Automation in Construction 133 (2022) 104006

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Robotic architectural assembly with tactile skills: Simulation and optimization

Boris Belousov [a,*], Bastian Wibranek [b], Jan Schneider [a], Tim Schneider [a], Georgia Chalvatzaki [a], Jan Peters [a], Oliver Tessmann [c]

[a] Intelligent Autonomous Systems, TU Darmstadt, Germany
[b] School of Architecture and Planning, University of Texas at San Antonio, United States
[c] Digital Design Unit, Department of Architecture, TU Darmstadt, Germany

ARTICLE INFO

Keywords:
Discrete design
Robots in architecture
Reinforcement learning

ABSTRACT

Construction is an industry that could benefit significantly from automation, yet still relies heavily on manual human labor. Thus, we investigate how a robotic arm can be used to assemble a structure from predefined discrete building blocks autonomously. Since assembling structures is a challenging task that involves complex contact dynamics, we propose to use a combination of reinforcement learning and planning for this task. In this work, we take a first step towards autonomous construction by training a controller to place a single building block in simulation. Our evaluations show that trial-and-error algorithms that have minimal prior knowledge about the problem to be solved, so called model-free deep reinforcement learning algorithms, can be successfully employed. We conclude that the achieved results, albeit imperfect, serve as a proof of concept and indicate the directions for further research to enable more complex assemblies involving multiple building elements.

1. Introduction

The global challenges of architecture and the construction industry, such as CO_2 emissions, waste production, and low productivity, are currently being approached by computational design, digital fabrication and robotics. Digitally prefabricated modular construction systems promise to improve the way we build in terms of productivity, precision, and future reuse. Computational design tools and digital fabrication allow for parametric mass customization of modules or complex combinatorics of discrete elements. However, the robotic assembly and future re-assembly of those elements on site, under messy and unpredictable circumstances is still an unsolved problem.

The digital continuity from design to production is interrupted when it comes to assembly, (dis-)assembly and (re-)assembly. However, automation is needed to make such a circular economy approach economically feasible. One possible way to achieve this ambitious goal is explored in this research project: The use of learning robots that do not need to be preprogrammed but act autonomously under closed-loop sensory feedback. We intend to develop novel robotic assembly processes and learning-based algorithms. We subdivided the assembly procedure into three steps: grasping a part at the pickup location, transporting it to the vicinity of the target location, and finally placing the part. The research then focused on trajectory planning for transportation and reinforcement learning (RL) for the part placement within the assembly process.

Recent developments in deep reinforcement learning have demonstrated the feasibility of extracting patterns from high-dimensional signals produced by visual and other sensors [32] thereby improving both the perception and actuation pipelines and enabling real robot control through reinforcement learning [5,19]. Therefore, several tasks that were previously not feasible are now within reach.

In this paper, we are concerned with the problem of autonomously connecting building modules (Fig. 2). It is a challenging contact-rich manipulation task that requires coordination between perception and action, between observing the visual and tactile sensor readings and sending action commands to the robot joints and the gripper. The aspired functionalities are aimed to enable component re-use, disassembly and re-assembly of parts in the sense of a productive and

* Corresponding author.
E-mail addresses: boris.belousov@tu-darmstadt.de (B. Belousov), bastian.wibranek@utsa.edu (B. Wibranek), jan.schneider.43@stud.tu-darmstadt.de (J. Schneider), tim.schneider94@t-online.de (T. Schneider), georgia.chalvatzaki@tu-darmstadt.de (G. Chalvatzaki), jan.peters@tu-darmstadt.de (J. Peters), tessmann@dg.tu-darmstadt.de (O. Tessmann).

https://doi.org/10.1016/j.autcon.2021.104006
Received 6 May 2021; Received in revised form 26 September 2021; Accepted 6 October 2021
Available online 21 October 2021

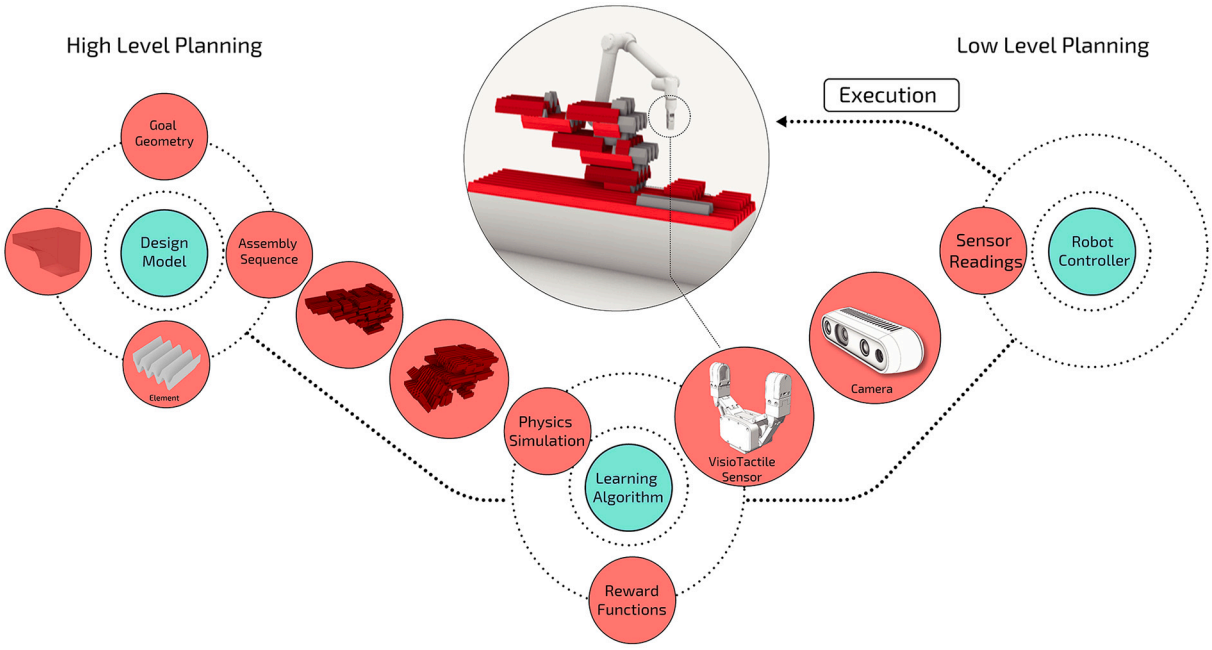

Fig. 1. Learning-based control architecture for autonomous assembly.

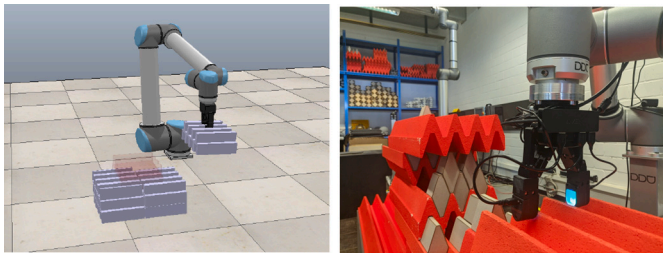

Fig. 2. Simulated and real robot setup for learning to stack building modules. The robot is equipped with a gripper and tactile sensors to enable fine contact-rich manipulation.

effective circular economy in construction. The code is available at https://github.com/b4be1/rl-arch-assembly.

2. Related work

Re-assembly through in-built manipulators that change space and construction during or after its use is subject to research and design speculation [12]. Early studies date back to the 1970's. The Architecture Machine Group at MIT explored the impact of constant re-assembly of a voxelated landscape populated with gerbils. While the animals constantly changed the voxel configuration, the robotic system (SEEK) sought to keep track of the "continually changing mismatch between three-dimensional reality and the model residing in the core memory of the computer" [29].

