Emergence of Tool Construction and Tool Use Through Hierarchical Reinforcement Learning

Qinbo Li[a,b], Yoonsuck Choe[a]

[a] *Department of Computer Science and Engineering*
Texas A&M University
College Station, TX 77843-3112 USA
[b] *Meta Inc.*
1101 Dexter Ave N
Seattle, WA 98109, USA

Keywords: Tool use; Tool construction; Reinforcement learning; Sensorimotor learning

1. Introduction

Tool use is often considered a key indicator of intelligence and complex cognition [1, 2, 3]. The most advanced form of tool use can be found only in humans. However, rudimentary tool use behavior has been observed in multiple non-human species, ranging from insects to great apes [4]. In many cases, tool use in animals is limited to the manipulation of a single object (such as a stone or a stick), but tool construction (or manufacture) capabilities have also been documented. Apes (including chimpanzees, orangutan, and others) are well known for such capabilities, and there is an extensive literature on the subject (see chapter 6 in [4] for a comprehensive review). Such rich tool use behavior is not only restricted to the species closest to humans. For example, New Caledonian crows can bend a wire to create a hook to retrieve food from a deep cylinder [5], they can also assess the 'value' of the tool (e.g., hooked vs. straight) [6], and they can even construct compound tools by connecting multiple objects [7].

Email addresses: lee@tamu.edu (Qinbo Li), choe@tamu.edu (Yoonsuck Choe)

Compared to the large body of research on tool use in animals, there is relatively fewer works in the AI field. A quick search on Google Scholar turns up 44,600 results for animal tool use, but only about 8,000 for tool use in artificial intelligence or robots (as of March 2022). An early paper on this topic by St. Amant and Wood raised the importance of tool use in the context of autonomous agents, where they introduced the "Tooling test", to replace the Turing test [2]. In a follow-up work, we observed that investigating tool use in artificial agents can lead to a breakthrough in the field ("Tooling test rebooted") [3]. A more recent extension of these ideas can be found in Nair et al. ("Tool MacGyvering") [8], and in Allen et al. ("The Virtual Tools Game") [9]. Finally, tool construction and use is especially interesting in the context of open-ended improvement in AI (see [10]), where the complexity of the tool and the cognitive capacity of the agent can form a co-evolutionary relationship (see [11] on co-evolution in neuroevolution).

Inspired by these works, we conducted a series of tool use and tool construction experiments in a simulated environment. The tasks in our prior works included the basic use of a stick for reaching [12, 13, 14], connecting two sticks for a longer reach (tool construction) [15], and grabbing a tool and dragging objects [16] using different tool types (I-shaped, L-shaped, and T-shaped) [17]. Different learning algorithms were used, depending on the task requirements. We started with neuroevolution (NEAT, [18]), and gradually moved to the deep reinforcement learning (e.g., ACKTR [19]), to fully utilize the visual input. In these works, we observed effective utilization of simple tools in the environment, and rudimentary tool construction. Our trained agents also exhibited novel emergent behaviors such as sweeping, throwing, and hitting to achieve the goal [17], without those specific objectives prescribed in the learning algorithm.

In this chapter, we build upon our prior work to investigate tool construction and use in a realistic physics simulation environment. Although we have investigated primitive tool construction in [15] and tool use in a realistic physics simulation environment [17, 16], we have not combined these two aspects, so the combination of the two will be our main focus and contribution.

The rest of this chapter is organized as follows: in section 2, we introduce some background on reinforcement learning, and some previous works related with learning to use tools, including our own previous work. We then introduce our proposed approach in section 3, followed by our experiments and results in section 4. The remaining sections are discussion (including future works) and conclusion in section 5.

2. Background and Related Works

In this section, we briefly review existing works on tool use in AI and robotics, and provide some preliminaries on reinforcement learning.

2.1. Tool Use in AI and Robotics

Tool use has recently gained attention in artificial intelligence and robotics (for a review, see 2), however tool construction is a relatively unexplored area. The various existing works on tool use include (1) programmed, hard-coded behavior [20]; (2) learning through demonstration [21, 22, 23, 24, 25]; (3) learning affordances via random trial-and-error or body babbling [26, 27, 28]; (4) tool use based on tool-body assimilation [29, 30, 31]. (5) Bayesian learning of tool affordances [32]; (6) Evolved tool using behavior [33, 34]. (Cf. 35 where body morphology [not tools] was co-evolved with the controller.); and (7) Deep reinforcement learning based tool use, with dexterous manipulations [36].

However, most of the works listed above depended on some degree of designer knowledge regarding tool use and motor control, e.g., fully hard coded behavior, the tool being pre-attached to the limb, pre-defined tool features, pre-defined motor primitives, etc. Evolution-based approaches [33, 34] were relatively free of these constraints, but in those cases the tools were more or less simple markers, not something than can be manipulated with a limb-like structure of the agent.

AI-based work on tool construction has been very rare, with notable exceptions. Wang et al. [37] took a synthetic approach to construct tools, but their focus was more on understanding the various mechanistic and energetic needs

of using the synthetically generated tools, not on tool construction by agents. In our own work (Reams and Choe, [15]), we demonstrated the construction and use of an extended stick in a reaching task. More recently, Yang et al. [38] used graph neural networks to construct tools, but in this work, the focus was only on construction, so the constructed tool was not used. Some works also appeared where environmental objects can be moved around to achieve a navigation goal. One example is by Choi et al. [39] who used a cognitive architecture called ICARUS to construct a bridge or a stair case using planks of different length. Another example is Baker et al. [40], who demonstrated emergent tool use behavior in a multi-agent competition environment. Perhaps the most advanced form of tool construction combined with tool use is the work by Nair et al., where they used a full pipeline to analyze the tool parts, construct the tool, and test them [8, 41]. The pipeline consisted of (1) workspace segmentation, (2) shape scoring, (3) attachment scoring, and (4) tool validation. The algorithm was implemented and tested on a physical robot.

