Robot Learning for Autonomous Assembly

Chia-Man Hung

Autonomous Intelligent Machines & Systems CDT
Oxford Robotics Institute
University of Oxford
chiaman@robots.ox.ac.uk

Abstract

We investigate autonomous assembly from a random initiation on Human Support Robot (HSR), with the aim of solving the Siemens Assembly Challenge. We formulate it as a Reinforcement Learning (RL) problem with sparse reward. A new learning paradigm, Scheduled Auxiliary Control (SAC), excels at sparse reward RL and appears to be a great fit to our problem. The key idea behind SAC is the high-level scheduling of auxiliary tasks and the execution of auxiliary policies to explore efficiently. We experiment in ROS-based Gazebo simulated environments built on top of the OpenAI Gym, with the potential of being extended to learn the same tasks on a real robot from scratch.

1 Introduction

A recent trend in manufacturing is shaping the future of industry towards small production volume and high product variability (Lasi et al., 2014), motivating a high degree of automation in flexible environments. To date, robots used in assembly tasks often operate in tightly controlled environments; initial conditions are predefined, parts are sorted and fed in a fixed manner, and guides are used for assuring the consistent behaviour of such repetitive tasks. Robots with predefined behaviours under human control can perform impressive tasks. However, designing the controller for autonomous operations remains a major challenge, even for basic tasks such as grasping objects.

Robotic assembly typically involves object manipulation tasks with contact and friction, such as inserting and squeezing tight-fitting objects, which are particularly hard for analytical models – a requirement for robotic systems with limited sensing. For current robots with access to a rich sensory stream, an infamous difficulty towards autonomous manipulation still lies in dealing with the inevitable uncertainty and noise in the sensory-motor system and the environment.

Over the past few years, a couple of approaches have been developed to tackle robotic manipulation problems, including associative skill memories (ASM) (Pastor et al., 2012) (Kappler et al., 2015) and deep reinforcement learning (DRL) (Levine et al., 2015) (Levine et al., 2016) (Tamar et al., 2017).

In the context of Reinforcement Learning (RL) in robotic manipulation applications, a natural problem of exploration occurs. Typically, an agent has to discover a long sequence of actions to find a configuration – assemble a whole set of components in a specific configuration – that yields the sparse reward. A multitude of methods have been proposed to cope with this problem, but they all rely on the availability of prior knowledge that is specific to the task. Scheduled Auxiliary Control (SAC) (Riedmiller et al., 2018) supports the agent during learining, but preserves the ability of the agent to learn from sparse rewards. Moreover, it has been applied to robotic manipulation tasks – stacking various objects and cleaning a table – with demonstrated success.

In this work, we extend the OpenAI Gym (Brockman et al., 2016) for robotics using the Robot Operating System (ROS) (Quigley et al., 2009) and the Gazebo simulator (Koenig and Howard, 2006)

Preprint. Work in progress.

(Zamora et al., 2016) to create simulated environments for the Toyota Human Support Robot (HSR)[1]. We summarise and apply the sequential version of Scheduled Auxiliary Control (SAC-Q), a new learning paradigm that enables learning of complex behaviours, in the presence of multiple sparse reward signals. We experiment with a *simple* task of stacking two boxes, with the aim of performing autonomous assembly of a whole set of gears in the Siemens challenge[2].

2 Related Work

Research on autonomous assembly has been ongoing for a few decades. Early works (De Mello and Sanderson, 1991) (Halperin et al., 2000) focus on planning a sequence of assembly steps or motions based on the geometry of different components. More recent work on automated furniture assembly system (Knepper et al., 2013) relies on user-supplied geometric specification and reasoning about the individual parts to deduce how they fit together. (Heger, 2010) examines more realistic scenarios and explores error detection and recovery strategies when assembly plans fail. These traditional methods cannot learn to perform autonomous assembly from scratch when the geometry specification is unknown.

