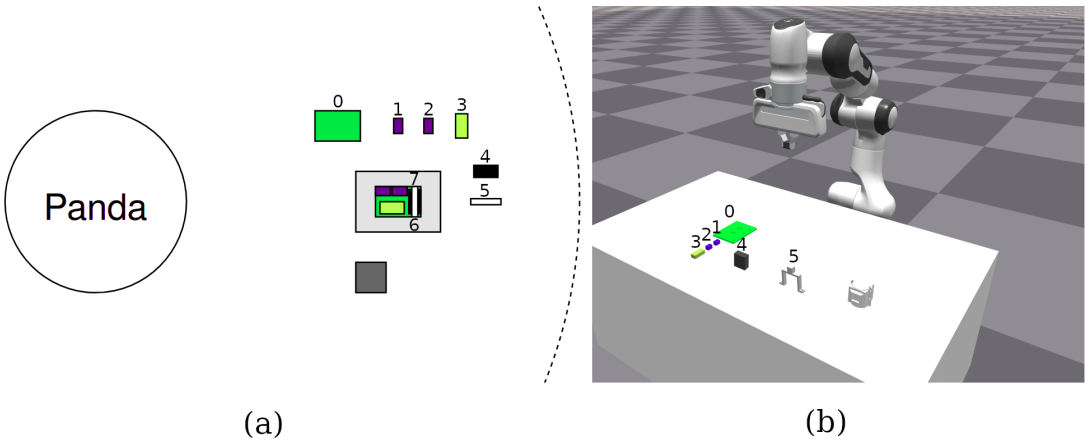

Fig. 4 A visualization of the component setup (a) along with its representation in the Isaac Gym simulation (b).

PCB (At start or goal)	1
	0
Switch 1 (At start or goal)	1
	0
Switch 2 (At start or goal)	1
	0
Clock (At start or goal)	1
	0
Transformer (At start or goal)	1
	0
Brace (At start or goal)	1
	0
Fasten 1	0
Fasten 2	0
Attached tool	0
Tool state	0
Prev. tool change	0

Fig. 5 Observable state space, here indicating the initial state configuration.

From Figure 5, it can be seen that a bit is added for all required skills for each component, while the robot state is defined by the current tool attached, 0 for gripper and 1 for fastener, the tool state, 0 for ready and 1 for engaged, along with a final bit to indicate whether the previous action is a tool change or not, which is included to avoid issues with the DQN agent exploiting repetitive tool changing.

4.2 Reinforcement Learning Agent

A DQN agent is defined through the SKRL library[13], which provides direct interfacing with Isaac Gym through environment through wrappers. The agent is configured with a discrete action space consisting of a total of 16 actions based on the skill portfolio, i.e., an action for each pick and place skill of the six components, an action for each fastening skill and an action for each tool change. These can be seen summarized in Figure 6. Given that the fastenings can be carried out in the end, the optimal sequence will never include more than one tool change and therefore the ability to change to the gripper becomes superfluous, however, there is nothing strictly limiting the agent from carrying out the fastenings early on and then returning to picking and placing.

As described above, each action is interpreted as a sequence of via points by a trajectory planner. These via points are fixed and determined by the positions of the components in the environment, and therefore, any new action will have the goal pose of the previous action as start pose. This allows for some opportunistic behaviour of the agent, where it can take advantage of being closer to e.g. the tool changing point at some poses.

The behavior of the agent is driven by a dense reward signal designed to incentivize three overall behaviours:

1) *Avoiding collisions*
2) *Completing the assembly sequence*
3) *Minimizing the sequence distance*

To encourage avoidance of collisions and completion of sequences, respectively a fixed penalty or a fixed reward is provided when encountered. Successful actions in between these two cases should be optimal, and therefore, their reward is scaled based on the 2D Cartesian distance from a given action's starting pose and goal pose. As the distance should be minimized, the actual distance is scaled by a negative factor.

Experiments with a reward signals based exclusively on this configuration showed that the agent would exploit

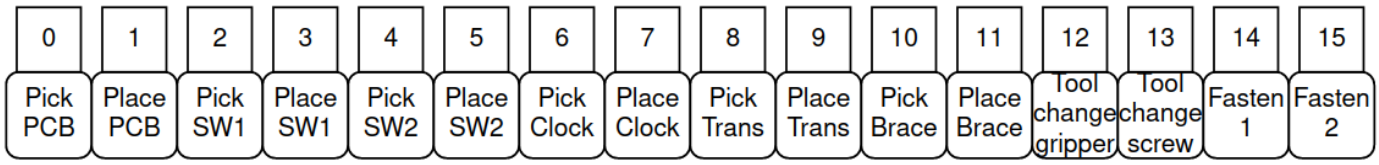

Fig. 6 The action space for the RL agent is defined as a discrete space with an action for each required process for the standardized assembly board. E.g., calling the action 1 will thus execute the trajectories necessary to place the PCB.