2.2. Reinforcement Learning

For sensorimotor tasks such as tool use and tool construction, reinforcement learning is the best fit. Reinforcement learning has seen rapid growth recently, with the emergence of deep reinforcement learning, utilizing deep neural networks [42]. Here we only review some reinforcement learning algorithms that are suitable to our tool construction and use task, including policy gradient, actor-critic, trust region policy optimization (TPRO), proximal policy optimization (PPO). and Deep Q Network (DQN).

(1) Policy Gradient:. There are many reinforcement learning algorithms that try to learn the value of the actions to select the actions. Policy gradient algorithms try to learn the policy directly without learning the value function [43]. Some policy gradient algorithms may still learn the value function, but the action selection is not based on the value function. As a result, policy gradient methods are ideal for the problems where the state and action space

are continuous. However, it is not straightforward to calculate the gradient because it depends on both the policy and the stationary distribution of the policy. The policy gradient theorem simplifies the calculation because it avoids the calculation of the partial derivatives of the stationary distribution [43].

(2) Actor-Critic:. To reduce the variance of the vanilla policy gradient method, Actor-Critic method learns a value function and use it to assist the policy update. There are two roles in Actor-Critic method: the actor, whose role is to maintain and update the policy to select an action; the critic, whose role is to estimate a value function with respect to the policy.

(3) TRPO and PPO:. Schulman *et al.* [44] proposed TRPO (Trust region policy optimization) and demonstrated its robust performance. TRPO updates parameters by taking large steps, and at the same time it enforces a KL divergence constraint to avoid performance collapse. PPO (Proximal Policy optimization) [45] has the same intuition as TRPO while being less complicated by using a surrogate objective. Both TRPO and PPO are on-policy algorithms and can be used in discrete space problems and continuous space problems. Because of the state-of-the-art performance, we use PPO to train part of our model, which will be elaborated in the following section.

(4) DQN:. Deep Q Network (DQN) is a value-based deep reinforcement learning algorithm that maps visual input sequence to the action value functions, using a convolutional neural network (CNN) front end, with a fully connected layer near the end [46]. DQN introduced experience replay and periodic update of the target network to achieve stability in learning. Since the model has a CNN front-end, it can deal with visual image sequences very well. The method has been used very successfully in domains such as video game playing, and related tasks that require the analysis of visual input. Several powerful variants of DQN were subsequently developed, including Double DQN (DDQN) [47]. We also use DDQN to train a different part of our model. Details will be provided in the following section.

(5) Hierarchical Reinforcement Learning. Many real-world tasks are very challenging, requiring multiple steps of decision making, while the rewards are sparse. Learning to construct and use tools is one of these challenging problems. Hierarchical reinforcement learning decomposes the task into different levels of abstraction. Here we briefly discuss recent advances in hierarchical reinforcement learning.

One of the most well-known formulation of hierarchical reinforcement is the concept of "options" introduced by Sutton *et al.* [48]. An option consists of a triplet $< I, \pi, \beta >$, where π is a policy, β is a terminal condition, and I is an input set $I \subseteq S$. An option can be seen as an action at the higher level. The implementation of options is simple and it can increase the convergence speed of training. Parr and Russell [49] proposed an approach called hierarchical abstract machines (HAM). Similar to options, HAM investigates the theory of Semi-Markov Decision Process (SMDP). However, HAM is complex to implement and thus does not have many applications. MAXQ value function decomposition is another hierarchical reinforcement learning algorithm proposed by Dietterich [50]. MAXQ decomposes the value function into two components: one is the total expected reward of the action-state pair, and the other is the total reward expected parent task. Compared to options, MAXQ directly decomposes the task into sub-tasks and the policy for the sub-tasks can be reused. However, the learned hierarchical policy is not guaranteed to be optimal. Dayan and Hinton [51] proposed a framework called Feudal Reinforcement Learning. In feudal reinforcement learning, there is a manager to assign subtasks to lower-level workers, and the workers learn to execute these subtasks. The manager observes the state of the environment at a higher level, while the workers focus on a subgoal in the original state space.

In our tool construction and use task, the agent needs to learn a sequence of steps to achieve the goal, for example, pick up the tools, construct the tool parts into a novel tool, use the novel tool to achieve the goal. Each of the steps can be seen as a subtask. Therefore, we use the feudal reinforcement learning approach to train the agent. Here we continue discuss some recent approaches

under the feudal reinforcement learning framework.

Kulkarni *et al.* [52] proposed to use two Deep Q Network (DQN) [47] for the manager (the workers are trained using PPO). Learning policies in multiple levels at the same time leads to non-stationary training. To address this issue, Nachum *et al.* [53] proposed Hierarchical Reinforcement Learning with Off-policy Correction (HIRO). Levy *et al.* [54] proposed Hierarchical Actor-Critic (HAC) and claimed that HAC is more efficient than HIRO when learning multiple levels of policies. Vezhnevets *et al.* [55] proposed a end-to-end differentiable model called FeUdal Networks (FuNs) that use dilated LSTM for the manager. Their experiments showed that FuNs improved long-term credit assignment in ATARI games and in some memory tasks.