One of the key ideas behind SAC – the use of auxiliary tasks in the context of RL to overcome sparse rewards – has been explored in the literature. Among the pioneers of this idea, the Horde architecture (Sutton et al., 2011) applies reinforcement learning to identify value functions for distinct pseudo-rewards and each value function is trained separately using distinct weights. It is further generalised by the Universal Value Function Approximators (UVFA) architecture (Schaul et al., 2015), a factored representation of a continuous set of optimal value functions, combining features of the state with an embedding of the pseudo-reward function. Recent work on Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) samples additional goals from previous trajectories and uses them as auxiliary tasks.

HER and SAC may be seen as an implicit form of curriculum learning, first introduced by Bengio et al. (2009). The main concept behind curriculum learning is that humans and animals learn much better when the examples are not randomly presented but organised in a meaningful order. Heess et al. (2017) combine this idea with RL on complex locomotion tasks. Similarly, divide-and-conquer RL (Ghosh et al., 2017) separates complex tasks into a set of local tasks, each of which can be used to learn a separate policy, and demonstrates their method on a set of robotic manipulation and locomotion tasks in simulated environments. Intrinsically motivated goal exploration processes (Forestier et al., 2017) automatically generate a learning curriculum by generating, selecting goals based on intrinsic rewards and reusing information systematically to improve goals. This method has been evaluated with a real robotic setup, which provides real world constraints that are not available in simulations.

Besides curriculum learning, other approaches of deep RL have been proposed to solve robotic manipulation tasks. Examples include guided policy search (Levine et al., 2015), (Levine et al., 2016) and using model predictive control to generate hindsight plan and replanning at each time step (Tamar et al., 2017). All three papers have conducted experiments in real environments using a real PR2 robot.

Perhaps the closest to our tasks is the recent work on Learning Robotic Assembly from CAD (Thomas et al., 2018). They attempt the same Siemens challenge of autonomous assembly using a PR2 robot. Their main idea is to leverage the prior knowledge of the geometry specification specified in the CAD design files. In addition, they propose a neural network architecture that can learn to track the motion plan, thereby generalising the assembly controller to changes in the object positions.

3 Background

We model the problem of reinforcement learning in a Markov Decision Process $(\chi, \mathcal{A}, \gamma, p, r)$. χ is the state space, $\mathcal{A}$ the action space, $\gamma \in [0, 1)$ the discount factor, p the transition function mapping state-action pairs $(s, a) \in \chi \times \mathcal{A}$ to distributions over χ, and $r : (s, a) \in \chi \times \mathcal{A} \to \mathbb{R}$ is the reward function. Actions are sampled from a policy distribution π mapping from χ to distributions over $\mathcal{A}$.

[1] https://www.toyota-global.com/innovation/partner_robot/robot/#link02

[2] https://www.siemens.com/us/en/home/company/fairs-events/robot-learning.html

The goal of reinforcement learning is to maximise the sum of discounted rewards $\mathbb{E}_\pi[R(\tau_{0:\infty})] = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t), s_0 \sim p(s)]$, where $p(s)$ denotes the initial state distribution or the state visitation distribution. We denote with $\tau_{t:\infty} = (s_t, a_t), ...$ the trajectory starting at time t in state s_t.

Policy Gradient Methods Many classical RL methods are action-value methods; they learn the value of actions and then select actions based on their estimated action values. In policy gradient methods, we consider methods that learn a parameterised policy that can select actions without consulting a value function. They use feedback from the environment directly to optimise the policy. Monte Carlo returns are often used, but one drawback is that they introduce high variance.

Actor-Critic Methods To solve the problem of high variance for policy learning, a natural extension consists of learning a parameterised policy and a parameterised value function. They are called the actor and the critic respectively. The actor learns a policy and updates its parameters in a direction suggested by the critic, while the critic evaluates the policy using policy evaluation methods. Through bootstrapping (updating the value estimate for a state from the estimated values of subsequent states), we introduce bias and asymptotic dependence on the quality of the function approximation. The bias introduced here is beneficial as it usually reduces variance and accelerates learning.