More recently, the design team R&Sie(n) put forth the project Olzweg to speculate on the design of constant re-assembly. Here a robot that constantly repositions elements becomes an integral part of the architecture. Space is in constant flux [27]. In a similar vein, Fabio Gramazio and Matthias Kohler presented the Endless Wall at the "Scientifica 2011", where a mobile robot unit builds and rebuilds a curved brick wall and constantly controls the results by means of 3D scan technology and is furthermore able to react in real-time on any deviations and adapt during the assembling process. These projects aimed at exploring the use of robotic-based fabrication methods on construction sites, considering aspects such as complexity of construction sites and sensing capabilities

Fig. 3. Luminale installation 2018. A centrally placed UR 10 robot constantly repositions modules. This reassembly in combination with changing light color alters form and meaning of the installation. (Image credit: Stefan Winkler). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of robots in regard to its own position, the surroundings and building elements [12].

In 2018, the authors designed an installation consisting of a modular structure, an industrial robot arm and a light system as a contribution to the light festival Luminale in Frankfurt, Germany. The installation is shown in Fig. 3. The corrugated modules made from EPS foam were constantly re-assembled by a UR 10 robot to change the form and meaning of the installation. The robot became an integral part of the installation and the modules were designed for robotic (re-)assembly. Their zig-zag shape provided self-calibrating capacities for stacking, while the longitudinal profile accepted small imperfections during the placement of the elements. The installation carried an anamorphic image that visitors could decipher only after finding the correct point of view. Once they identified the message, the robot would deconstruct it by re-positioning the modules. The project explored the constant change of space due to changing light conditions and permanent (re-) assembly.

Further academic research to connect the material system with

robotic systems was conducted in the projects Material-Robot System (MRS) [15] and Robot-oriented design (ROD) [3]. Here, the design of elements follows specific considerations of the robot's strengths and capabilities, including connections between elements and error-correction through self-alignment. In order to foster an understanding of the challenges in the robotic assembly process, designers simulate the robotic construction inside the digital design environment. Hence, assembly workflows become an integral part of a continuous design process.

The computational problems arising in robotic fabrication and assembly are challenging and therefore new methods and solutions at the interface between digital design and computer science are being explored. A methodology for using neural networks as a design tool in architecture was proposed in [28], where convolutional autoencoders were employed to predict the geometry of curved metal surfaces from a tracking pattern applied on the English wheel and vice versa. This allows the designer to make use of the trained models in both directions: as a way to design a surface and observe the resulting imprinted pattern, which is the traditional design-to-fabrication sequence, but also in a more crafty way of designing the desired fabrication marks and observing their effect on the geometry. A conceptually similar approach based on supervised learning has been applied to robotic carving [4], where human demonstrations were used as the training data to teach the robot to perform skillful timber manufacturing.

Although supervised learning is quite powerful and can be integrated in many ways into the creative design process as a regression subroutine, some types of problems require a qualitatively different approach. Namely, problems involving multi-step planning are more naturally framed in the language of reinforcement learning [31]. Such are the problems of autonomous assembly where a number of building elements need to be put together in a desired sequence or arrangement. An overview of prior research on robot-based autonomous construction is provided in [23]. In the present paper, we are framing the problem of part placement as a reinforcement learning problem and thereby enabling the use of these novel computational tools in the digital design and fabrication context.

The problem of building a structure from a set of given blocks by an autonomous robot was considered in [13], where the challenge was approached by applying task-and-motion planning to calculate a path in the configuration space of the robots. However, the authors did not consider the low-level control problem and instead assumed that the robot is able to follow the plan perfectly without any deviations and that the parts can be grasped without any slippage. Optimization of the assembly sequence for curve-shaped designs based on reinforcement learning was proposed in [36], but no corresponding robot controller was developed. In [10], a hierarchical control framework was proposed that learns to sequence the building blocks to construct arbitrary 3D designs and ensures their feasibility with the robot-in-the-loop execution. A multiagent approach to the assembly problem was developed in [30], where a swarm of autonomous robots was trained with reinforcement learning to achieve a common goal.

Utilizing tactile information for robotic manipulation tasks is an area of active research [16,37]. Since the information from tactile sensors is typically high-dimensional and hard to interpret, learning is commonly employed to develop controllers that can use such rich feedback signals. In [5], a neural network model was trained to predict the stability of a planned grasp sensed with the GelSight tactile sensor [38]. In [33], reinforcement learning was used to stabilize an object with a robot arm based on tactile feedback from a BioTac tactile sensor [8]. In [32], reinforcement learning was used to move and rotate small objects such as marbles and dice using robotic fingers equipped with a modified GelSight sensor.

In this paper, we employ reinforcement learning to achieve part placement within a robotic architectural assembly. The advantage of the learning approach consists in the system's ability to adapt to changes in the environment, such as part positions or desired configurations.

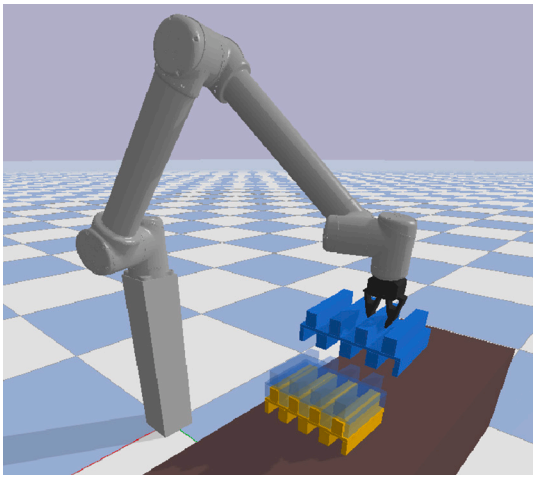

Fig. 4. Robot is learning to connect modules in our simulated environment (PyBullet physics engine). The robot controller is trained on 10,000 randomly generated structures.

Contact between parts, friction, and deviations between planned and actual part locations require on-the-fly adaptation, which is a core strength of learning-based solutions. Furthermore, instead of specifying a path for the end-effector or providing similar low-level commands to the robot, the designer may specify a high-level objective for the robot and let the learning system figure out the optimal control sequence for the task. Finally, the use of rich high-dimensional sensors, such as depth cameras, Lidar, vision-based tactile sensors, etc. calls for scalable methods that can work with such inputs. Deep reinforcement learning is a promising approach to enable the use of such rich feedback signals for control and therefore we explore its potential in architectural assembly.

3. Reinforcement learning for contact-rich modular assembly

A crucial challenge in autonomous robotic assembly is to deal with the contact between building parts. As long as the robot does not need to bring parts in contact with each other during manipulation, classical methods such as path planning and open-loop kinematic control can be applied. For example, brick laying robots can be made sufficiently precise without invoking sophisticated sensing modalities [26]. However, if parts have irregular shape or if they need to be assembled in a non-trivial manner, then more advanced robot control strategies are required. Currently, this problem is not completely solved and is a part of research in robotics on contact-rich manipulation [19,32,37].

Our goal is to accomplish autonomous modular assembly using a robot arm equipped with a gripper (Fig. 4). As the first step towards this goal, we developed a simulation environment and let the robot learn how to place parts in that simulated environment. Initially, we explored the strategies that do not require tactile feedback and only react based on sensing the part positions and orientation. We found that a learned controller is capable of placing parts when the total tolerance is allowed to be greater or equal to 8 mm. For precise manipulation, this is still too big.