(6) Curiosity:. In addition to hierarchical reinforcement learning, we use curiosity reward to encourage exploration. As one of the related works, Burda *et al.* [56] proposed random network distillation to add an exploration bonus, where the bonus is the error of a neural network predicting the future states of the environment. They show that random network distillation achieves outstanding performance on hard exploration Atari games. For a more general treatment on the curiosity as an internal reward (intrinsic motivation), see Oudeyer and Smith [57].

3. Approach

Similar to Nguyen *et al.* [58], we created a physical simulated environment using OpenAI gym and PyBullet, illustrated in Figure 1. The environment includes two robot arms, two tools and one target. Note that this is the first time we are using two arms, since in all our previous works, we only had one arm in the simulation. The addition of the extra arm allows us to conduct tool construction in the truest sense, by having the two arms pick up one object part each. Formally, we define the robot arm as the three-joint arm that directly belongs to the robot's body, and we define the tool as the sticks that can be picked up by the robot to achieve some goals. To simply the problem, we add

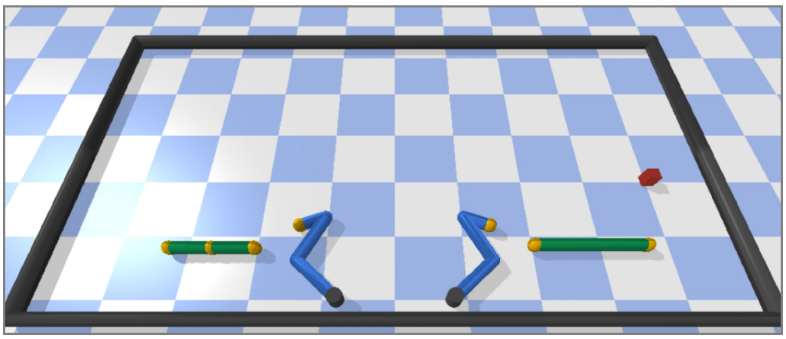

Figure 1: The tool construction environment build with PyBullet and following the OpenAI gym protocol. The environment is surrounded by a rectangle arena (black). There are two robot arms (blue), two tools (green), and one target (red). The joints marked as yellow are special joints that can be connected together upon contact. The goal for the agent is to move the object to the bottom of the environment.

special joints (yellow joints showed in fig. 1) to the robot hand and the tool. The special joints can be connected to each other upon contact (like magnets). The goal of the agent is to move the target (red cube) to the bottom of the environment.

3.1. Multimodal Input

Instead of using only raw pixels of the image as input, we use both the image and proprioceptive feedback as input, similar to Nguyen *et al.*'s method [58]. To be specific, we use the following input:

- Vision input: RGB image of the entire environment....
- Proprioception input: the position and velocities of all the agent's joints.

3.2. Hierarchical Reinforcement Learning

In order to achieve the goal, the agent needs to complete a sequence of non-trivial sub-tasks, for example, pick up the tool on the left using the left hand, pick the tool on the right using the right hand, construct a "T" shape tool by connecting one end of a tool to the middle of another tool, drag the target to the desired area. Nguyen *et al.* investigated simply using tools to drag

the target with one arm, and they used manually designed multi-step reward functions. Instead of such manually designed reward functions, reinforcement learning algorithms can be used directly with the final task goal as the reward, but the problem cannot be solved effectively because the exploration space is very large and the reward is sparse.

We propose to use hierarchical reinforcement learning with curiosity reward. Specifically, our model consists of a manager and many workers. The manager learns to plan the "macro steps" such as "pick up the two-joint tool", "pick up the three-joint tool", etc. The workers learn to execute the macro steps generated by the manager. If the macro step leads to a new state that the agent has not seen previously, then there will be some reward for the manager. Finally, there will also be a reward for the manager if the goal is reached. As for the macro step, if we define them manually, we have to enumerate all the possible steps. In addition, if we were given a novel task, we might have to re-define those macro steps. Therefore, we define the macro step in a more general way: we define the macro step as minimizing the distance between two salient objects (whatever they are), where the salient objects in our environment are the joints (yellow sphere shown in figure 1) and the target (the red cube in figure 1). Because the final goal is to move the target to the bottom of the environment, reducing the distance between the target and the bottom is another macro step. In addition, we add two macro steps that both the robot hands can release the tools they are holding.

The action space for the manager is all the pairs between all the salient objects, and the action space for the worker is the control of the two three-joint robot arm. We use DDQN [47] to train the manager and PPO [45] to train the worker. Algorithm 2 describes the details of training the manager with DDQN. In Algorithm 2, for each macro step generated by the manager, a worker will be trained with PPO (Algorithm 1) to try to accomplish the macro step. Figure 2 illustrates the overall model architecture of our approach.

