2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Deep Reinforcement Learning for High Precision Assembly Tasks

Tadanobu Inoue[1], Giovanni De Magistris[1], Asim Munawar[1], Tsuyoshi Yokoya[2] and Ryuki Tachibana[1]

***Abstract*— The high precision assembly of mechanical parts requires precision that exceeds that of robots. Conventional part-mating methods used in the current manufacturing require numerous parameters to be tediously tuned before deployment. We show how a robot can successfully perform a peg-in-hole task with a tight clearance through training a recurrent neural network with reinforcement learning. In addition to reducing manual effort, the proposed method also shows a better fitting performance with a tighter clearance and robustness against positional and angular errors for the peg-in-hole task. The neural network learns to take the optimal action by observing the sensors of a robot to estimate the system state. The advantages of our proposed method are validated experimentally on a 7-axis articulated robot arm.**

I. INTRODUCTION

Industrial robots are increasingly being installed in various industries to handle advanced manufacturing and high precision assembly tasks. The classical programming method is to teach a robot to perform industrial assembly tasks by defining key positions and motions by using a control box called a "teach pendant." This on-line programming method is usually tedious and time consuming. Even after programming, it takes a long time to tune the parameters for deploying the robot to a new factory line due to environmental variations.

Another common method is off-line programming or simulation. This method can reduce the downtime of actual robots, but it may take longer overall than on-line programming including the time for developing the simulation and testing on the robot. It is quite hard to represent the real world including environmental variations with 100% accuracy in a simulation model. Therefore, this off-line method is not sufficient for some industrial applications such as precision machining and flexible material handling where the required precision is higher than that of robots.

In this paper, we propose a skill acquisition method where the low accuracy of conventional programming methods is compensated for by a learning method without parameter tuning. Using this method, a robot learns a high precision fitting task by using sensor feedback without explicit teaching.

For such systems, reinforcement learning (RL) algorithms can be utilized to enable a robot to learn new skills through trial and error by using a process that mimics the way humans learn [1]. The abstract level concept is shown in Fig. 1. Recent studies have shown the importance of RL for robotic tasks that use cameras and encoders [2][3][4][5], but none of these methods can be applied directly to high precision industrial applications.

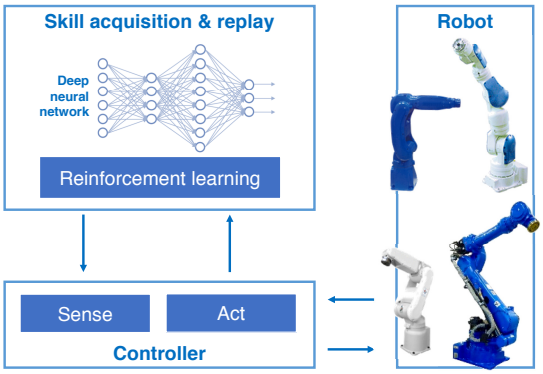

Fig. 1. Robots learn new skills by deep reinforcement learning.

To show the effectiveness of this approach, we focus on learning a tight clearance cylindrical peg-in-hole task. This is a benchmark problem for force-controlled robotic assembly. The precision required to perform this task exceeds that of robots. In addition to a tight clearance, the hole can be tilted in either direction, further adding to the difficulty. Instead of using super-precise force-torque sensors or cameras, we rely on the common force and position sensors that are ubiquitous in industrial robots. To learn the peg-in-hole task, we use a recurrent neural network, namely, long short-term memory (LSTM) trained by using RL.

The rest of the paper is organized as follows. In Section II, we explain the problem. Details on our proposed method are given in Section III. A quantitative analysis of the method on a real robot is presented in Section IV. Finally, we conclude the paper in Section V with directions for future work.

II. PROBLEM FORMULATION

A high-precision cylindrical peg-in-hole task is chosen as our target task for force-controlled robotic assembly. This task can be broadly divided into two main phases [6]:

- Search: the robot places the peg center within the clearance region of the hole center. At the beginning of this phase, the peg is located at a distance from the center of the hole in a random direction. The distance from the hole is assumed as the "positional error."
- Insertion: the robot adjusts the orientation of the peg with respect to the hole orientation and pushes the peg to the desired position. This phase is executed after the completion of the search phase subsequently.

In this paper, we study and learn these two phases separately.

[1]IBM Research - Tokyo, IBM Japan, Japan. {inouet, giovadem, asim, ryuki}@jp.ibm.com
[2]Tsukuba Research Laboratory, Yaskawa electric corporation, Japan. Tsuyoshi.Yokoya@yaskawa.co.jp

A. Search Phase

Although industrial robots have reached a good level of accuracy, it is difficult to set pegs and holes with a precision of a few tens of μm by using a position controller. Visual servoing is also impractical due to the limited resolution of cameras or internal parts that are occluded during assembly, for example, in the case of meshing gears and splines in transmission. In this paper, we use a common 6-axis force-torque sensor to learn the location of a hole with respect to the peg position.