Therefore, we equipped our gripper with a vision-based tactile sensor DIGIT [18] to enable more fine-grained manipulation, as exemplified in [32]. To train a reinforcement learning agent, we developed a simulation of the tactile sensor based on the model of light propagation [35]. However, this model does not include soft-body simulation and therefore we found it not suitable for contact-rich manipulation tasks that involve deformations of the gel on the sensor, as is the case in our setting. Therefore, we finish our presentation with a real-robot learning setup, where the learning is performed on the real robot, without employing a simulation environment. We give an outlook on how this method can be applied to architectural assembly tasks.

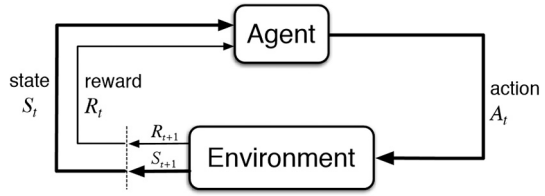

Fig. 5. Reinforcement learning loop. Agent performs action A_t, Environment responds with state S_t and reward R_t.

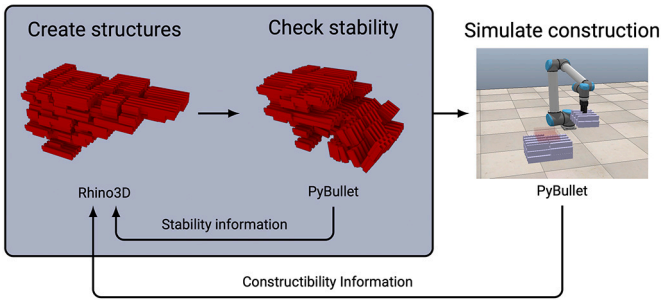

Fig. 6. Integration of Rhino3D with PyBullet physics engine for evaluating the stability of assembled structures within the reinforcement learning loop.

3.1. Reinforcement learning problem setting

We formulate the problem of part stacking depicted in Fig. 4 as a reinforcement learning problem. We aim do develop a controller that is robust with respect to deviations in positions of the parts and can react to contact when parts are being joined together.

Reinforcement learning [31] is a general framework for developing controllers for environments which are hard to model analytically. In our scenario, it is very hard to come up with an analytical model for a controller that would react to various contact interactions between parts. Therefore, reinforcement learning is a more promising approach. Our problem can be seen as a more difficult version of a classic in the control literature peg-in-a-hole task [14]. A similar setting in the context of architecture was recently considered within the robotic assembly of timber joints [1]. The difference in our case is that the parts have multiple slots that need to be aligned, which makes the problem more difficult and calls for the use of the tactile sensing technology.

To describe our problem in the language of reinforcement learning, we need to specify states, actions, and the reward function. A generic RL loop is shown in Fig. 5. The agent is a program that sends commands to the actuators of the robot (Cartesian joint velocity of the end-effector and gripper closure commands in our case) and receives back the state information (part positions and orientations, robot state, tactile sensor readings, etc.) together with a reward signal that evaluates how well the task is solved (e.g., position error between the desired and current part poses). The agent keeps interacting with the environment and adapting itself till it finds a successful policy $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, a mapping from states to distributions over actions, that yields the highest expected return.

3.2. Rhino 6 integration

We aim to develop a general learning-based framework for autonomous assembly of architectural modular structures, as shown in Fig. 1. The goal of the framework is to allow an architect to specify a high-level design objective in Rhino/Grasshopper [22], which is then achieved by an intelligent agent. The agent first turns the design objective into a modular assembly plan, and then it turns the assembly plan into a robot controller that is able to build the structure.

The problem of finding a sequence of assembly steps that satisfies a given objective can also be framed as a reinforcement learning problem [2]. Here we assume that an assembly plan is already given and our task is to design a controller that can execute that assembly plan in a robust manner. However, although assembly in 2D has been successfully tackled in [2], extension to 3D and more complex blocks is non-trivial and is a part of our research programme.

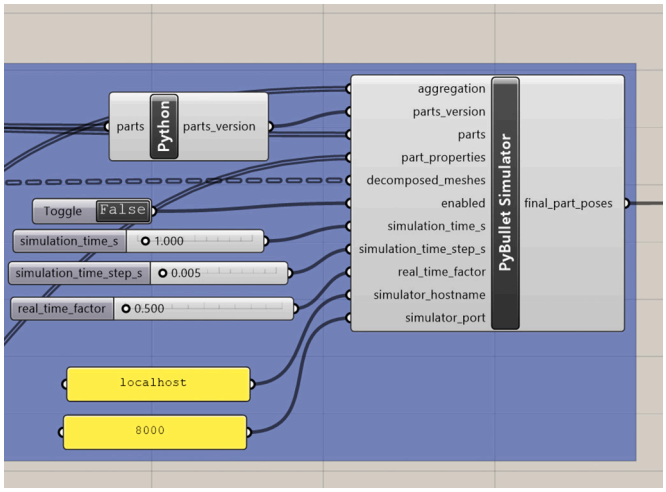

Fig. 7. PyBullet interface as seen in Grasshopper. Data about the scene and parts can be exported from Rhino and imported in PyBullet. The result of the simulation in PyBullet subsequently becomes available in Grasshopper.

In order to accommodate for reinforcement learning in the context of architecture, we developed several bridges between Grasshopper and PyBullet [6]. Fig. 6 demonstrates the high-level functionality provided by the bridges. Information can flow in both direction, during the design phase (part geometry and other properties are exported from Grasshopper and imported in PyBullet; results of physical simulation are communicated back from PyBullet to Grasshopper) as well as during the construction phase (current state of construction communicated from PyBullet to Grasshopper, allowing on-the-fly changes to the design). Fig. 7 shows how the developed PyBullet interface appears in Grasshopper.

We use PyBullet because it is a fast physics engine widely used in robotics. It not only enables learning of the block stacking controller but can also be used to check the physical properties of a structure designed in Rhino, such as stability, structural performance, and tectonics during and after assembly. The communication is implemented using TCP/IP sockets.

The integration between Rhino and PyBullet enables physical simulation of a modular structure in PyBullet and subsequent visualization of the resulting poses of all parts in Rhino. The results of this simulation can be used to compute a cost function and pass it to an optimization algorithm, such as an evolutionary algorithm or reinforcement learning. Thus, the entire simulation ecosystem of grasshopper can be used to derive cost measures for reinforcement learning from the wide range of architectural requirements such as structure, material behavior, daylight, temperature, context, etc.

3.3. Learning algorithm description

There are multiple algorithms available in the RL toolbox [25] for solving the resulting optimization problem. Depending on the problem properties—such as discrete or continuous state and action space, sparse or dense reward function, cheap or expensive simulation, etc.—one can choose a different algorithm. We found Twin Delayed DDPG (TD3) [9] to be most suitable for our task. TD3 is a more stable version of the well-known Deep Deterministic Policy Gradients (DDPG) algorithm [20], that extends Q-learning to continuous action spaces. DDPG for the

first time showed that a Q-learning-like algorithm can be applied to solve continuous control problems, such as robot control in simulated environments.

TD3 is a model-free off-policy RL algorithm. Model-free means that the model of the environment is not known to the agent, i.e., the agent does not know a priori how its actions affect the observed states and rewards. Such ignorance may be surprisingly useful, as demonstrated by the great success of model-free RL in tasks ranging from robotic in-hand manipulation [24] to playing computer games such as StarCraft II [34]. Off-policy means that the algorithm can utilize experiences collected by another policy. In particular, past experiences can be kept in memory and reused. This allows off-policy algorithms to be more sample-efficient, requiring fewer interactions with the environment to solve the task [7].