There is one additional issue to be addressed for the macro step: what should be the initial environmental state for the macro steps? For the first

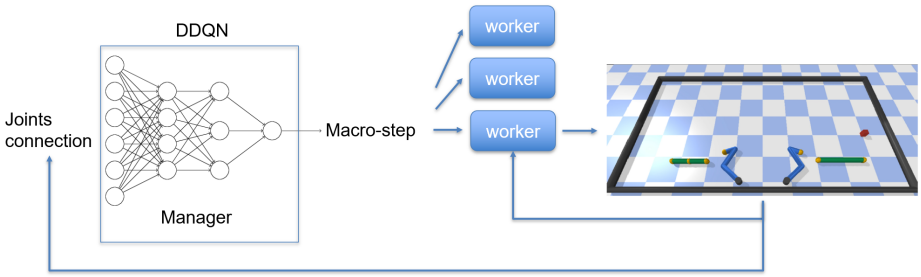

Figure 2: The overall architecture of our approach.

macro step, the initial state is the environment's initial state. But what is the initial state for the following macro step? For example, if the first macro step is "pick up the tool on the left side" and the second macro step is "connect the end of one tool to the middle of another tool", then what should be the initial state of the second macro step? To address this issue, we can save all the states of the environment (all positions of the objects, joint's velocity, joint's connection status, etc.) at the end of the previous step and use it to be the initial environmental state to train the next macro step. However, the initial state from the previous macro step is not fixed. For example, if the previous macro step is "pick up the tool on the left side", then in the initial state of the next macro step, the left hand should be holding the tool, but it could be at any position. Therefore, we run the previous macro step for k times and save all the environment's state. When training the next macro step, we randomly pick one environment's state to resume as the initial state. In our experiment, we set $k = 5$.

3.3. Neural Network Architecture

We trained the manager using the DDQN (Double DQN) algorithm. The neural network structure for the manager is three fully connected layers with a hidden layer size of 128 for each hidden layer. A Relu activation layer is followed by each layer except for the last layer. The input to the manager is the connection status of all the joint in the environment, for example, whether the left hand is holding a tool, whether two tools are connected to each other, and

Algorithm 1: Proximal Policy Optimization (PPO) algorithm [45]

Input: policy parameter θ, value function parameter ϕ

For k in 0,1,2...:

Collect trajectories D_k using policy π

Collect rewards R_t

Compute advantage estimates $\hat{A}_t$

$\theta_{k+1} =$

$\arg\max_\theta \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)))$

$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$

End

Algorithm 2: Pseudo-code of training the manager, using DDQN [47]

Input: Visual and proprioceptive inputs

$manager = DQN(observation\_space, action\_space)$

$target\_manager = DQN(observation\_space, action\_space)$

$state = env.reset()$

$for\ i\ in\ (1, number\_of\_iterations)$:

$action = manager.act(state)$

$next\_state, reward, done, info = env.step(action)$

$replay\_buffer.push(state, action, reward, next\_state, done)$

$state = next\_state$

$if\ done$:

$state = env.reset()$

$if\ len(replay\_buffer) > batch\_size$:

$loss = compute\_td\_loss(batch\_size)$

$if\ i\%100 == 0$:

$update\_target\_model(manager, target\_manager)$

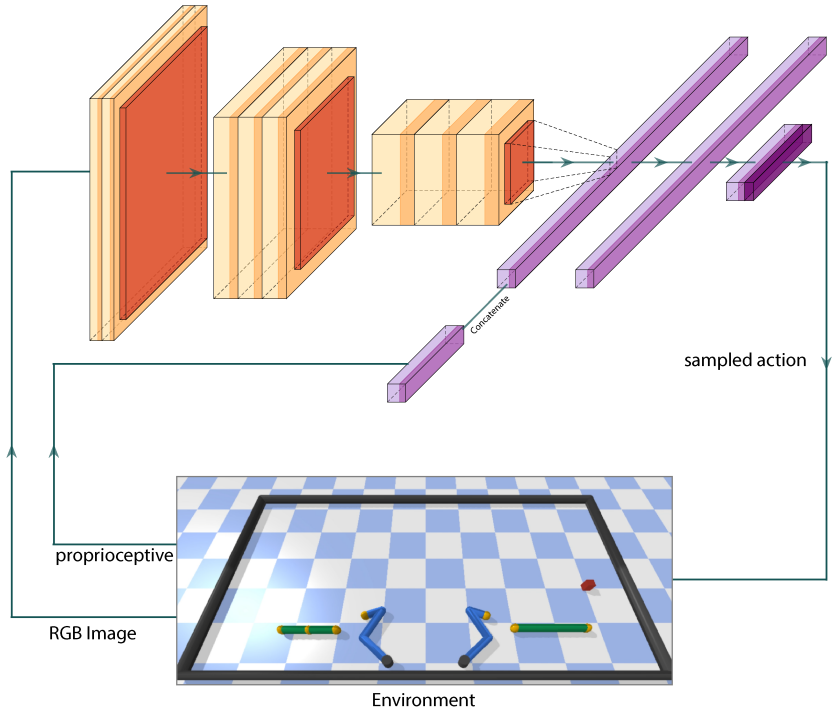

Figure 3: The overall network architecture of the workers. This figure only shows the actor network of the workers. The critic network shares the same network architecture with the actor network, except the the last output layer only has 1-dimensional output.

so on. The output of the manager is two joints to be connected, for example, the right hand of the agent and the tool on the right hand side, the middle joint of one tool and the left joint of another tool, etc. These subtasks are distributed among the workers.

We trained the workers using the Proximal Policy Optimization (PPO) algorithm. In PPO, we used the same network architecture for the actor and the critic. The overall network structure of the workers is illustrated in fig. 3. We used the RGB image as well as the proprioceptive feedback as input.

We first used three convolutional layers to process the RGB image of the environment. The first convolutional layer consists of 32 kernels with kernel size of 8 and stride of 4. The second convolutional layer consists of 64 kernels with kernel size of 4 and stride of 2. The third convolutional layer consists of

64 kernels with kernel size of 3 and stride of 1. Each of the convolutional layers is followed by a Relu activation function. Then, we flatten the feature into a 1-dimensional vector and concatenate it with the 1-dimensional proprioceptive input vector (joint angles and velocities).