4 Scheduled Auxiliary Control

We now summarise the method Scheduled Auxiliary Control (Riedmiller et al., 2018) for RL in sparse reward problems.

It is a variation of actor-critic methods with the actor parameterised by θ and the critic by ϕ. The sparse reward problem is defined as finding the optimal policy in an MDP $\mathcal{M}$. To enable learning, a set of auxiliary tasks are used, characterised by a set of auxiliary MDPs $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_K\}$. These auxiliary MDPs share the state space, the action space and the transition dynamics with the main MDP $\mathcal{M}$, but have separate auxiliary reward functions $r_{\mathcal{A}_1}(s, a), ..., r_{\mathcal{A}_K}(s, a)$.

Learning the Policy (Actor θ) The action-value function $Q_\mathcal{T}(s_t, a_t)$ for task $\mathcal{T}$ is defined as

$$Q_\mathcal{T}(s_t, a_t) = r_\mathcal{T}(s_t, a_t) + \gamma \mathbb{E}_{\pi_\mathcal{T}} \left[\sum_{t'=t+1}^{\infty} \gamma^{t'} r_\mathcal{T}(s_{t'}, a_{t'}) \right], \tag{1}$$

where $\mathcal{T} \in \mathcal{A} \cup \{\mathcal{M}\}$, $\pi_\mathcal{T} = \pi_\theta(a|x, \mathcal{T})$.

To learn the parameters, we optimise

$$\mathcal{L}(\theta) = \mathcal{L}(\theta; \mathcal{M}) + \sum_{k=1}^{|\mathcal{A}|} \mathcal{L}(\theta; \mathcal{A}_k), \tag{2}$$

with $\mathcal{L}(\theta; \mathcal{T}) = \sum_{\mathcal{B} \in \mathcal{A} \cup \{\mathcal{M}\}} \mathbb{E}_{p(s|\mathcal{B})} \left[Q_\mathcal{T}(s, a)|a \sim \pi_\theta(\cdot|s, \mathcal{T}) \right]$.

The $Q_\mathcal{T}(s, a)$ values are obtained using a parameterised predictor $\hat{Q}^\pi_\mathcal{T}(s, a; \phi)$. Using this predictor together with a replay buffer B containing gathered trajectories τ, a gradient based approach can be taken.

$$\nabla_\theta \mathcal{L}(\theta) \approx \sum_{\mathcal{T} \in \mathcal{A} \cup \{\mathcal{M}\}, \tau \sim B} \nabla_\theta \mathbb{E}_{\pi_\theta(\cdot|s_t, \mathcal{T}), s_t \in \tau} \left[\hat{Q}^\pi_\mathcal{T}(s_t, a; \phi) + \alpha \log \pi_\theta(a|s_t, \mathcal{T}) \right], \tag{3}$$

where $\nabla_\theta \mathbb{E}_{\pi_\theta(\cdot|s_t, \mathcal{T}), s_t \in \tau} \left[\log \pi_\theta(a|s_t, \mathcal{T}) \right]$ corresponds to entropy regularization with weighting factor α.

Learning the Q-function (Critic ϕ) Since the policy parameter is constantly being updated, the trajectories are generated by different behaviour policies. The off-policy evaluation Retrace (Munos et al., 2016) is used to optimise the estimator $\hat{Q}^\pi_\mathcal{T}(s, a; \phi)$.

Learning the Scheduler The scheduler is used to determine the current intention of the agent based on previous intentions. Denote ξ by the period at which the scheduler switches tasks, H the total number of possible tasks and $\mathcal{T}_{0:H-1} = \{\mathcal{T}_0, ..., \mathcal{T}_{H-1}\}$ the H scheduling choices made within an episode. The return of the main task is then defined as

$$R_{\mathcal{M}}(\mathcal{T}_{0:H-1}) = \sum_{h=0}^{H} \sum_{t=h\xi}^{(h+1)\xi-1} \gamma^t r_{\mathcal{M}}(s_t, a_t), \tag{4}$$

where $a_t \sim \pi_\theta(\cdot|s_t, \mathcal{T}_h)$.