Newman *et al.* [7] calculate the moments from sensors and interpret the current position of a peg by mapping the moments onto positions. Sharma *et al.* [6] utilize depth profiles in addition to roll and pitch data to interpret the current position of a peg. Although these approaches are shown to work in simulation, it is difficult to generalize them for real world scenarios. In a real case, it is very difficult to obtain a precise model of the physical interaction between two objects and calculate the moments caused by the contact forces and friction [8].

B. Insertion Phase

The insertion phase has been extensively researched. Gullapalli *et al.* [9] use associative RL methods for learning robot control. Majors and Richards [10] use a neural network based approach. Kim *et al.* [11] propose an insertion algorithm that can recover from tilted mode without resetting the task to the initial state. Tang *et al.* [12] propose a method for autonomously aligning the peg by measuring force and moment before the insertion phase on the basis of a three-point contact model.

Compared with these previous works, we insert a peg into a hole with a very small clearance of about 10 μm. This high precision insertion is extremely difficult, even for humans. This is due to the fact that humans cannot be so precise and the peg usually gets stuck in the very initial stage of insertion. It is also very difficult for robots to perform an insertion with a clearance tighter than their positional accuracy. Therefore, robots need to learn in order to perform this precise insertion task by using the force-torque sensor information.

III. Reinforcement Learning with Long Short-Term Memory

In this section, we explain the RL algorithm for learning the peg-in-hole task. As shown in Fig. 2, an RL agent observes the current state $\mathbf{s}$ of a system defined as:

$$\mathbf{s} = \left[F_x, F_y, F_z, M_x, M_y, M_z, \tilde{P}_x, \tilde{P}_y\right], \quad (1)$$

where F and M are the average force and moment obtained from the force-torque sensor; the subscript x, y, z denotes the axis.

The peg position P is calculated by applying forward kinematics to joint angles measured by robot encoders. During learning, we assume that the hole is not set to the precise position and that there are positionanl errors. By doing this, we add robustness against positional errors that may occur during inference. To satisfy this assumption, we

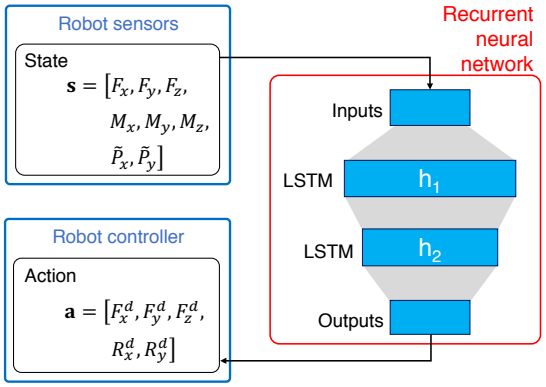

Fig. 2. Network structure using LTSM

calculate the rounded values $\tilde{P}_x$ and $\tilde{P}_y$ of the positional data P_x and P_y by using the grid shown in Fig. 3. Instead of the origin (0, 0), the center of the hole can be located at $-\mathrm{c} < x < \mathrm{c}$, $-\mathrm{c} < y < \mathrm{c}$, where c is the margin for the positional error. Therefore, when the value is $(-\mathrm{c}, \mathrm{c})$, it will be rounded to 0. Similarly when the value is $[\mathrm{c}, 2\mathrm{c})$, it will be rounded to c, and so on. This gives auxiliary information to the network to accelerate the learning convergence.

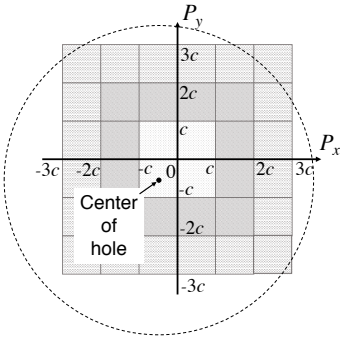

Fig. 3. Positional data rounded to grid size

The machine learning agent generates an action $\mathbf{a}$ to the robot control defined as:

$$\mathbf{a} = \left[F_x^d, F_y^d, F_z^d, R_x^d, R_y^d\right], \quad (2)$$

where F^d is the desired force and R^d is the desired peg rotation given as input to the hybrid position/force controller of the manipulator. Each component of the vector $\mathbf{a}$ is an elementary movement of the peg described in Fig. 4. An action is defined as a combination of one or more elementary movements.

The RL algorithm starts with a random exploration of the solution space to generate random actions $\mathbf{a}$. By increasing exploitation and reducing exploration over time, the RL algorithm strives to maximize the cumulative reward:

$$R_k = r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \ldots + \gamma^{n-k} r_n = r_k + \gamma R_{k+1}, \quad (3)$$

where γ is the discount factor, r is the current reward assigned to each action, and k is the step number.