the distance reward of doing repetitive tool changing, as these happen at the same pose thus yielding an action distance of 0. To avoid this issues, a penalty is included for repetitive tool changing, and as mentioned in the definition of the observable state, a bit is included to allow the agent to observe whenever it had done a tool change. Furthermore, excessively long sequences would also occasionally be generated, and therefore, the sequence window of the agent is fixed to be a maximum of 20 skills, after which it is considered as out-of-time and a penalty applied. This scheme is summarized in Table II

Tab. II Reward signal scheme.

Collision	-10
Complete sequence	+10
Successful action	-0.1d_A
Out-of-time	-10
Repeat tool change	-2.5

Through this overall configuration, it is seen that the agents could generate both feasible and optimal solutions, however, the agents would encounter a "critical point", after having carried out all of the pick and place skills and would have to incorporate the tool change, where the agents occasionally would become stuck in a local minima.

4.3 Collision Detection

The Isaac Gym framework allowed collisions to be detected by defining a net contact force tensor for each asset which is composed by a force vector for each rigid body in that asset. Assets for the simplified inverter are defined as individual rigid bodies, and are thus given by a 3 dimensional vector defining the applied forces on the X-, Y-, or Z-axis, which is updated each time step. Collisions could thus be detected by applying a threshold to the force vector.

Through experiments within the Isaac Gym environment, where desirable assembly related collisions and undesirable collisions are compared, it is seen that the desirable collisions only had forces applied for single simulation time steps, while actual collisions is consistent over multiple simulation time steps. Therefore, to filter away assembly related collisions, a mean of the absolute net force over a time span of the last 6 frames is computed and thresholded.

An alternative option for collision detection is to measure the applied force of the gripper fingers; However, it is found that with this method it is more challenging to threshold collisions as each rigid body had unique mass properties thus requiring the gripper fingers to apply more force on certain rigid bodies.

The specific threshold is determined through trial and error, again through experiments with desirable and undesirable collisions, after which it is found that a threshold value of 1 is appropriate to consistently distinguish between the two.

4.4 Robot Trajectory Planner

To move the robot accurately in the simulation environment, a trajectory planner is developed. Initially, an inverse-kinematics controller is applied, however, it is found that this controller only accounted for the joint positions and thus disregarding dynamics, causing it to fail in accurately reaching goal poses.

Isaac Gym also supports Operation Space Control (OSC), which allows accurate control by incorporating dynamics. This controller utilizes various gains to achieve stability and takes 6 arguments as an input, namely the Cartesian position and Quaternion orientation of a goal position, the DoF position state of the robot, the mass matrix, Jacobian matrix for the end-effector forces, joint velocity of the robot and the velocity of the gripper. The methods and equations used for OSC are described by Peters et al. in [14]. This controller helped to overcome issues caused by the previously used inverse-kinematics controller and improved the overall trajectory planning which is resulting in more accurate robot manipulator movements.

5. Experiments and Results

The RL algorithm discussed in the previous section is trained over ten sessions for 20000 steps, after which the best performing model from each training session is evaluated to determine its optimality, while the training itself is evaluated to establish whether the agents are learning to act as intended. Therefore, two parameters are monitored throughout the training of each model: the ratio between feasible and infeasible sequences, to verify that the agent progressively learned to put feasible sequences together, and the total distance for feasible sequences throughout the training, to verify that the agent converged towards an optimal sequence.

The implementation used for testing performance of the agent is delimited to use a precedence chart for detection of collisions due to integration issues with the collision detection substitute at the time of testing.

5.1 Output Sequence Feasibility

The ratio between feasible and infeasible sequences are measured by writing to a log file each time an environment is either reset, marking those sequences that had collisions or failed pre- or postconditions, or had raised the done flag, marking those that succeeded. A 0 is appended to the log for each failed sequence, while a 1 is appended for each successful sequence. A summary of all tests are visualized in Figure 7.

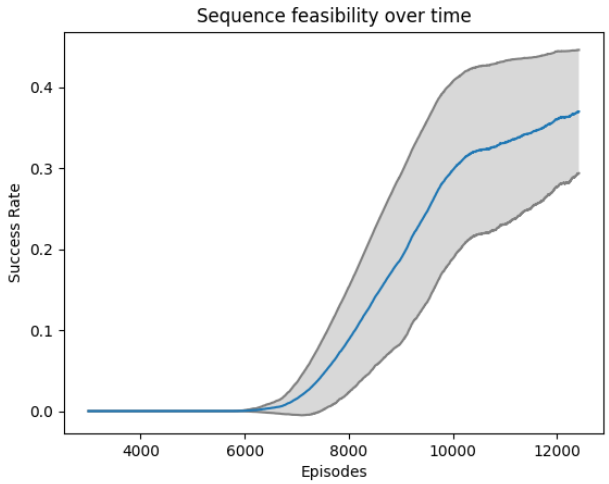

Fig. 7 A smoothed mean success rate per episode. Grey shadowed area indicates a standard deviation of the success rate.