Denote the scheduling policy with $P_{\mathcal{S}}(\mathcal{T}|\mathcal{T}_{0:h-1})$. The probability of an action a_t can be defined as

$$\pi_{\mathcal{S}}(a_t|s_t, \mathcal{T}_{0:h-1}) = \sum_{\mathcal{T}} \pi_\theta(a_t|s_t, \mathcal{T}) P_{\mathcal{S}}(\mathcal{T}|\mathcal{T}_{0:h-1}). \tag{5}$$

Combining these two definitions from Equations 4 and 5, we optimise the objective below to learn the scheduler

$$\mathcal{L}(\mathcal{S}) = \mathbb{E}_{P_{\mathcal{S}}} \left[R_{\mathcal{M}}(\mathcal{T}_{0:H-1}|\mathcal{T}_h \sim P_{\mathcal{S}}(\mathcal{T}|\mathcal{T}_{0:h-1})) \right]. \tag{6}$$

The solution to Equation 6 can be approximated by the Boltzmann distribution

$$P_{\mathcal{S}}(\mathcal{T}_h|\mathcal{T}_{1:h-1;\eta}) = \frac{\exp(\mathbb{E}_{P_{\mathcal{S}}}[R_{\mathcal{M}}(\mathcal{T}_{h:H})]/\eta)}{\sum_{\bar{\mathcal{T}}_{h:H}} \exp(\mathbb{E}_{P_{\mathcal{S}}}[R_{\mathcal{M}}(\bar{\mathcal{T}}_{h:H})]/\eta)}, \tag{7}$$

where η corresponds to the greediness of the schedule. The expectation of the return can be approximated by the Monte Carlo return using the last executed trajectories.

The actor and the critic are represented by two neural networks as in Appendix B additional model details in the paper. The complete SAC-Q algorithm can be found in Appendix C in the paper.

5 Experiments

We perform experiments based on Human Support Robot (HSR)[3] in simulation. Human Support Robots are robots developed by Toyota that offer broad-based assistance in daily life. They are designed for basic support such as picking up and carrying objects with potential applications in nursing and healthcare.

5.1 Experimental Setup and Implementation Details

OpenAI Gym environments for HSR using ROS and Gazebo (link to source code[4])

OpenAI Gym (Brockman et al., 2016) is a toolkit for developing and comparing reinforcement learning algorithms that has recently gained popularity in the RL research community and become a benchmark. To interface it, we define and overwrite a few functions, including *step*, *reset*, *render*, *close*. Real-world operation is achieved combining Gazebo simulator (Koenig and Howard, 2006), a physics engine, 3D modeling and rendering tool, with ROS (Quigley et al., 2009), a collection of software frameworks for robot software development. As benchmarking in robotics remains an unsolved issue, this integration (Zamora et al., 2016) provides a controlled environment with a real robotic (HSR) API enabling fast experiments and development.

At the start and after every *reset* of the simulation, the robot is placed in a neutral configuration facing the table. The objects (boxes or gears) are dropped at random positions on the table. The robot can perform action *step*s to move in six directions – forward, backward, up, down, left, right – and to open or close the gripper. Under the hood, these actions are implemented using the HSR library *hsrb_interface*, which provides a wrapper around an inverse kinematics library. After every *step*, three things are returned – an observation of the current state, a scalar reward of the main task and a boolean indicating whether the episode is terminal. At the end of the simulation, we need to *close* the environment to kill running processes.