With the proposed method, we compute only one reward r at the end of each episode. Then, the reward value is

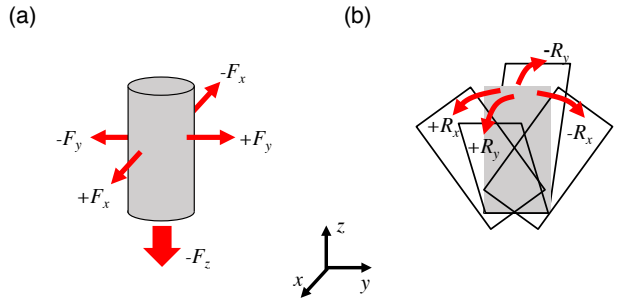

Fig. 4. Elementary movement. (a) Force movements, (b) rotation movements.

propagated with a propagation decay ℓ from the end of the episode to the beginning of the episode. We avoid using the precise distance information nearby the hole because the positional data during the learning phase can have errors. If the trial succeeds, the following positive reward r is provided to the network:

$$r = 1.0 - \frac{k}{\mathrm{k_{max}}}, \tag{4}$$

where $\mathrm{k_{max}}$ is the maximum number of steps in one episode, $k \in [0, \mathrm{k_{max}})$.

As we can see from Eq. (4), the target of the learning is to successfully perform the task in a minimum number of steps. If we cannot finish the task in $\mathrm{k_{max}}$, the distance between the starting point and the final position of the peg is used to compute the penalty. The penalty is different for the search phase and the insertion phase. For the search phase, the penalty or negative reward is defined as:

$$r = \begin{cases} 0 & (d \leq \mathrm{d_0}) \\ -\frac{d-\mathrm{d_0}}{\mathrm{D}-\mathrm{d_0}} & (d > \mathrm{d_0}) \end{cases}, \tag{5}$$

where d is the distance between the target and the peg location at the end of an episode, $\mathrm{d_0}$ is the initial positional error of the peg, and D is the distance defining the safe zone. For the insertion phase, the penalty is defined by:

$$r = -\frac{\mathrm{Z} - z}{\mathrm{Z}}, \tag{6}$$

where Z is insertion goal depth and z is the downward displacement from the initial peg position in the vertical direction.

The reward is designed to stay within the range of $-1 \leq r < 1$. The maximum reward is less than 1 because we cannot finish the task in zero steps. An episode is interrupted with reward -1 if the distance of the peg position and goal position is bigger than D in the search phase. In the insertion phase, the reward r becomes the minimum value -1 when the peg is stuck at the entry point of the hole.

To maximize the cumulative reward of Eq. (3), we use a variant RL called the "Q-learning" algorithm. At every state, the RL agent learns to select the best possible action. This is represented by the following policy $\pi(\mathbf{s})$:

$$\pi(\mathbf{s}) = \mathrm{argmax}_{\mathbf{a}} \mathrm{Q}(\mathbf{s}, \mathbf{a}). \tag{7}$$

In the simplest case, the Q-function is implemented as a table, with states as rows and actions as columns. In Q-learning, we can approximate a table update by using the Bellman equation:

$$Q(\mathbf{s}, \mathbf{a}) \leftarrow Q(\mathbf{s}, \mathbf{a}) + \alpha\Big(r + \gamma \ \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a})\Big), \tag{8}$$

where $\mathbf{s}'$ and $\mathbf{a}'$ are the next state and action, respectively.

As the state space is too big, we train a deep recurrent neural network to approximate the Q-table. The neural network parameters θ are updated by the following equation:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L_\theta, \tag{9}$$

where α is the learning rate, ∇ denotes the gradient function, and L is the loss function as follows:

$$\begin{aligned} L_\theta &= \tfrac{1}{2}\left[\mathrm{target} - \mathrm{prediction}\right]^2 \\ &= \tfrac{1}{2}\left[r + \gamma \max_{\mathbf{a}'} Q_\theta(\mathbf{s}', \mathbf{a}') - Q_\theta(\mathbf{s}, \mathbf{a})\right]^2. \end{aligned} \tag{10}$$

Using the Q-learning update equation, the parameter update equation can be written as:

$$\theta \leftarrow \theta + \alpha\Big(r + \gamma \max_{\mathbf{a}'} Q_\theta(\mathbf{s}', \mathbf{a}') - Q_\theta(\mathbf{s}, \mathbf{a})\Big)\nabla_\theta Q_\theta(\mathbf{s}, \mathbf{a}). \tag{11}$$

As shown in [13], we store the data for all previous episodes of the agent experiences to a memory pool $\mathbf{P}$ with maximum size $\mathrm{P_{replay}}$ in a first-in-first-out (FIFO) manner (Algorithm 1). Random sampling from this data provides replay events to provide diverse and decorrelated data for training.