The concatenated feature vector is then fed into two fully connected layers with a hidden layer size of 512 units. A Relu activation layer is followed by the first fully connected layer. Both the actor and the critic share the same network architecture and parameters. The only difference is that the output dimension for the actor is 6, representing the control signal for the robot joints, while the output dimension for the critic is 1, representing the value. Figure 3 shows the details of the actor network and the critic network.

4. Experiments and Results

We implemented our approach in PyTorch [59]. We used the DDQN algorithm to train the manager and used the Adam optimizer [60] with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-8). The replay buffer size was set to 1000 and the batch size war set to 4. For every 100 iterations the target model is synchronized with the current model.

We used the PPO algorithm to train the workers. The resolution of the image of the environment was (200, 360). We resized the image to (84, 84) and stacked 4 consecutive frames before feeding them to the neural network. The maximum steps for each episode was 4,000. We set the discount rate gamma to be 0.99, and used the Adam optimizer with default parameters.

We also evaluated the model using only RGB image sequences as input (i.e., no proprioceptive inputs). The network structure was the same as figure 3, except that the feature vector from the last layer of CNN was not concatenated with the proprioceptive input.

We ran all the training and evaluation on a machine with a NVIDIA GeForce RTX 2080 Ti GPU with 11G memory.

The manager successfully learned a sequence of sub-tasks to solve the task:

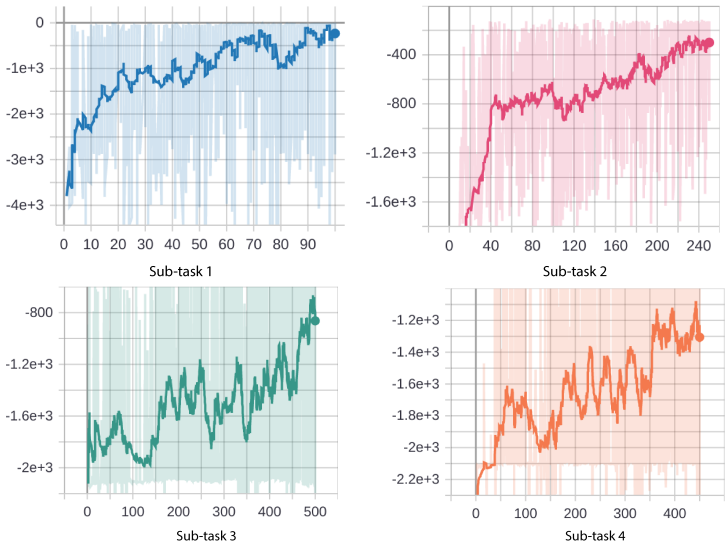

Figure 4: The smoothed reward for each sub-tasks. The Y-axis stands for the reward and the X-axis stands for epochs.

(1) pick up the tool on the right; (2) pick up the tool on the left; (3) connect the two-joint-tool with the middle joint of the three-joint-tool to construct a "T"-shaped tool; (4) use the "T"-shaped tool to drag the target to the bottom. Figure 4 shows the smoothed reward for each sub-tasks. The overall success rate of our approach is 48%, while the success rate when using only RGB image as input (i.e., no proprioceptive input) is only 20%. Table 1 describes the success rate of the agent in finishing each of the sub-tasks. As it is can be seen from the table, the success rate using vision and proprioceptive input is much higher than the success rate using vision input only. Table 2 also shows the number of steps taken by the agent to achieve each of the sub-tasks. Similarly, the agent when using vision and proprioceptive input requires fewer steps to achieve the sub-tasks, except for the last sub-task where the number of steps are similar.

Figure 5 illustrates one of the successful trials of constructing a tool and dragging the target. As can be seen from the figure, without any predefined knowledge, the agent learned to construct a novel tool to drag the object. The agent first reaches the tool on the right side with its right arm (frame 3), and

Task	vision only	vision + proprioceptive
sub-task 1	76%	99%
sub-task 2	90.8%	94.9%
sub-task 3	82.6%	92.6%
sub-task 4	35.1%	55.2%
Overall	20%	48%

Table 1: Comparison of success rate between using vision input only and using vision and proprioceptive input.

	Vision only		full input	
Task	steps	std	steps	std
Sub-task 1	53	113.6	21.3	1.5
Sub-task 2	166.6	171.3	32	17.2
Sub-task 3	245.3	197.5	145.5	138.5
Sub-task 4	234.1	129	257.6	136.4
Overall	2923.2	2090.1	1488.5	1176.2

Table 2: Comparison of number of steps needed to achieve the goal between using vision input only and using vision and proprioceptive input.

then reaches the tool on the left side with its left arm (frame 4), and then constructs a "T"-shaped tool (frame 6), and finally uses this tool to drag the object to the bottom of the screen (frame 7-8).

Figure 6 shows more results.