The data in Figure 7 is smoothed through a moving average filter, based on the outcome of the prior 100 sequences, for easier inspection. From this test it is seen that the agents learn to generate complete sequences after approximately 6000 steps and tends to converge towards approximately 40% success rate, however, throughout the training, larger periods are seen where the agents seems to diverge from its current path to explore other sequences which results in successively failed attempts until it corrects it self. The ability to output feasible assembly sequences can potentially be increased by allowing the agents to train for longer periods, as one of the agent only achieves this in the very of its training session. However, once a certain length of assembly sequences is achieved, there is a very larger number of actions available to choose from when exploring, while only very few of them are actually feasible. Therefore, the better the agent becomes at producing assembly sequences, the more risky it is for it to extend it or explore new ways of putting actions together, and thus, most exploration results in a failed sequence. Adjustments may therefore be difficult to make without sacrificing a degree of exploration, which makes the agent more prone to becoming stuck in local minimas around the critical point.

5.2 Minimum Distance Convergence

Here the length of a sequence is again appended to a log file each time the done flag is raised. The results for each training sessions is visualized in Figure 8, again, smoothed with a moving average of the prior 100 distances for easier inspection.

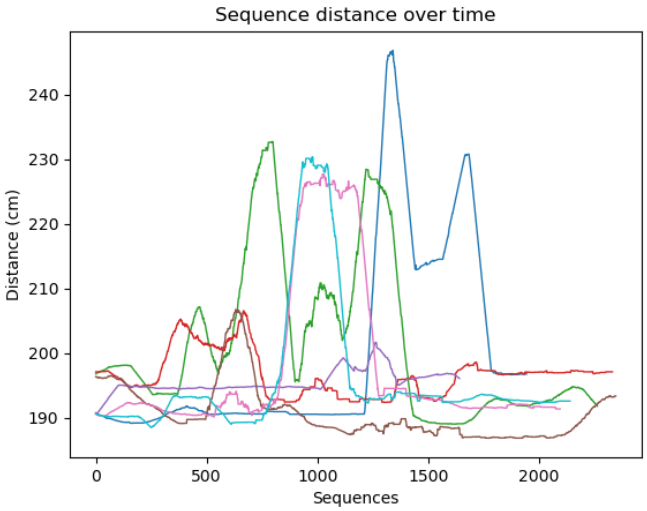

Fig. 8 Total distance for each sequence generated in each training session. Each training session is indicated by an individual color.

Figure 8 does not show any clear signs of converging towards the real optimal solution, however, it is seen

that the exploration of the agents occasionally causes them to diverge and starts outputting sequences with higher distance and after a while converge back towards sequences of lower distances.

For this test, distances are only recorded for successful, and thus complete, sequences. As described previously, the agents seems to struggle with getting past the critical point and this test seems to indicate that some optimization of the sequence of pick and place skills may occur before the agents eventually learn to apply the tool change and proceed. As the majority of the assembly sequence is related to picking and placing, this means that once the agent passes the critical point and learns to output a complete sequence, the majority of it may already be optimized to some degree, and therefore, strong convergence is not seen on complete sequences.

5.3 Output Sequence Optimality

The optimal assembly sequence is determined through an exhaustive search to be a total of 1.867m long and defined as PCB, Switch 1, Switch 2, Transformer, Clock, Brace, Fasten 1, Fasten 2. Correspondingly, this can be written in terms of skills, following the discretization from Figure 6, as [0,1,2,3,4,5,8,9,6,7,10,11,13,14,15].

After training the ten models for 20000 steps, the best output sequence and its distance, along with its deviation from the real optimal solution, is determined and can be seen summarized in Table III. N/A values are attributed to models which failed to produce a feasible sequence within the training session.

Tab. III Results from sequence optimality test.