In the case of machine learning for a real robot, it is difficult to collect the data and perform the learning offline. A robot operates in a loop, and RL keeps improving the performance of the robot over time. To efficiently collect data and learn, the proposed algorithm uses two threads: an action thread and a learning thread. Algorithm 1 shows the pseudo code of the action thread. The episode ends when we successfully finish the phase, the maximum number of allowed steps $\mathrm{k_{max}}$ is exceeded, or a safety violation occurs, i.e., going outside the safe zone D. It stores an observation in a replay memory and outputs an action on the basis of the neural network decision. Algorithm 2 shows the learning thread. It updates a neural network by learning with the replay memory.

Unlike [13], we use multiple LSTM layers to approximate the Q-function. LSTM can achieve good performance for complex tasks where part of the environment's state is hidden from the agent [14][15][16][17]. In our task, the peg is in physical contact with the environment, and the states are not clearly identified. Furthermore, when we issue an action command as shown in Eq. (2), the robot controller interprets the command and executes the action at the next cycle. Therefore, an environment affected by an actual robot action can be observed after 2 cycles from the issuing action. Experiments show that LSTM can compensate for this delay by considering the history of the sensed data.

Algorithm 1 Action thread

Initialize replay memory pool $\mathbf{P}$ to size $\mathrm{P}_{\mathrm{replay}}$
for episode = 1 to M **do**
  Copy latest network weights θ from learning thread
  Initialize the start state to sequence $\mathbf{s}_1$
  while NOT EpisodeEnd **do**
    With probability ϵ select a random action $\mathbf{a}_k$, otherwise select $\mathbf{a}_k = \mathrm{argmax}_{\mathbf{a}} Q_\theta(\mathbf{s}, \mathbf{a})$
    Execute action $\mathbf{a}_k$ by robot and observe reward r_k and next state $\mathbf{s}_{k+1}$
    Store $(\mathbf{s}_k, \mathbf{a}_k, r_k, \mathbf{s}_{k+1})$ in $\mathbf{P}$
    $k = k + 1$
  end while
end for
Send a termination signal to the learning thread

Algorithm 2 Learning thread

Initialize the learning network with random weights
repeat
  if current episode is greater than $\mathrm{E}_{\mathrm{threshold}}$ **then**
    Sample random minibatch of data $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ from $\mathbf{P}$. The minibatch size is $\mathrm{P}_{\mathrm{batch}}$
    Set target = $r + \gamma \max_{\mathbf{a}'} Q_\theta(\mathbf{s}', \mathbf{a}')$
    Set prediction = $Q_\theta(\mathbf{s}, \mathbf{a})$
    Update the learning network weight using Eq. 11.
  end if
until Receive a termination signal from the action thread

IV. EXPERIMENTS

The proposed skill acquisition method was evaluated by using a 7-axis articulated robot arm. A 6-axis force-torque sensor and a gripper were attached to the end effector of the robot as shown in Fig. 5(a). The rated load of the force-torque sensor was $200\,\mathrm{N}$ for the force and $4\,\mathrm{N\,m}$ for the moment. The resolution of the force was $0.024\,\mathrm{N}$. The gripper was designed to grasp cylindrical pegs of a diameter between 34 and 36 mm. In this paper, we suppose that the peg is already grasped and in contact with the hole plate.

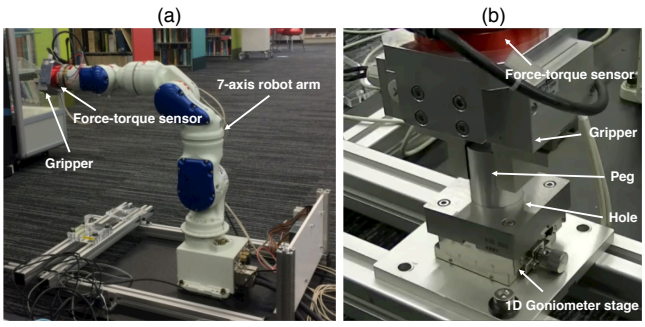

Fig. 5. (a) Robot. (b) Description of peg-in-hole components.

We prepared pegs and holes with different sizes as shown in Table I. The clearance between the peg and the hole is defined in Table I. The robot arm accuracy was only $\pm 60\,\mu\mathrm{m}$. As shown in Fig. 5(b), a 1D goniometer stage was attached to the base plate of "Hole 1" to adjust the angle of the plate with respect to the ground.

TABLE I
PEG AND HOLE DIMENSIONS

Type	Diameter	Height	Material	Clearance
Peg 1	34.990 mm	60 mm	Steel	10 µm (to Hole 1)
Peg 2	34.980 mm	60 mm	Steel	20 µm (to Hole 1)
Peg 3	34.998 mm	60 mm	Steel	6 µm (to Hole 2)
Hole 1	35.000 mm	20 mm	Steel	
Hole 2	35.004 mm	20 mm	Steel	

Fig. 6 shows the architecture of the experimental platform. The robot arm was controlled by action commands issued from an external computer (Intel Core® i7, 2.5 GHz). The computer communicated with the robot controller via the User Datagram Protocol (UDP). The sensors were sampled every 2 ms, and the external computer polled the robot controller every 40 ms to get 20 data points at one time. These 20 data points were averaged to reduce the sensor noise. The learned model was also deployed on a Raspberry Pi® 3 for execution. The machine learning module in Fig. 6 trained a LSTM network by using RL to perform an optimal action for a given system state.