5. Discussion and Conclusion

In this chapter, we introduced a physical simulated environment for tool construction and tool use. We proposed an approach to use raw pixel input as well as proprioceptive input to learn to construct and use a novel tool. Unlike our previous works [58], we did not use manually designed reward functions. We used hierarchical reinforcement learning with curiosity reward to avoid manu-

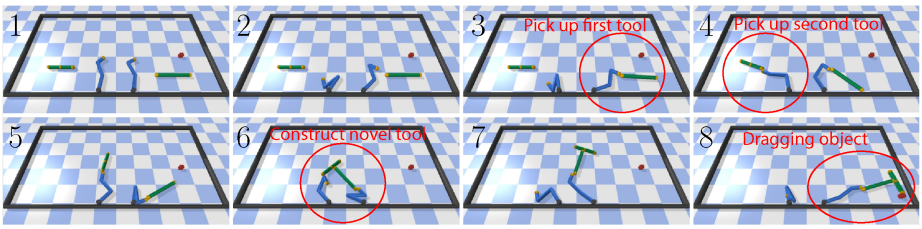

Figure 5: Controlling two arms with joints to construct a "T"-shaped tool for dragging an object (red cube) to the bottom of the screen.

ally designed rewards and accelerate learning. In our approach, the manager proposes and assigns sub-goals to workers and the workers then learn to achieve the sub-goals. We designed the sub-goals to be very general: minimizing the distance between some salient objects in the environment to encourage behaviors such as picking up tools, connecting two tools to build a new tool, and so on. Generally, such a definition of the sub-goals allows us to explore the environment at a higher level: to reach and grab some objects and to construct some objects. We believe our strategy to generate general sub-goals to encourage novel behaviors and to accelerate exploration can be used in other reinforcement learning tasks. Our experiments show that the agent can successfully learn a sequence of sub-tasks to construct a novel tool to achieve the goal. In addition, adding proprioceptive input can accelerate the training and lead to a better performance. In future work, more immediately, we intend to include more realism and complexity to the task, e.g., by introducing a gripper, requiring that tool parts to be aligned and joined (rather than snapping on), and extending the task environment to a full 3D space, as in the IKEA Furniture Assembly environment [61]. In longer term, an important goal will be to further analyze the evolved or trained neural networks (continuing our own work [12, 13]), relate them to findings in neuroscience regarding tool use (for example, Maravita and Iriki showed changes in the neural substrates due to tool use [62]), and investigate how tool use and tool construction and intelligence can co-evolve in AI ("Triadic Niche Construction" [63]).

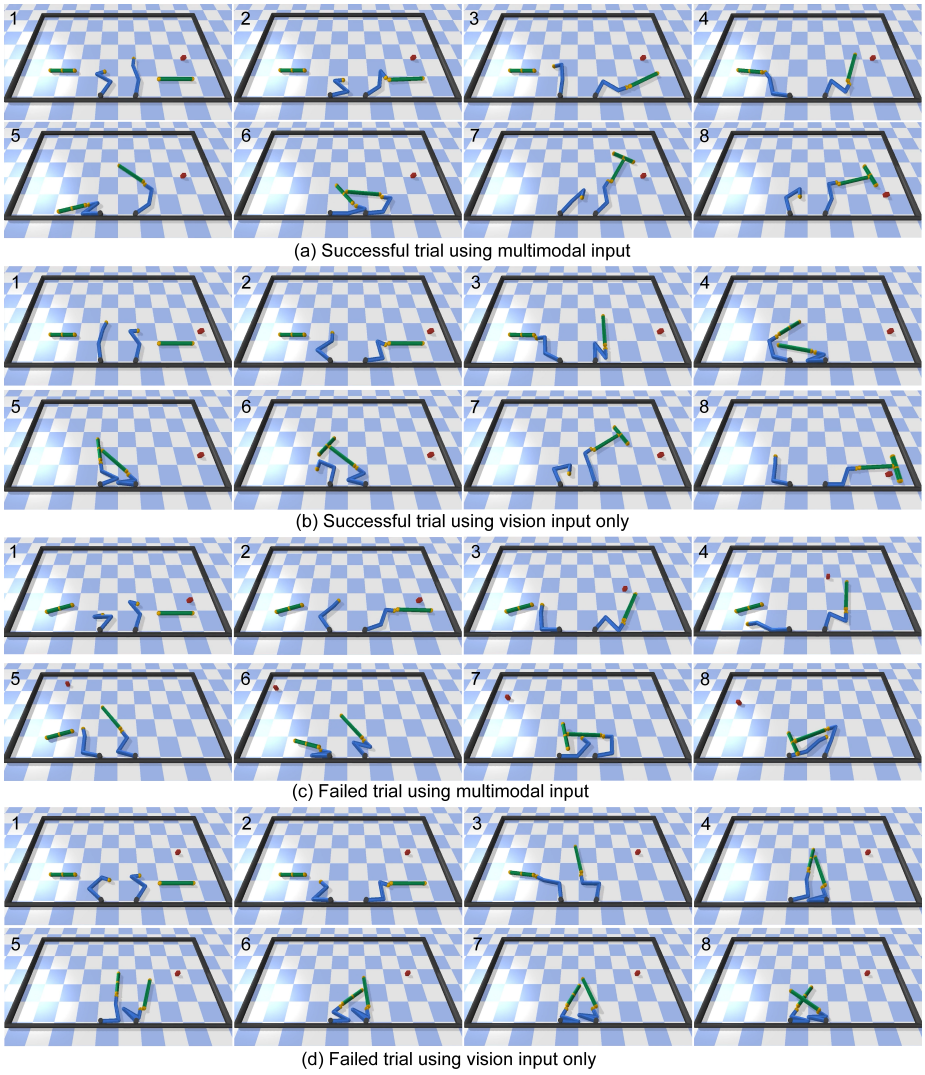

Figure 6: (a) Successful trial using multimodal input. It took 1,700 steps to complete the task for this trial. (b) Successful trial using vision input only. It took 3,400 steps to complete the task for this trial. (c) Failed trial using multimodal input. The agent accidentally pushed the target away when constructing the "T"-shaped tool, making the task unsolvable. (d) Failed trial using vision input only. The agent struggles to construct the "T"-shaped tool without proprioceptive input.