TD3 learns two Q-functions Q_{θ_1} and Q_{θ_2} and a policy π_ψ. Both Q-functions and the policy are parameterized by neural networks with parameters $\{\theta_1, \theta_2, \psi\}$, respectively. To make learning more stable, TD3 uses separate target Q-functions $Q_{\theta_1'}$ and $Q_{\theta_2'}$ and a target policy $\pi_{\psi'}$ to update the current Q-functions Q_{θ_1} and Q_{θ_2}. The parameters of the target networks are updated to slowly track the parameters of the current networks:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i',$$
$$\psi' \leftarrow \tau\psi + (1-\tau)\psi',$$

with $\tau \in (0, 1)$ being a hyperparameter. Furthermore, TD3 maintains a replay buffer in which it stores previously seen transitions as tuples (s, a, r, s'). At each update step, TD3 samples a random minibatch of N transitions (s_j, a_j, r_j, s_j') from the replay buffer, and the Q-functions Q_{θ_1} and Q_{θ_2} are adjusted to minimize the Bellman error on this minibatch. To counter the overestimation bias present in DDPG, both target Q-functions $Q_{\theta_1'}$ and $Q_{\theta_2'}$ are evaluated and the smaller of the two values is used as the target. This results in the loss function

$$L(\theta_i) = \frac{1}{N}\sum_j \left(y_j - Q_{\theta_i}(s_j, a_j)\right)^2,$$

$$y_j = r_j + \gamma \min_{i=1,b} Q_{\theta_i'}\left(s_j', \tilde{a}_j\right),$$

where $\tilde{a}_j$ denotes the action chosen by the target policy $\pi_{\psi'}$ supplemented by Gaussian noise,

$$\tilde{a}_j = \pi_{\psi'}(s_j') + \varepsilon,$$
$$\varepsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma), -\varepsilon_{max}, \varepsilon_{max}).$$

The noise is added to robustify the Q-function estimate by enforcing it to be smoother and return similar values for similar actions. The policy π_ψ is trained to maximize the agent's performance given by the expected discounted return

$$J(\psi) = \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t r(s_t, \pi_\psi(s_t))\right]$$

This optimization is performed using stochastic gradient descent (SGD), plugging in the estimates of the Q-functions described above into the policy gradient expression

$$\nabla_\psi J(\psi) \approx \frac{1}{N}\sum_j \nabla_a Q_{\theta_1}(s, a)|_{s=s_j, a=\pi_\psi(s_j)} \nabla_\psi \pi_\psi(s)|_{s=s_j}.$$

For more details on the theoretical underpinnings of TD3 and its implementation, see papers [9,25].

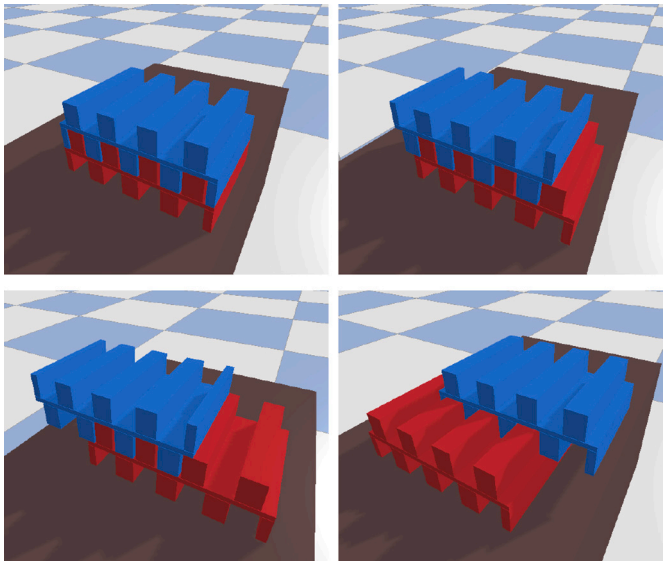

Fig. 8. The agent is trained on thousands of random goal configurations where the blue module is connected to the red module. By learning to solve all of these configurations, the agent develops a robust strategy. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.4. Reinforcement learning for part stacking

In this section, we apply the TD3 algorithm described in Section 3.3 to solve the stacking problem described in Section 3.1. We describe the experimental setting and present the results.

Training proceeds in episodes. At the beginning of each episode, the starting block is placed at random in a square with 60 cm side length. The robot then has to place a second block on top of the starting block in a given stable configuration. Examples of these stable configurations are shown in Fig. 8. While placing the second block, the robot has to ensure that the first block is not displaced.

The spawning location of the second block is sampled uniformly in a cube of side length 2 cm that is located 30 cm above the center of the starting block's spawning area. The spawning orientation is obtained by slightly perturbing the goal orientation. This perturbation is achieved by sampling Euler angles between $-5°$ and $5°$ and rotating the block accordingly.

Each episode, the robot starts with the block in its gripper. To avoid overfitting with respect to the exact grasp location, the position of the grasp on the block is also randomized along the ridges of the block and along the block's up-axis. Fig. 4 displays a possible starting configuration of the training environment. All these randomizations are included to make the controller more robust with respect to deviations in the positioning of the block and gripper that might occur on the real robot due to inaccuracies in the localization of the blocks and controller deviations.

An episode terminates if the gripper moves more than 10 cm away from the current block or if the time limit of 3 s is reached. With the controller frequency of 20 Hz, this yields a maximum sequence length of 60 steps.

3.4.1. Optimization objective

The objective of the learning agent is to learn a policy $\pi_\theta(\mathrm{a}|\mathrm{o})$ that minimizes the expected cost of an episode $\tau = (\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{a}_{T-1}, \mathbf{s}_T)$:

$$\min_\theta \ \mathbb{E}_{\tau\sim p(\tau|\pi_\theta)}\left[\varphi_f(\mathrm{s}_T) + \frac{1}{T}\sum_{t=1}^{T}\varphi_i(\mathrm{s}_t, \mathrm{a}_t)\right]$$

where $\mathbf{s}_t$ is the full system state at time step t, $\mathbf{o}_t$ is the observation the

policy gets of $\mathbf{s}_t$, $\mathbf{a}_t$ is the chosen action, $\varphi_f(\mathrm{s}_T)$ is the final cost, $\varphi_i(\mathrm{s}_t, \mathrm{a}_t)$ is the intermediate cost at step t, and

$$p(\tau|\pi_\theta) = p(\mathbf{s}_1)\prod_{t=1}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)p(\mathbf{o}_t|\mathbf{s}_t)$$

is the trajectory distribution induced by policy π_θ.

The action vector $\mathbf{a} = (\mathbf{a}_{\text{arm}}, a_{\text{gripper}})$ is composed of two components: the target joint velocities $\mathrm{a}_{\text{arm}} \in \mathbb{R}^6$ of the 6 arm joints and the target closure state $a_{\text{gripper}} \in [0, 1]$ of the gripper. Concerning the gripper control signal, a 1 indicates that the gripper shall be fully closed, while a 0 indicates that the gripper shall be fully opened. The signal is realized by a position controller on the gripper motor.

Our main objective is that both blocks of the structure are placed in their respective target pose. Hence, we chose the final cost function as follows:

$$\varphi_f(\mathrm{s}_T) = \alpha_{\text{pose}}\varphi_{\text{pose}}(\mathrm{s}_T) \tag{1}$$

where $\alpha_{\text{pose}} \in \mathbb{R}$ is a weighting factor. φ_{pose} is a penalty for the pose error of the blocks. To ensure that the robot does not displace the block that is already placed at the beginning of the episode, this penalty includes the pose errors of both blocks. Note that "pose" here refers to a combination of position and orientation. It is challenging to define a distance metric on poses directly since it is not straightforward to weight position and orientation error up in a useful manner. Thus, we chose a different approach to measuring pose error. To measure the error of the pose, we compute the average positional error of the eight vertices of the bounding box of each block:

$$d_k^2 = \frac{1}{8}\sum_{j=1}^{8} \left\|\mathrm{m}_{k,j} - \widetilde{\mathrm{m}}_{k,j}\right\|_2^2$$

where $\mathbf{m}_{k,j}$ is the target position of vertex j of block k at the final time step and $\widetilde{\mathrm{m}}_{k,j}$ is the actual position, respectively.