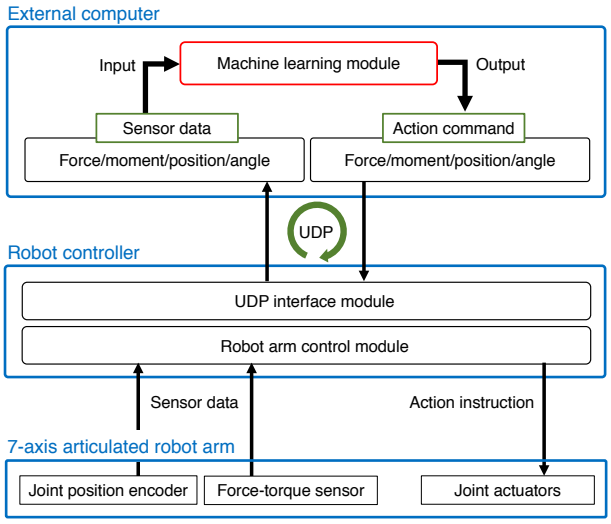

Fig. 6. Architecture of experimental platform

We treat search and insertion as two distinct skills, and we train two neural networks to learn each skill. Both networks use two LSTM layers of size $h_1 = 20$ and $h_2 = 15$ in Fig. 2. At the first step, the search phase is learned, and then the insertion phase is learned with the search skill already in place.

The maximum size of the replay memory $\mathrm{P}_{\mathrm{replay}}$ shown in Algorithm 1 was set to $20,000$ steps, and it was overwritten in a FIFO manner. The maximum number of episodes M was set to 230, and the maximum number of steps $\mathrm{k}_{\mathrm{max}}$ was set to 100 for the search phase and 300 for the insertion phase. The learning thread shown in Algorithm 2 started learning after

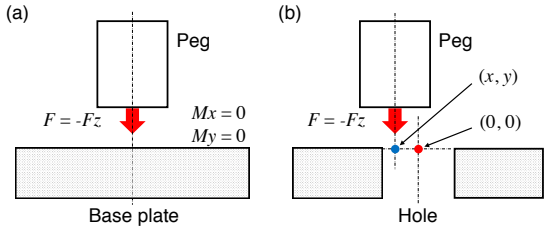

Fig. 7. Preliminary experiments for moment analysis. (a) Aligning peg to get zero moment values. (b) Stamping peg nearby hole.

$\mathrm{E}_{\mathrm{threshold}} = 10$ episodes. The batch size was $\mathrm{P}_{\mathrm{batch}} = 64$ to select random experiences from $\mathbf{P}$.

The initial exploration rate ϵ for the network was set to 1.0, i.e., the actions were selected randomly at the start of learning. The exploration was reduced by 0.005 after each episode until it reached 0.1. This allowed for a gradual transition from exploration to exploitation of the trained network. AdaDelta [18] was used for the optimizer with $\rho = 0.95$ and $\epsilon = 1\mathrm{e}{-6}$. The discount factor γ was set to 0.5. The propagation decay ℓ was set to 0.95.

A. Preliminary Experiments

Preliminary experiments and analysis on actual robot moments were performed to compute the optimal vertical force F_z^d. We first calibrated the 6-axis force-torque sensor. In particular, we adjusted the peg orientation (R_x, R_y) to ensure that both M_x and M_y were 0 for a vertical downward force $F_z = 20\,\mathrm{N}$ as shown in Fig. 7(a). After calibration, we analyzed the moments for three different downward forces F_z^d at three different peg locations (x, y) as shown in Fig. 7(b).

Fig. 8 shows the moment values for nine different configurations of peg position and force. Figs. 8(a) and 8(d) show that we cannot get a detectable moment by pushing down with a force of 10 N. In contrast, it is clear that a downward force of both 20 N and 30 N can be used for estimating the hole direction on the basis of the moment values. As expected, in the case of $F_z^d = -20\,\mathrm{N}$ in Figs. 8(b) and 8(e), M_y was bigger when the peg was closer to the hole. It is better to use a weaker force to reduce wear and tear of the apparatus, especially for relatively fragile material, e.g., aluminum or plastic. As a result, we decided to use a downward force of 20 N for all subsequent experiments in the search phase.