Acknowledgments

This chapter is largely based on the first author's Ph.D. dissertation [64]. Preliminary results of this work were presented at a workshop [65].

References

[1] J. Call, Three ingredients for becoming, Tool use in animals: Cognition and ecology (2013) 3–20.

[2] R. St. Amant, A. B. Wood, Tool use for autonomous agents., in: AAAI, 2005, pp. 184–189.

[3] Y. Choe, J. Yoo, Q. Li, Tool construction and use challenge: Tooling test rebooted, in: AAAI-15 Workshop on Beyond the Turing Test, 2015, 2 pages.

[4] R. W. Shumaker, K. R. Walkup, B. B. Beck, Animal tool behavior: the use and manufacture of tools by animals, JHU Press, 2011.

[5] C. Rutz, S. Sugasawa, J. E. Van der Wal, B. C. Klump, J. J. St Clair, Tool bending in new caledonian crows, Royal Society open science 3 (8) (2016) 160439.

[6] B. C. Klump, J. J. St Clair, C. Rutz, New caledonian crows keep valuable hooked tools safer than basic non-hooked tools, Elife 10 (2021) e64829.

[7] A. M. P. von Bayern, S. Danel, A. Auersperg, B. Mioduszewska, A. Kacelnik, Compound tool construction by new caledonian crows, Scientific reports 8 (1) (2018) 1–8.

[8] L. Nair, J. Balloch, S. Chernova, Tool macgyvering: Tool construction using geometric reasoning, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 5837–5843.

[9] K. R. Allen, K. A. Smith, J. B. Tenenbaum, Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning, Proceedings of the National Academy of Sciences 117 (47) (2020) 29302–29310.

[10] R. Wang, J. Lehman, J. Clune, K. O. Stanley, Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, arXiv preprint arXiv:1901.01753 (2019).

[11] K. O. Stanley, R. Miikkulainen, Competitive coevolution through evolutionary complexification, Journal of Artificial Intelligence Research 21 (2004) 63–100.

[12] Q. Li, J. Yoo, Y. Choe, Emergence of tool use in an articulated limb controlled by evolved neural circuits, in: Proceedings of the International Joint Conference on Neural Networks, 2015, DOI: 10.1109/IJCNN.2015.7280564.

[13] H. Wang, J. Yoo, Q. Li, Y. Choe, Dynamical analysis of recurrent neural circuits in articulated limb controllers for tool use, in: Proceedings of the International Joint Conference on Neural Networks, 2016, pp. 4339–4345.

[14] M. Freitag, Y. Choe, Analysis of tool use strategies in evolved neural circuits controlling an articulated limb, in: Proceedings of the International Joint Conference on Neural Networks, 2016, pp. 4331–4338.

[15] R. Reams, Y. Choe, Emergence of tool construction in an articulated limb controlled by evolved neural circuits, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 642–649.

[16] K. N. Nguyen, J. Yoo, Y. Choe, Speeding up affordance learning for tool use, using proprioceptive and kinesthetic inputs, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019, pp. 1–8.

[17] K. Nguyen, Y. Choe, Emergence of different modes of tool use in a reaching and dragging task, in: 2021 International Joint Conference on Neural Networks (IJCNN), 2021, in press.

[18] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary Computation 10 (2002) 99–127.

[19] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, J. Ba, Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation, in: Advances in neural information processing systems, 2017, pp. 5279–5288.

[20] R. Murphy, Introduction to AI robotics, MIT press, 2000.

[21] D. Lee, H. Kunori, Y. Nakamura, Association of whole body motion from tool knowledge for humanoid robots, in: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, IEEE, 2008, pp. 2867–2874.

[22] A. M. Arsenio, Learning task sequences from scratch: applications to the control of tools and toys by a humanoid robot, in: Control Applications, 2004. Proceedings of the 2004 IEEE International Conference on, Vol. 1, IEEE, 2004, pp. 400–405.

[23] R. Saegusa, G. Metta, G. Sandini, L. Natale, Developmental perception of the self and action, Neural Networks and Learning Systems, IEEE Transactions on 25 (1) (2014) 183–202.

[24] Y. Wu, Y. Demiris, Learning dynamical representations of tools for tool-use recognition, in: Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on, IEEE, 2011, pp. 2664–2669.

[25] P. Pastor, H. Hoffmann, T. Asfour, S. Schaal, Learning and generalization of motor skills by learning from demonstration, in: Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, IEEE, 2009, pp. 763–768.

[26] A. Stoytchev, Behavior-grounded representation of tool affordances, in: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, IEEE, 2005, pp. 3060–3065.

[27] D. Katz, O. Brock, Manipulating articulated objects with interactive perception, in: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, IEEE, 2008, pp. 272–277.

[28] D. Bullock, S. Grossberg, F. H. Guenther, A self-organizing neural model of motor equivalent reaching and tool use by a multijoint arm, Journal of Cognitive Neuroscience 5 (4) (1993) 408–435.

[29] S. Nishide, J. Tani, T. Takahashi, H. G. Okuno, T. Ogata, Tool–body assimilation of humanoid robot using a neurodynamical system, Autonomous Mental Development, IEEE Transactions on 4 (2) (2012) 139–149.