[3] https://www.toyota-global.com/innovation/partner_robot/robot/#link02

[4] https://github.com/ascane/gym-gazebo-hsr

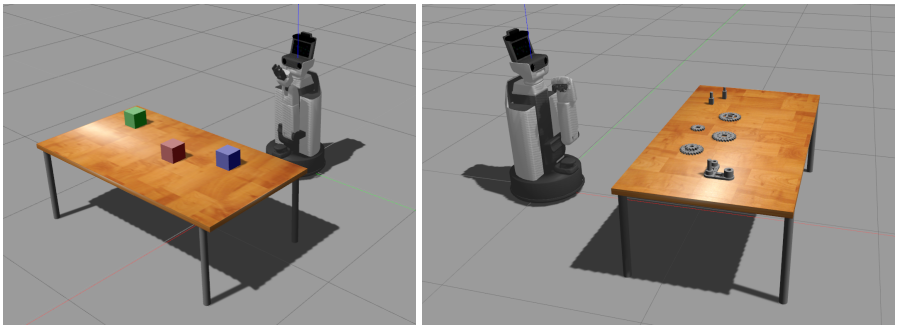

Figure 1: Two HSR simulated environments in Gazebo. Left: Stacking two boxes. Right: Siemens assembly challenge.

SAC-Q on HSR (link to source code[5])

Multiple design choices have been made when implementing SAC-Q on HSR.

- □ Learning the Policy (Actor θ)

  In Equation 3, the entropy regularisation term is supposed to be non negative by definition. Therefore, we opt for the change of the sign before the weighting factor α and keep α non negative. The equation for computing the gradient for the policy update becomes

$$\nabla_\theta \mathcal{L}(\theta) \approx \sum_{\mathcal{T} \in \mathcal{A} \cup \{\mathcal{M}\}, \tau \sim B} \nabla_\theta \mathbb{E}_{\pi_\theta(\cdot|s_t,\mathcal{T}), s_t \in \tau} \left[\hat{Q}^\pi_\mathcal{T}(s_t, a; \phi) - \alpha \log \pi_\theta(a|s_t, \mathcal{T}) \right]. \quad (8)$$

- □ Learning the Q-function (Critic ϕ)

  In Equation (13) of the SAC paper (Riedmiller et al., 2018), from the second line, the i is used but never defined. From our understanding, this value indicates the index of the timestep in the trajectory from which the Q return is calculated. The paper does not specify how this is chosen. In our implementation, 32 trajectories are randomly sampled from the replay buffer and then a random i is chosen in that trajectory.

  The off-policy evaluation formula in the SAC paper differs from the original Retrace paper (Munos et al., 2016). In the SAC paper, $\delta_Q(s_i, s_j)$ is defined as $\mathbb{E}_\pi[Q(s_i, \cdot)] - Q(s_j, a_j)$ while in the Retrace paper it is defined as $\mathbb{E}_\pi[Q(s_{j+1}, \cdot)] - Q(s_j, a_j)$. We follow the original Retrace paper in our implementation.

- □ Learning the Scheduler

  How to approximate the schedule returns in the SAC paper is unclear to us. In Equation 12 of the paper, it can be approximated from a Monte Carlo estimation of the expectation using the last $M = 50$ trajectories, which differs from Algorithm 3 of the paper where M is continuously increased as the task is visited more often. This causes the size of the gradient to decrease, rendering it useless by the time any task has actually been learned. We adopt a constant learning rate to replace $\frac{1}{M}$ instead.

5.2 Stacking Two Boxes

As something *simple* to start with, we consider the task of stacking the green box on top of the red one. This requires the robot to perform the following actions sequentially – move close to the table to be able to reach objects on it, grasp the green box placed at a random positon, lift it up to at least the height of the red one and place it on top of it in a stable configuration. A screenshot of the simulation is shown in Figure 1 on the left-hand side.

We design three auxiliary tasks with sparse rewards – Reach, Move, Lift. In the Reach task, we receive a positive reward when the hand is close enough to the green box. In the Move task, a reward is attributed based on the horizontal move of the green box. In the Lift task, only when the green box is lifted higher than a certain threshold is a reward given.