B. Search Phase

Due to the accuracy of robot sensors, there is an inherent error of 60 µm in the initial position of the peg. In addition, the hole can be set by humans manually in a factory, and there can be large positional errors in the initial position of the hole. Therefore, we executed learning the search phase from a different position at the beginning of every episode in order to make the system robust to positional errors. We added a constant positional error from the center of the hole in 1 of 16 directions randomly selected. Instead of directly starting from a large positional error, the learning

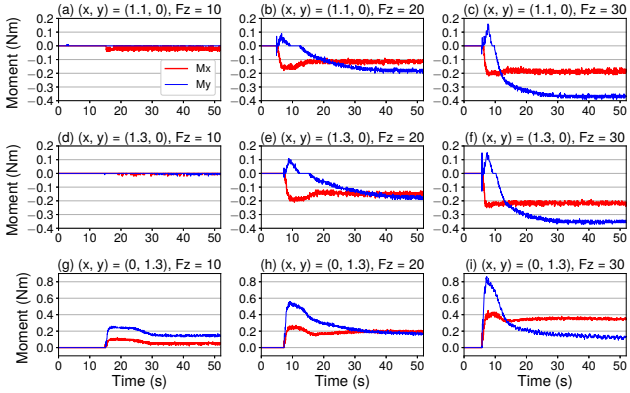

Fig. 8. Moment values in preliminary experiments, M_x in red and M_y in blue. (a)(b)(c) $(x, y) = (1.1, 0)\mathrm{mm}$ and (a) $F_z = 10\,\mathrm{N}$, (b) $F_z = 20\,\mathrm{N}$, (c) $F_z = 30\,\mathrm{N}$. (d)(e)(f) $(x, y) = (1.3, 0)\mathrm{mm}$ and (d) $F_z = 10\,\mathrm{N}$, (e) $F_z = 20\,\mathrm{N}$, (f) $F_z = 30\,\mathrm{N}$. (g)(h)(i) $(x, y) = (0, 1.3)\mathrm{mm}$ and (g) $F_z = 10\,\mathrm{N}$, (h) $F_z = 20\,\mathrm{N}$, (i) $F_z = 30\,\mathrm{N}$.

was done in stages for the search phase. We started with a small initial positional error of $\mathrm{d}_0 = 1\,\mathrm{mm}$ for the peg from the hole and learned the network parameters. Using this as prior knowledge, we increased the initial positional error to $\mathrm{d}_0 = 3\,\mathrm{mm}$. Instead of starting from an exploration rate of 1.0, we set the initial exploration rate to 0.5 for the subsequent learning stage.

The state input $\mathbf{s}$ to the search network is a 7-dimensional vector of Eq. (1). These data are rescaled to a similar level before being fed to the network. The size of the grid in Fig. 3 was set to $c = 3\,\mathrm{mm}$ for $\mathrm{d}_0 = 1\,\mathrm{mm}$ and $c = 5\,\mathrm{mm}$ for $\mathrm{d}_0 = 3\,\mathrm{mm}$. The neural network selected one of the following four actions defined by using Eq. (2):

1) $[+F_x^d, 0, -F_z^d, 0, 0]$
2) $[-F_x^d, 0, -F_z^d, 0, 0]$
3) $[0, +F_y^d, -F_z^d, 0, 0]$
4) $[0, -F_y^d, -F_z^d, 0, 0]$

with $F_x^d = 20\,\mathrm{N}$, $F_y^d = 20\,\mathrm{N}$, and $F_z^d = 20\,\mathrm{N}$. Since the peg stayed in contact with the hole plate at a constant force $-F_z^d$, it could enter the hole during the motion. Compared with step-wise movements, continuous movements made by force control can prevent static friction.

The peg position P_z was used to detect when a search was successful. If P_z became smaller than $\Delta \mathrm{z_s} = 0.5\,\mathrm{mm}$ compared with the starting point, we say that the peg was inside the hole. We set 10 mm for the maximum safe distance D in Eq. (5).

Fig. 9 shows the learning progress in the case of a clearance of 10 µm, a tilt angle of 0°, and an initial offset of 1 mm. Fig. 9(a) shows the learning convergence, and Fig. 9(b) illustrates that the number of steps to successfully accomplish the search phase was reduced significantly. The training time for the search phase was approximately 50 minutes.

C. Insertion Phase

After training the searching network, we trained a separate but similar network for insertion. Successful searching is

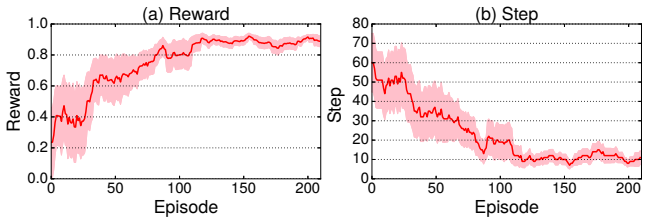

Fig. 9. Performance of proposed method during learning search phase with clearance of 10 µm, tilt angle of 0°, and initial positional error of 1 mm. (a) Reward, (b) step. Means and 90% confidence bounds in moving window of 20 episodes.

a pre-requisite for the insertion phase. We executed the searching phase to get the starting point for learning the insertion phase. On the basis of the 7-dimensional vector of Eq. (1), we define the following state input vector of this network:

$$\mathbf{s} = [0, 0, F_z, M_x, M_y, 0, 0]\,, \tag{12}$$

where M_x and M_y sense the peg orientation, while F_z indicates if the peg is stuck or not.