Test	Output Sequence	Distance	Deviation
1	[0,1,4,5,2,3,8,9,10,11,6,7,13,14,15]	1.880	0.70%
2	N/A	N/A	N/A
3	[0,1,2,3,4,5,8,9,10,11,6,7,13,14,15]	1.867	0.00%
4	[0,1,4,5,2,3,8,9,10,11,6,7,13,14,15]	1.880	0.70%
5	[0,1,2,3,4,5,6,7,8,9,10,11,13,14,15]	1.885	0.96%
6	[0,1,2,3,4,5,8,9,10,11,6,7,13,14,15]	1.867	0.00%
7	[0,1,2,3,4,5,8,9,10,11,6,7,13,14,15]	1.867	0.00%
8	[0,1,6,7,2,3,8,9,4,5,10,11,13,14,15]	1.939	3.86%
9	N/A	N/A	N/A
10	[0,1,2,3,4,5,8,9,10,11,6,7,13,14,15]	1.867	0.00%

It should be mentioned that the agents that failed in generating a full sequence still managed to complete all of the pick and place tasks.

Though the previous tests shows that the agents are not converging towards the real optimal sequence, four agents succeeded in running through it at some point before diverging again, while the other agents also comes relatively close.

6. Conclusion

A reinforcement learning algorithm is designed to generate feasible assembly sequences based on a fixed discrete action space, while collision detection algorithms could be used to determine feasibility of specific actions, where actions are defined as specific robot skills. The actions are discretized such that an action defined a complex trajectory. Reward schemes are used to incentivize the generation of feasible and optimal assembly sequences based on distances for these actions. A critical point is found for the agents, which is learning to apply a tool change and continue the assembly afterwards, where either they would succeed and complete the sequence, or fail and become stuck in a local minima. Experiments indicated that agents showed signs of learning as intended. Success rates of output sequences increase over time and sequences converge near the real optimal assembly sequence.

7. Future Work

For future work it would be interesting to scale the assembly products by modifying the state space and including more options and test to see if the agent would be able to find assembly sequences for a wider selection of products. Products that require multiple tool changes would be of particular interest, to evaluate the influence of the critical point.

Furthermore, other algorithms than DQN, e.g., more modern ones such as Proximal Policy Optimization (PPO), is also suggested to see if they can improve learning capabilities.

Finally, sensors capable of evaluating pre- and postconditions should also be developed, to provide a higher reliability of the system and make it more rigid.

Acknowledgement

The authors of this work gratefully acknowledge Grundfos for sponsoring the 10th MechMan symposium.

References

[1] Z. Xue, J. Liu, C. Wu, and Y. Tong, "Review of in-space assembly technologies," Chinese Journal of Aeronautics, vol. 34, no. 11, pp. 21–47, 2021.

[2] Use of Extraterrestrial Resources for Human Space Missions to Moon or Mars. Springer International Publishing, 2nd ed., 2018.

[3] J. De Winter, I. Makrini, G. Van de Perre, A. Nowé, T. Verstraten, and B. Vanderborght, "Autonomous assembly planning of demonstrated skills with reinforcement learning in simulation," Autonomous Robots, vol. 45, no. 8, pp. 1097–1110, 2021.
[4] M. Rashid, W. Hutabarat, and A. Tiwari, "A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches," The International Journal of Advanced Manufacturing Technology, vol. 59, pp. 335—349, 2012.
[5] Z. Saribatur, V. Patoglu, and E. Erdem, "Finding optimal feasible global plans for multiple teams of heterogeneous robots using hybrid reasoning: an application to cognitive factories," Autonomous Robots, vol. 43, pp. 213—238, 2019.
[6] T. De Fazio and D. Whitney, "Simplified generation of all mechanical assembly sequences," IEEE Journal on Robotics and Automation, vol. 3, no. 6, pp. 640–658, 1987.
[7] B. Sunil, V and S. Pande, S, "Webrobot: Internet based robotic assembly planning system," Computers in Industry, vol. 54, no. 2, pp. 191–207, 2004.
[8] L. Xu, C. Wang, Z. Bi, and J. Yu, "Autoassem: An automated assembly planning system for complex products," IEEE Transactions on Industrial Informatics, vol. 8, no. 3, pp. 669–678, 2012.
[9] I. Rodriguez, K. Nottensteiner, D. Leidner, M. Kaßecker, F. Stulp, and A. Albu-Schäffer, "Iteratively refined feasibility checks in robotic assembly sequence planning," IEEE Transactions on Industrial Informatics, vol. 8, no. 3, pp. 669–678, 2012.
[10] R. Sutton and G. Barton, Reinforcement learning: an introduction. The MIT Press, 2nd ed., 2018.
[11] S. Bøgh, O. Nielsen, M. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?," 2012.
[12] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," arXiv:2108.10470v1 [cs.RO] 24 Aug 2021, pp. 4–31, 2021.
[13] A. Serrano-Muñoz, N. Arana-Arexolaleiba, D. Chrysostomou, and S. Bøgh, "skrl: Modular and flexible library for reinforcement learning," 2022.
[14] J. Peters and S. Schaal, "Learning operational space control," pp. 1–8, 2006.