[5] https://github.com/ascane/sacq-hsr

The observation of the state is designed to be the left and the right images taken from the robot's head and the state of the joints. They are transformed into a vector when being fed into the neural networks.

5.3 Siemens Assembly Challenge

Researchers at Siemens Corporate Technology in Berkeley, CA, have developed a set of gears to test different robot learning approaches to assembly[6]. There are seven components and the goal is to assemble them in to the end configuration as shown in Figure 2. Different from the previous task, there is more than one possible sequence to perform this task. To acheive this, the robot has to learn to move close to the table, grab the gears and assemble them. A screenshot of the simulation is shown in Figure 1 on the right-hand side.

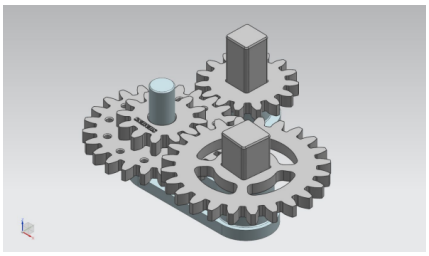

Figure 2: The desired end configuration of the Siemens assembly challenge. Credits: Siemens Corporate Technology.

We design the auxiliary task to be grasping and lifting the six components (except the ground base that stays on the table), and moving the components close to each other.

The observation stays the same as in the previous task. One difficulty is to determine whether the gears are assembled properly from the observation.

5.4 Challenges

At the time of writing, SAC has been run locally in the OpenAI Gym environments for HSR successfully. However, it has not been trained to solve our tasks. Besides technical difficulties of setting up a Docker container to train on a server with GPU, we have not been able to run Gazebo simulator as fast as realtime, even without the rendering (gzclient). It takes around 10 minutes to perform 100 actions locally. To run 10k episodes of 100 action steps each, a rough estimate suggests it would take approximately 10 weeks just to perform the actions, which cannot be accelerated by the use of a GPU.

6 Conclusion

Autonomous assembly remains a major challenge in robot learning. In this work, we extended the OpenAI Gym for robotics using the Robot Operating System (ROS) and the Gazebo simulator to create simulated environments for the Human Support Robot. We gave an overview and provided a working implementation of the Scheduled Auxiliary Control algorithm. We applied it to solve the task of stacking two boxes and the Siemens assembly challenge. Many algorithmic and technical problems encountered along the way are solved. However, our current simulated environment needs a speed-up to enable training in a reasonable amount of time. Experimenting in a ROS-based environment will allow us to learn from scratch on a real robot in the future.

7 Acknowledgements

The author would like to thank Ioannis Havoutis, Sasha Salter and Ingmar Posner for their continuous support and useful discussions throughout this project.

References

H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal. Towards associative skill memories. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 309–315. IEEE, 2012.

[6]https://www.siemens.com/us/en/home/company/fairs-events/robot-learning.html

D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, and S. Schaal. Data-driven online decision making for autonomous manipulation. In *Robotics: Science and Systems*, 2015.

S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 156–163. IEEE, 2015.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel. Learning from the hindsight plan—episodic mpc improvement. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 336–343. IEEE, 2017.

M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, N. Heess, and T. Springenberg. Learning by playing - solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

N. Koenig and A. Howard. Gazebo-3d multiple robot simulator with dynamics, 2006.

I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.

L. H. De Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE transactions on Robotics and Automation*, 7(2):228–240, 1991.

D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26(3-4):577–601, 2000.

R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 855–862. IEEE, 2013.

F. W. Heger. Assembly planning in constrained environments: Building structures with multiple mobile robots. 2010.

R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

S. Forestier, Y. Mollard, and P.-Y. Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel. Learning robotic assembly from cad. *arXiv preprint arXiv:1803.07635*, 2018.

R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.