To accomplish the insertion phase, the system chooses from the following five actions of Eq. (2).

1) $[0, 0, -F_z^d, 0, 0]$
2) $[0, 0, -F_z^d, +R_x^d, 0]$
3) $[0, 0, -F_z^d, -R_x^d, 0]$
4) $[0, 0, -F_z^d, 0, +R_y^d]$
5) $[0, 0, -F_z^d, 0, -R_y^d]$

The vertical peg position P_z is used for goal detection. If the difference between the starting position and the final position of the peg P_z becomes larger than Z, we can judge that the insertion is completed successfully. We used 19 mm for the stroke threshold Z in Eq. (6). The reward for a successful episode is similar to the one used in the search phase as shown in Eq. (4). The training time for the insertion phase was approximately 80 minutes.

D. Results

To show the robustness of the proposed method, we performed experiments with different combinations of a peg and a hole for different clearances. We also performed learning and execution with a tilted hole plate by using a 1D goniometer stage under the plate. In this paper, the tilt angle was fixed during one online learning. The results are shown in the video (see `https://youtu.be/b2pC78rBGH4`).

We executed the peg-in-hole task 100 times after learning to evaluate the time performances of the learning method. We achieved a 100% success rate with a different execution time for each configuration. Fig. 10 shows histograms of the execution time for the following two cases regarding search, insertion, and total time.

- Case A: an initial positional error of 3 mm in 1 of 16 random directions, a clearance of 6 µm, and a tilt angle of 0°.
- Case B: an initial positional error of 1 mm in 1 of 16 random directions, a clearance of 20 µm, and a tilt angle of 1.6°.

Fig. 10(a) shows the distribution of the execution time spread over a wider area and is shifted further right than Fig. 10(d). When the tilt angle was larger, the execution time for the insertion increased as the angle of the peg needed to be aligned with the hole.

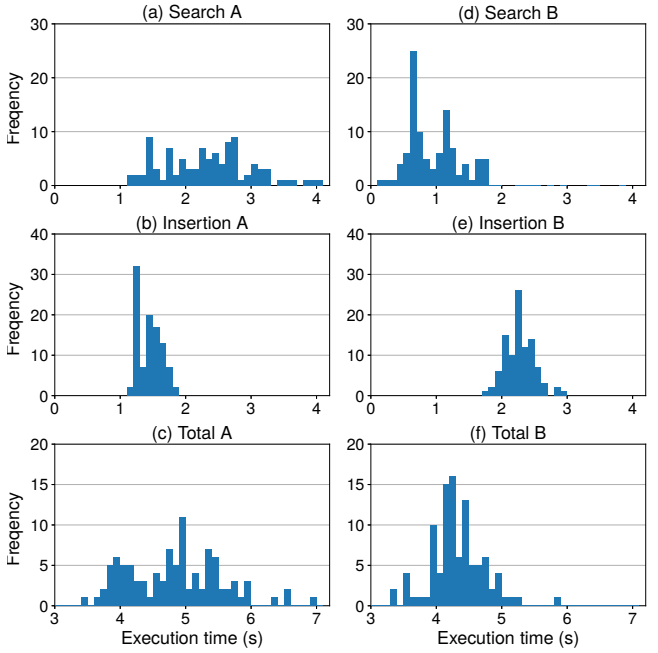

Fig. 10. Histograms of execution time. Case A with clearance of 6 µm, tilt angle of 0°, and initial positional error of 3 mm. (a) Search time, (b) insertion time, (c) total time. Case B with clearance of 20 µm, tilt angle of 1.6°, and initial positional error of 1 mm. (d) Search time, (e) insertion time, (f) total time.

TABLE II

AVERAGE EXECUTION TIME FOR PEG-IN-HOLE TASK.

(1) CONVENTIONAL APPROACH USING FIXED SEARCH PATTERNS [19].

(2) OUR PROPOSED METHOD.

Approach	(1)	(2)	(2)	(2)	(2)	(2)
Clearance [µm]	≥ 10	10	**6**	10	**6**	20
Angular error [°]	≤ 1.0	0	0	0	0	**1.6**
Positional error [mm]	≤ 1.0	1.0	1.0	**3.0**	**3.0**	1.0
Search time (s)	–	0.97	1.47	2.26	2.43	0.95
Insertion time (s)	–	1.40	1.45	1.33	1.43	2.31
Total time (s)	∼5.0	3.47	3.98	4.68	4.93	4.36

Table II summarizes the average execution time in the 100 trials for the 6 cases. For comparison, our results are compared with the specifications on the product catalog of the conventional approach [19]. The conventional approach also uses force sensing control, and a predefined fixed search pattern is used during the search phase in a fitting task. The minimum clearance with the conventional approach is 10 µm, and the maximum positional and angular errors allowed by the conventional approach are 1 mm and 1°, respectively.