While it would be possible to use the mean of the distances d_k^2 directly as a cost function, this approach comes at a major drawback: the closer a block is to its target pose, the flatter the squared penalty becomes and the less incentive the learner has to place the block even more precisely. In construction tasks, even tiny deviations from the correct position can have a huge impact on whether a structure can be assembled properly. We therefore use a modified cost function

$$\delta_{log}\left(d^2\right) = d^2 + 0.01\left(ln\left(d^2 + \varepsilon\right) - ln(\varepsilon)\right)$$

with a small value of $\varepsilon = 10^{-5}$. Due to the function's concave shape close to 0, even small positioning errors are punished significantly.

The final pose cost is then computed as

$$\varphi_{\text{pose}}(\mathrm{s}_T) = min\left\{\frac{\frac{1}{B}\sum_{k=1}^{B}\delta_{log}\left(d_k^2\right)}{\varphi_{\text{pose}}^{max}}, 1\right\}$$

where $B = 2$ is the number of blocks in the scene and $\varphi_{\text{pose}}^{max}$ is a hyperparameter that controls the clipping of this term. The intuition behind the clipping is that if the block is placed too far away from the target location, we consider the task as failed and simply assign a maximum cost. Hence, the reward becomes more sparse and the Q-function in regions far away from the target state (e.g., if the robot threw the block away from itself) becomes easier to learn. The inverse scaling by $\varphi_{\text{pose}}^{max}$ simply ensures that $\varphi_{\text{pose}}(\mathrm{s}_T) \in [0, 1]$, which makes it easier to choose the weights α later since all cost terms will share the same domain.

The intermediate cost $\varphi_i(\mathrm{s}_t, \mathrm{a}_t)$ consists of a constant term, a penalty for movements of the current block while it is not grasped, and a penalty on the acceleration of the end-effector:

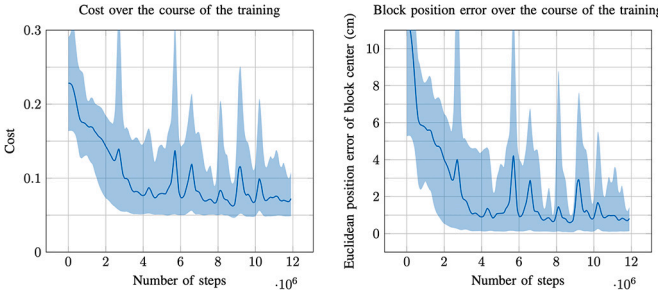

Fig. 9. Average cost (left) and block position error (right) over the course of TD3 training in simulation. The x-axis shows the number of agent's interactions with the simulated environment running at 20 Hz. The plots show the mean and 5–95% percentiles obtained by evaluating the model on 1000 unseen test configurations every 100,000 steps of training.

$$\varphi_i(\mathrm{s}_t, \mathrm{a}_t) = \alpha_{\text{time}} + \alpha_{\text{rel}}\varphi_{\text{rel}}(\mathrm{s}_t) + \alpha_{\text{acc}}\varphi_{\text{acc}}(\mathrm{s}_t, \mathrm{a}_t) \tag{2}$$

The first term encourages the learner to complete the episode quickly. φ_{rel} is a penalty for the movement of released blocks, i.e., blocks that are not grasped, to discourage the robot from throwing the blocks at the target.

As long as the block is in the gripper, the agent should be able to move it around freely, without additional cost. The cost φ_{rel}, thus, is activated only if the block is released from the gripper. Whether the block is grasped is determined by checking if the distance from the block to the gripper's fingers d_{fingers} is larger than 1 cm.

$$\varphi_{\text{rel}}(\mathrm{s}_t) = \begin{cases} \varphi_{\text{vel}}(\mathrm{s}_t) & \text{if } d_{\text{fingers}} > 0.01m \\ 0 & \text{otherwise} \end{cases}$$

The cost for the movement of a released block is realized as a quadratic penalty on the block's linear and angular velocity.

$$\varphi_{\text{vel}}(\mathrm{s}_t) = min\left\{\frac{\left\|\mathrm{v}_{\text{lin}}\right\|_2^2 + \left\|\mathrm{v}_{\text{ang}}\right\|_2^2}{\varphi_{\text{vel}}^{max}}, 1\right\}$$

Here $\mathbf{v}_{\text{lin}}$ is the linear velocity of the block's center, $\mathbf{v}_{\text{ang}}$ the angular velocity, respectively, and $\varphi_{\text{vel}}^{max}$ is again a hyperparameter.

The penalty on the accelerations of the end-effector is realized as a quadratic cost on the linear and angular velocity differences between the current and the last time step.

$$\varphi_{\text{acc}}(\mathrm{s}_t, \mathrm{a}_t) = \frac{1}{\varphi_{\text{acc}}^{max}}\left(\left\|\left(\mathrm{v}_{\text{lin}}^{\text{ee}}\right)_t - \left(\mathrm{v}_{\text{lin}}^{\text{ee}}\right)_{t-1}\right\|_2^2 + \left\|\left(\mathrm{v}_{\text{ang}}^{\text{ee}}\right)_t - \left(\mathrm{v}_{\text{ang}}^{\text{ee}}\right)_{t-1}\right\|_2^2\right) \tag{3}$$

where $\left(\mathrm{v}_{\text{lin}}^{\text{ee}}\right)_t$ and $\left(\mathrm{v}_{\text{ang}}^{\text{ee}}\right)_t$ are the linear and angular velocities of the end-effector at time t, respectively. The purpose of this term is to encourage smooth movements in order to reduce energy consumption and decrease the strain on the robot's motors and gears.

3.4.2. Results

The controller was trained on 10,000 randomly generated structures in simulation. Every 100,000 steps, the controller is evaluated on 1000 unseen test structures. The learning rate starts at $10^{-4.5}$ and is decayed over time, multiplied by $1/\sqrt{10}$ after 40%, 60%, 80%, 90%, and 95% of the total training time of 48 h. Fig. 9 shows the evolution of the average cost (negative return) and the position error in block placement over time. The controller reaches a block center position error of around 8 mm after 12 million steps or 48 h of training. The same accuracy is actually achieved already after 4 million steps (16 h), therefore the training could be stopped earlier without a significant drop in task performance. While the accuracy of 8 mm may be satisfactory for certain types of assemblies when part tolerances are sufficiently large, we

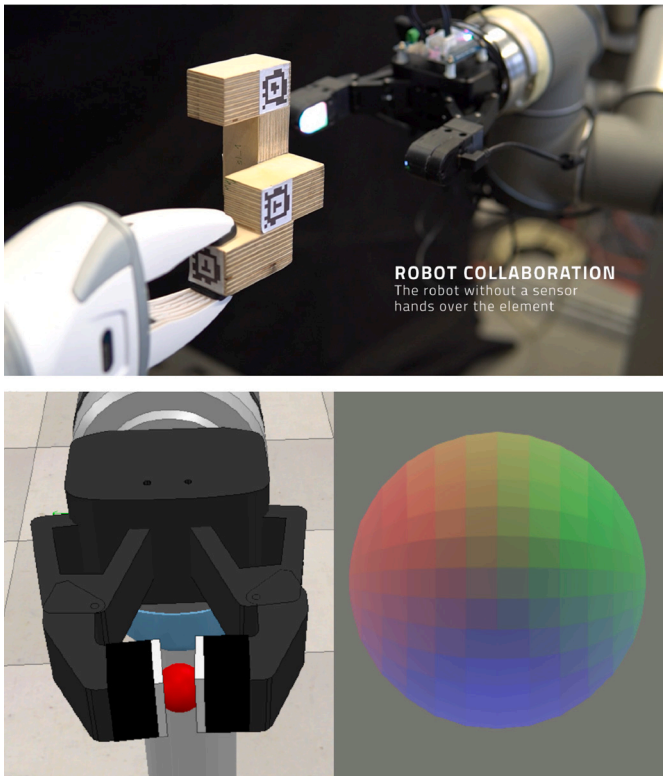

Fig. 10. Tactile sensor DIGIT. Top: Real gripper approaching a part. Bottom: Simulation of the gripper and RGB illumination providing detailed tactile feedback.

generally would like to develop more precise controllers. Therefore, the next section introduces tactile feedback that allows for finer control.