[30] K. Takahshi, T. Ogata, H. Tjandra, Y. Yamaguchi, Y. Suga, S. Sugano, Tool-body assimilation model using a neuro-dynamical system for acquiring representation of tool function and motion, in: Advanced Intelligent Mechatronics (AIM), 2014 IEEE/ASME International Conference on, IEEE, 2014, pp. 1255–1260.

[31] K. Takahshi, T. Ogata, H. Tjandra, Y. Yamaguchi, Y. Suga, S. Sugano, Tool-body assimilation model using a neuro-dynamical system for acquiring representation of tool function and motion, in: Advanced Intelligent Mechatronics (AIM), 2014 IEEE/ASME International Conference on, IEEE, 2014b, pp. 1255–1260.

[32] R. Jain, T. Inamura, Learning of usage of tools based on interaction between humans and robots, in: Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on, IEEE, 2014, pp. 597–602.

[33] B. Schäfer, N. Bergfeldt, M. J. Riveiro Carballa, T. Ziemke, Evolution of tool use behavior, in: Proceedings of the First IEEE Symposium on Artificial Life, Citeseer, 2007, pp. 31–38.

[34] J. R. Chung, Y. Choe, Emergence of memory in reactive agents equipped

with environmental markers, Autonomous Mental Development, IEEE Transactions on 3 (3) (2011) 257–271.

[35] K. Sims, Evolving 3d morphology and behavior by competition, Artificial life 1 (4) (1994) 353–372.

[36] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, arXiv preprint arXiv:1709.10087 (2017).

[37] L. Wang, L. Brodbeck, F. Iida, Mechanics and energetics in tool manufacture and use: a synthetic approach, Journal of The Royal Society Interface 11 (100) (2014) 20140827.

[38] C. Yang, X. Lan, H. Zhang, N. Zheng, Autonomous tool construction with gated graph neural network, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 9708–9714.

[39] D. Choi, P. Langley, S. T. To, Creating and using tools in a hybrid cognitive architecture., in: AAAI Spring Symposia, 2018.

[40] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, I. Mordatch, Emergent tool use from multi-agent autocurricula, arXiv preprint arXiv:1909.07528 (2019).

[41] L. Nair, N. S. Srikanth, Z. M. Erickson, S. Chernova, Autonomous tool construction using part shape and attachment prediction., in: Robotics: Science and Systems, 2019.

[42] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (6) (2017) 26–38.

[43] R. S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, Advances in neural information processing systems 12 (1999).

[44] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International conference on machine learning, PMLR, 2015, pp. 1889–1897.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (7540) (2015) 529–533.

[47] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 30, 2016.

[48] R. S. Sutton, D. Precup, S. Singh, Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning, Artificial intelligence 112 (1-2) (1999) 181–211.

[49] R. Parr, S. Russell, Reinforcement learning with hierarchies of machines, Advances in neural information processing systems (1998) 1043–1049.

[50] T. G. Dietterich, Hierarchical reinforcement learning with the maxq value function decomposition, Journal of artificial intelligence research 13 (2000) 227–303.

[51] P. Dayan, G. E. Hinton, Feudal reinforcement learning, in: S. Hanson, J. Cowan, C. Giles (Eds.), Advances in Neural Information Processing Systems, Vol. 5, Morgan-Kaufmann, 1993.

[52] T. D. Kulkarni, K. Narasimhan, A. Saeedi, J. Tenenbaum, Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, Advances in neural information processing systems 29 (2016) 3675–3683.

[53] O. Nachum, S. Gu, H. Lee, S. Levine, Data-efficient hierarchical reinforcement learning, arXiv preprint arXiv:1805.08296 (2018).

[54] A. Levy, G. Konidaris, R. Platt, K. Saenko, Learning multi-level hierarchies with hindsight, arXiv preprint arXiv:1712.00948 (2017).

[55] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, K. Kavukcuoglu, Feudal networks for hierarchical reinforcement learning, in: International Conference on Machine Learning, PMLR, 2017, pp. 3540–3549.

[56] Y. Burda, H. Edwards, A. Storkey, O. Klimov, Exploration by random network distillation, arXiv preprint arXiv:1810.12894 (2018).

[57] P.-Y. Oudeyer, L. B. Smith, How evolution may work through curiosity-driven developmental process, Topics in Cognitive Science 8 (2) (2016) 492–502.

[58] K. N. Nguyen, J. Yoo, Y. Choe, Speeding up affordance learning for tool use, using proprioceptive and kinesthetic inputs, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019, pp. 1–8.

[59] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8024–8035.

[60] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations (ICLR), 2015.

[61] Y. Lee, E. S. Hu, J. J. Lim, IKEA furniture assembly environment for long-horizon complex manipulation tasks, in: IEEE International Conference on Robotics and Automation (ICRA), 2021.

[62] A. Maravita, A. Iriki, Tools for the body (schema), Trends in cognitive sciences 8 (2) (2004) 79–86.

[63] A. Iriki, H. Suzuki, S. Tanaka, R. B. Vieira, Y. Yamazaki, The sapient paradox and the great journey: Insights from cognitive psychology, neurobiology, and phenomenology, Psychologia (2021).

[64] Q. Li, Exploring multimodal information in deep learing, Ph.D. thesis, Department of Computer Science and Engineering, Texas A&M University (2022).

[65] Q. Li, Y. Choe, Construction and use of tools through hierarchical deep reinforcement learning, in: 2021 IEEE/RSJ IROS Workshop on Human-like Behavior and Cognition in Robots, 2021, p. TBA.