Our proposed method achieves a tighter fitting task with a clearanance of 6 µm while keeping similar total execution time. The results show that more robust fitting skills against

positional and angular errors can be acquired by the proposed learning method.

V. CONCLUSION AND FUTURE WORK

There are industrial fitting operations that require very high precision. Classical robot programming techniques require a long setup time to tune parameters due to environmental variations. In this paper, we propose an easy to deploy teach-less method for precise peg-in-hole tasks and validate its effectiveness by using a 7-axis articulated robot arm. The results showed the better performance for a tigher clearance and robustness against positional and angular errors for a fitting task.

In this paper, the high precision fitting task was learned for each configuration by using online learning. As the next step, we will generalize the model for various configurations by offline learning with an experience pool gathered for multiple tilt angles and clearance configurations. As future work, we will gather trial information from multiple robots in various configurations and upload them to a cloud server. A more general model will be learned on the cloud by using this data pool in batches. We would like to generalize the model so that it can handle different materials, robot manipulators, and also different shapes. Then, skill as a service will be delivered to robots in new factory lines with a shortened setup time.

The proposed method uses a discrete number of actions to perform the peg-in-hole task. As an obvious next step, we will analyze the difference between this method and continuous space learning techniques such as A3C [20] and DDPG [21].

ACKNOWLEDGMENTS

We are very grateful to Masaru Adachi at the Tsukuba Research Laboratory of the Yaskawa Electric Corporation, Japan for his helpful support toward this work.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," International Journal of Robotic Research, vol.32, no.11, pp.1238-1274, 2013.

[2] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," International Symposium on Experimental Robotics (ISER), 2016.

[3] L. Pinto and A. Gupta, "Supersizing Self-supervision: Learning to Grasp from 50k Tries and 700 Robot Hours," IEEE International Conference on Robotics and Automation (ICRA), 2016.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," IEEE International Conference on Robotics and Automation (ICRA), 2017.

[5] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient Deep Reinforcement Learning for Dexterous Manipulation," arXiv:1704.03073, 2017.

[6] K. Sharma, V. Shirwalkar, and P. K. Pal, "Intelligent and Environment-independent Peg-in-hole Search Strategies," International Conference on Control, Automation, Robotics and Embedded Systems (CARE), 2013.

[7] W. S. Newman, Y. Zhao, and Y. H. Pao, "Interpretation of Force and Moment Signals for Compliant Peg-in-Hole Assembly," IEEE International Conference on Robotics and Automation, 2001.

[8] C. Bouchard, M. Nesme, M. Tournier, B. Wang, F. Faure, and P. G. Kry, "6D Frictional Contact for Rigid Bodies," Proceedings of Graphics Interface, 2015.

[9] V. Gullapalli, R. A. Grupen, and A. G. Barto, "Learning Reactive Admittance Control," IEEE International Conference on Robotics and Automation, 1992.

[10] M. D. Majors, and R. J. Richards, "A Neural Network Based Flexible Assembly Controller," Fourth International Conference on Artificial Neural Networks, 1995.

[11] I. W. Kim, D. J. Lim, and K. I. Kim, "Active Peg-in-hole of Chamferless Parts using Force/Moment Sensor," IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999.

[12] T. Tang, H. C. Lin, Y. Zhao, W. Chen, and M. Tomizuka, "Autonomous Alignment of Peg and Hole by Force/Torque Measurement for Robotic Assembly," IEEE International Conference on Automation Science and Engineering (CASE), 2016.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," NIPS Deep Learning Workshop, 2013.

[14] B. Bakker, "Reinforcement Learning with Long Short-Term Memory," 14th International Conference on Neural Information Processing Systems (NIPS), 2001.

[15] M. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15), 2015.

[16] G. De Magistris, A. Munawar, and P. Vinayavekhin, "Teaching a Robot Pick and Place Task using Recurrent Neural Network," Vision Engineering Workshop (ViEW), 2016.

[17] A. Munawar, P. Vinayavekhin, and G. De Magistris, "Spatio-Temporal Anomaly Detection for Industrial Robots through Prediction in Unsupervised Feature Space," IEEE Winter Conference on Applications of Computer Vision (WACV), pp.1017-1025, 2017.

[18] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," arXiv:1212.5701, 2012.

[19] Yaskawa Europe GmbH, Motofit, "https://www.yaskawa.eu.com/index.php?eID=dumpFile&t=f&f=11644&token=241c4282605991b04d445f52399c614c3192d811."

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," International Conference on Machine Learning, 2016.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971, 2015.