3.5. Simulation of the tactile sensor DIGIT

DIGIT [18] is a high-dimensional vision-based tactile sensor. Its design is based on GelSight [38] but DIGIT is significantly cheaper and easier to produce. It consists of an opaquely coated layer of gel illuminated from inside the sensor by three RGB-colored LEDs. These LEDs are placed inside the casing so that they illuminate the gel from different directions. The reflections of the LEDs' light from the gel is captured by an RGB camera. If the sensor touches an object, the gel deforms according to the geometry of the object. This deformation changes how the light of the LEDs reflects from the gel and thus the geometry of the touching object can be seen in the image captured by the camera. See Figs. 11 and 10.

In order to enable reinforcement learning in simulation, the DIGIT sensor needs to be integrated into PyBullet. We considered two approaches. One is based on simulating the light propagation and reflection [11,35]. While appealing for small objects such as the little sphere in Fig. 10 or dice in [32], this method does not work well for objects that cover the whole surface of the sensor, as is the case with our blocks. When large objects shift inside the gripper, this type of simulation is not able to capture any signal, whereas the real sensor does produce a signal because the silicone gel at the fingertip deforms. The other simulation approach is based on soft-body physics simulation. Modern physics engines do not yet support reliable and realistic soft-body simulation as required for robot control [21]. We were able to tune the soft-body simulation in PyBullet to resemble the real sensor, but this required very small time step, making the approach impractical for use with RL.

3.6. Learning on the real robot

Due to the aforementioned difficulties with the simulation of soft vision-based tactile sensors, we are investigating learning on the real robot, bypassing the simulation altogether. Sample sensor readings are shown in Fig. 11. Compared to the simulated signal in Fig. 10, there is much more nuance in the real data. Therefore, similar to the approaches [32] and [18], we aim to utilize these real signals directly for control by learning latent dynamics models.

Learning on the real system is different from learning in simulation. The agent can collect only a limited number of experiences and the robot can break easily if drastic actions are taken. Therefore, we pursue a model-based approach, where the agent is learning a transition model using experiences collected on the real system in a guided fashion. This is similar to how autonomous driving cars may learn to drive from human driving data. Instead of starting from scratch, we bootstrap the agent with a good initialization. The guiding procedure in our case consists in providing a few initial demonstrations (on the order of 10 trajectories) which are subsequently used by the agent to collect a larger

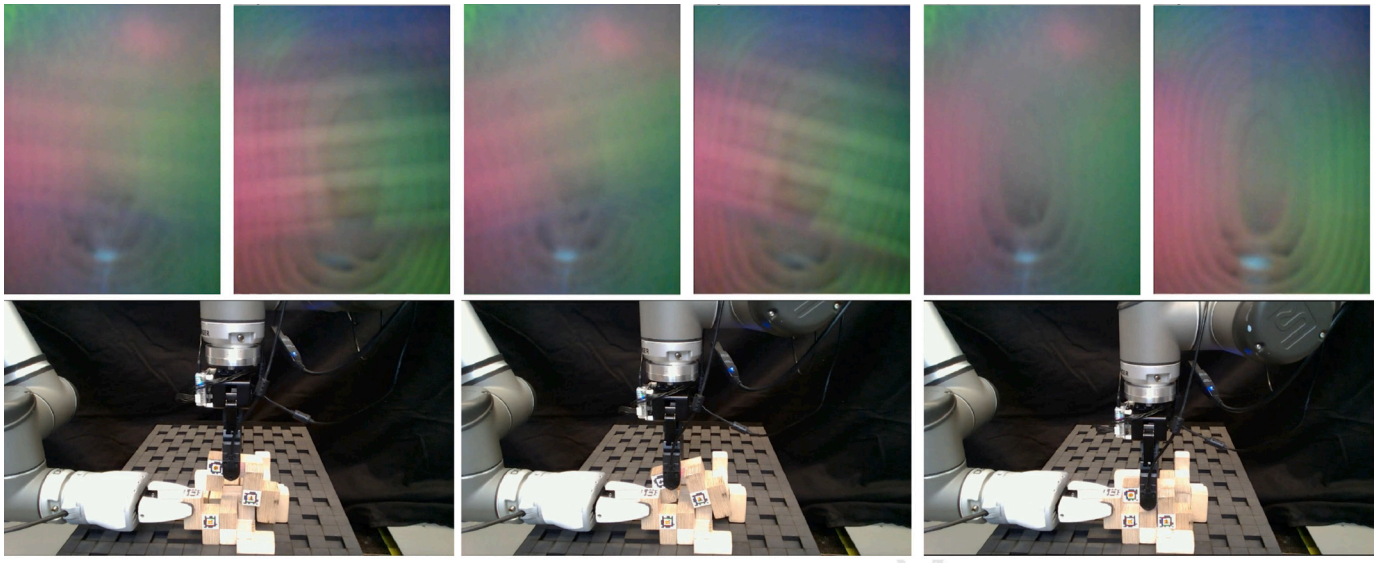

Fig. 11. Tactile sensor readings while learning on the real system. Compare the patterns of gel deformation on the left and on the right tactile sensors to the case where no contact is present (rightmost figure).

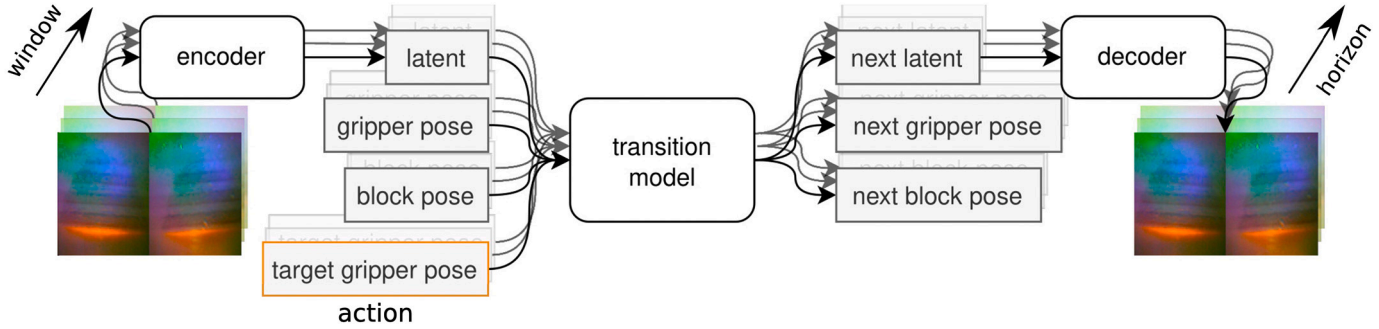

Fig. 12. Architecture of the model learned from the real-world data collected by the robot. As input the model receives a window of past observations, shown in the figure by the shadow going into the depth of the diagram. Each observation is multi-modal and consists of three parts: (i) two tactile image (left and right gripper fingers), (ii) gripper pose, (iii) poses of the manipulated blocks. Raw sensor observations pass through a learnable encoder before passing into the transition model. Given a window of such observations together with a window of actions, the transition model returns a window of future observation, which is called horizon in this case.

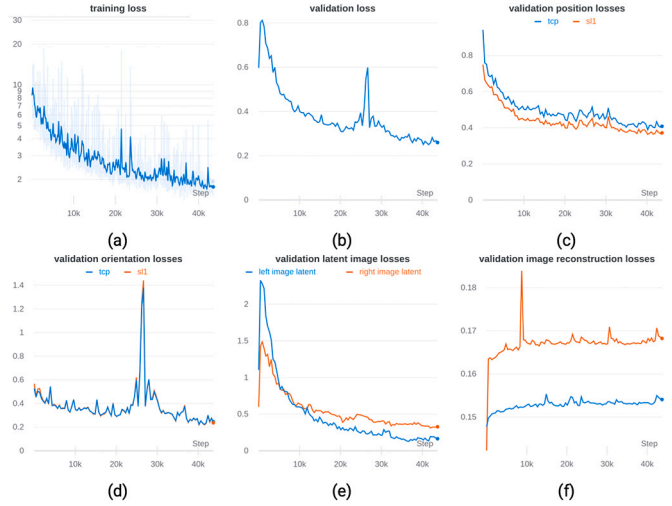

Fig. 13. Losses during the training of the transition model. All losses converge except the image reconstruction loss. The additional information of the other sensors could not be utilized to forward predict images well despite all other predictions matching the ground truth rather precisely.

dataset via replaying those trajectories with added perturbations. With this procedure, we build a dataset of 30,000 time steps recorded at 15 Hz. Each sample consists of the state of the robot and the state of the environment, as explained in Fig. 12. The encoder and decoder of the tactile images are trained jointly as a static autoencoder. Subsequently, the transition model is trained in a self-supervised manner.

Fig. 13 shows the learning curves for the transition model training. A window of 7 observations was fed as input to the transition model, and a horizon of 3 time steps was used for prediction. An autoencoder with ReLU activations and 16-dimensional latent space was trained on batches of size 16 with learning rate 0.0002. Although the model is able to predict the robot state and the state of the objects rather well, as seen from the position losses in Fig. 13, the error in the prediction of the tactile images is high, both in the latent space and after reconstruction. This may be attributed to the agent not needing to reconstruct the whole image in order to extract task-relevant information from it.

Once the model is learned, we run model predictive control using the cross-entropy method, similar to [18,32]. But performance varies significantly. Out of 30 trial executions on the real system, 19 finished with a successful joining of the SL-blocks, which requires the precision on the level of 1 mm. The unsuccessful executions terminated prematurely due to the blocks getting jammed. The reward function is given by the position and orientation error, but it does not include any punishment on hitting other blocks. As a result, the execution may deviate from the planned trajectory when multiple points of contact are encountered simultaneously, causing the blocks to get stuck. We view these experiments as encouraging, since they shows that the proposed method is feasible and can be applied on the real robot, but further investigation is required to improve the performance and reduce the variance in the results.

4. Conclusion and future work

In this paper, we addressed the problem of autonomous construction of modular structures. The current stage focuses on the simulation and training of basic assembly skills. At a later stage, this will be complemented by combining tactile skills and physical manipulation. Reinforcement learning was shown to be effective at learning closed-loop control policies to enable part stacking. However, the training time may become a bottleneck if more parts are added into the scene.

In order to construct arbitrary structures, a controller is needed that is able to deal with structures consisting of a variable number of blocks. Since every structure is different, the robot needs a way of sensing where other blocks of the structure are in order to place the next block without collisions. Here, the major challenge is to process an arbitrary number of block poses and achieve invariance to permutations. Both of these challenges could be tackled with Graph Convolutional Networks [17]. Alternatively, these challenges could be solved by using images of an RGB or depth camera as input to the policy instead. However, using camera images introduces a series of new challenges, as some blocks might be occluded and the robot additionally has to learn to interpret complex visual signals.

This project's long-term goal is to be able to construct structures using a real robot. Nevertheless, there is a number of challenges that need to be tackled when transferring our method to the real world. First of all, simulators can be fairly inaccurate when contact is happening, which makes transferring the policy to the real system challenging as it first has to be fine-tuned on the new system dynamics. Furthermore, some measurements that are easy to obtain in simulation are not directly available on the real system. One example is the poses of the parts that are used in the cost function and as a sensor input to the agent, which in reality require an optical tracking system to be estimated. Finally, the simulation of the DIGIT sensors is likely to be inaccurate, as the deformation of the gel is not simulated in detail. While we did some experiments with soft body simulation, we found the results to be unstable and inaccurate and hence decided to refrain from simulating deformation. Currently, the most promising approach with regards to learning based on tactile sensing is learning on the real system.

This paper presents first results of a research that intends to develop autonomous assembly through robots and reinforcement learning

algorithms. The full integration of the tactile sensor and its feedback will allow the robot to distinguish different building elements based on surface quality, weight and form. The integration of PyBullet as robot simulation environment into Rhino allows to use architectural requirements as cost measures for reinforcement learning algorithms. Equipping robots with these capabilities provides a step towards enabling autonomously built structures made from parts of various materials and functions suitable for assembly and re-assembly, contributing to a circular economy in architecture.

Supplementary data to this article can be found online at https://doi.org/10.1016/j.autcon.2021.104006.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 640554 and from the Forum for Interdisciplinary Research at TU Darmstadt for the project Multimaterial Modular Design Through Robot Learning and Tactile Sensing. We thank Schunk for providing access to the EGH gripper.

References

[1] A.A. Apolinarska, M. Pacher, H. Li, N. Cote, R. Pastrana, F. Gramazio, M. Kohler, Robotic assembly of timber joints using reinforcement learning, Autom. Constr. 125 (2021) 1–8, https://doi.org/10.1016/j.autcon.2021.103569.

[2] V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. Stachenfeld, P. Kohli, P. Battaglia, J. Hamrick, Structured agents for physical construction, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, vol. 97). PMLR, 2019, pp. 464–474, in: https://proceedings.mlr.press/v97/bapst19a.html.

[3] T. Bock, T. Linner, Robot Oriented Design: Design and Management Tools for the Deployment of Automation and Robotics in Construction, Cambridge University Press, 2015. ISBN: 9781107076389.

[4] G. Brugnaro, S. Hanna, Adaptive robotic carving, in: J. Willmann, P. Block, M. Hutter, K. Byrne, T. Schork (Eds.), Robotic Fabrication in Architecture, Art and Design 2018, Springer International Publishing, Cham, 2019, pp. 336–348. ISBN: 978-3-319-92294-2.

[5] R. Calandra, A. Owens, D. Jayaraman, J. Lin, W. Yuan, J. Malik, E.H. Adelson, S. Levine, More than a feeling: learning to grasp and regrasp using vision and touch, IEEE Robot. Autom. Lett. 3 (4) (2018) 3300–3307, https://doi.org/10.1109/LRA.2018.2852779.

[6] E. Coumans, Y. Bai, Pybullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning, 2016–2021. https://pybullet.org (Retrieved 06.09.21).

[7] M.P. Deisenroth, G. Neumann, J. Peters, A survey on policy search for robotics, Found. Trends® Robot. 2 (1–2) (2013) 1–142, https://doi.org/10.1561/2300000021.

[8] J.A. Fishel, G.E. Loeb, Sensing tactile microvibrations with the biotac – comparison with human sensitivity, in: 2012 4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob), IEEE, 2012, pp. 1122–1127.

[9] S. Fujimoto, H. van Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, vol. 80). PMLR, 2018, pp. 1587–1596, in: https://proceedings.mlr.press/v80/fujimoto18a.html.

[10] N. Funk, G. Chalvatzaki, B. Belousov, J. Peters, Learn2assemble with structured representations and search for robotic architectural construction, in: Proceedings of the 5th Annual Conference on Robot Learning, 2021. https://openreview.net/forum?id=wBT0lZJAJ0V.

[11] D.F. Gomes, P. Paoletti, S. Luo, Generation of gelsight tactile images for sim2real learning, IEEE Robot. Autom. Lett. 6 (2) (2021) 4177–4184, https://doi.org/10.1109/LRA.2021.3063925.

[12] F. Gramazio, M. Kohler, V. Helm, S. Ercan, In-situ robotic construction. Extending the digital fabrication chain in architecture, in: Synthetic Digital Ecologies: Proceedings of the 32nd Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA, San Francisco, CA, 2012, pp. 169–176. ISBN: 978-1-62407-267-3.

[13] V. Hartmann, O. Oguz, D. Driess, M. Toussaint, A. Menges, Robust task and motion planning for long-horizon architectural construction planning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 – January 24, 2021. IEEE, 2020, pp. 6886–7689, https://doi.org/10.1109/IROS45743.2020.9341502.

[14] T. Inoue, G.D. Magistris, A. Munawar, T. Yokoya, R. Tachibana, Deep reinforcement learning for high precision assembly tasks, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24–28, 2017. IEEE, 2017, pp. 819–825, https://doi.org/10.1109/IROS.2017.8202244.

[15] B. Jenett, A. Abdel-Rahman, K.C. Cheung, N. Gershenfeld, Material-robot system for assembly of discrete cellular structures, IEEE Robot. Autom. Lett. 4 (4) (2019) 4019–4026, https://doi.org/10.1109/LRA.2019.2930486.

[16] Z. Kappasov, J.A. Corrales, V. Perdereau, Tactile sensing in dexterous robot hands – review, Robot. Auton. Syst. 74 (2015) 195–220, https://doi.org/10.1016/j.robot.2015.07.015.

[17] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings. OpenReview.net, 2017. https://openreview.net/forum?id=SJU4ayYgl.

[18] M. Lambeta, P.-W. Chou, S. Tian, B. Yang, B. Maloon, V.R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, D. Jayaraman, R. Calandra, Digit: a novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation, IEEE Robot. Autom. Lett. 5 (3) (2020) 3838–3845, https://doi.org/10.1109/LRA.2020.2977257.

[19] S. Levine, N. Wagener, P. Abbeel, Learning contact-rich manipulation skills with guided policy search, in: IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26–30 May, 2015. IEEE, IEEE, 2015, pp. 156–163, https://doi.org/10.1109/ICRA.2015.7138994.

[20] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: Y. Bengio, Y. LeCun (Eds.), 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings, 2016. arXiv:1509.02971.

[21] J. Matas, S. James, A.J. Davison, Sim-to-real reinforcement learning for deformable object manipulation, in: 2nd Annual Conference on Robot Learning, CoRL 2018. Proceedings (Proceedings of Machine Learning Research, vol. 87). PMLR, Zürich, Switzerland, 29–31 October 2018, 2018, pp. 734–743, in: http://proceedings.mlr.press/v87/matas18a.html.

[22] R. McNeel, Rhinoceros, 1993–2021. https://www.rhino3d.com/ (Retrieved 06.09.21).

[23] N. Melenbrink, J. Werfel, A. Menges, On-site autonomous construction robots: towards unsupervised building, Autom. Constr. 119 (2020) 103312, https://doi.org/10.1016/j.autcon.2020.103312.

[24] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, L. Zhang, Solving Rubik's Cube With a Robot Hand, 2019. arXiv:1910.07113 (Arxiv).

[25] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, N. Dormann, Stable Baselines3, 2019. https://github.com/DLR-RM/stable-baselines3 (Retrieved 06.09.21).

[26] C. Robotics, Semi-Automated Mason (SAM), 2007–2021. https://www.construction-robotics.com/sam-2/ (Retrieved 06.09.21).

[27] F. Roche, S. Lavaux, J. Navarro, Olzweg, 2006. https://new-territories.com/welostit.htm (Retrieved 06.09.21).

[28] G. Rossi, P. Nicholas, Re/learning the wheel: methods to utilize neural networks as design tools for doubly curved metal surfaces, in: ACADIA // 2018: Recalibration. On Imprecision and Infidelity, 2019, pp. 146–155. http://2018.acadia.org/schedule.html. , ADIA 2018: Recalibration. On imprecision and infidelity; Conference date: 18-10-2018 Through 20-10-2018.

[29] J. Rowe, Life in a Computerised Environment. Electronics Australia, 1972. Identifier: ea197203_201912, September.

[30] G. Sartoretti, Y. Wu, W. Paivine, T.K.S. Kumar, S. Koenig, H. Choset, Distributed reinforcement learning for multi-robot decentralized collective construction, in: N. Correll, M. Schwager, M.W. Otte (Eds.), Distributed Autonomous Robotic Systems, The 14th International Symposium, DAR 2018 (Springer Proceedings in Advanced Robotics, vol. 9), 9, Springer, Boulder, CO, USA, 2018, pp. 35–49, https://doi.org/10.1007/978-3-030-05816-6. October 15–17, 2018.

[31] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018. ISBN: 9780262364010.

[32] S. Tian, F. Ebert, D. Jayaraman, M. Mudigonda, C. Finn, R. Calandra, S. Levine, Manipulation by feel: touch-based control with deep predictive models, in: International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20–24, 2019. IEEE, 2019, pp. 818–824, https://doi.org/10.1109/ICRA.2019.8794219.

[33] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, J. Peters, Stable reinforcement learning with autoencoders for tactile and visual data, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9–14, 2016. IEEE, 2016, pp. 3928–3934, https://doi.org/10.1109/IROS.2016.7759578.

[34] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J.P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T.L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T.P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver, Grandmaster level in starcraft II using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354, https://doi.org/10.1038/s41586-019-1724-z.

[35] S. Wang, M. Lambeta, L. Chou, R. Calandra, Tacto: A Fast, Flexible and Open-Source Simulator for High-Resolution Vision-Based Tactile Sensors, 2020. arXiv:2012.08456 (Arxiv).

[36] B. Wibranek, Y. Liu, N. Funk, B. Belousov, J. Peters, O. Tessmann, Reinforcement learning for sequential assembly of sl-blocks: self-interlocking combinatorial design based on machine learning, in: Proceedings of the 39th eCAADe Conference, vol. 1, CUMINCAD, 2021, pp. 27–36. http://papers.cumincad.org/cgi-bin/works/paper/ecaade2021_247 (ISBN: 978-94-91207-22-8).

[37] H. Yousef, M. Boukallel, K. Althoefer, Tactile sensing for dexterous in-hand manipulation in robotics – a review, Sens. Actuators A Phys. 167 (2) (2011) 171–187, https://doi.org/10.1016/j.sna.2011.02.038. Solid-State Sensors Actuators and Microsystems Workshop.

[38] W. Yuan, S. Dong, E.H. Adelson, Gelsight: high-resolution robot tactile sensors for estimating geometry and force, Sensors 17 (12) (2017) 2762, https://doi.org/10.3390/s17122762.