Robot gaining accurate pouring skills through self-supervised learning and generalization

Yongqiang Huang, Juan Wilches, Yu Sun *

Department of Computer Sci & Eng, University of South Florida, 4202 E. Fowler Ave, Tampa, FL, 33620, United States

ARTICLE INFO

ABSTRACT

Pouring is one of the most commonly executed tasks in humans' daily lives, whose accuracy is affected by multiple factors, including the type of material to be poured and the geometry of the source and receiving containers. In this work, we propose a self-supervised learning approach that learns the pouring dynamics, pouring motion, and outcomes from unsupervised demonstrations for accurate pouring. The learned pouring model is then generalized by self-supervised practicing to different conditions such as using unaccustomed pouring cups. We have evaluated the proposed approach first with one container from the training set and four new but similar containers. The proposed approach achieved better pouring accuracy than a regular human with a similar pouring speed for all five cups. Both the accuracy and pouring speed outperform state-of-the-art works. We have also evaluated the proposed self-supervised generalization approach using unaccustomed containers that are far different from the ones in the training set. The self-supervised generalization reduces the pouring error of the unaccustomed containers to the desired accuracy level.

1. Introduction

Pouring is one of the most commonly executed tasks in humans' daily lives, especially for food preparation. In cooking scenarios, it is the most frequently executed motion [1]. Pouring is a challenging task given the variety of containers and materials we can find in a kitchen. Humans excel at pouring liquids and solid materials, a skill that a cooking robot needs to master. This skill becomes particularly tricky when there is the need to execute it with precision and speed and for different setups and conditions. Accurate pouring is not a trivial task, and it is affected by many factors, including the property of the material, the geometry of the source and receiving containers, the manipulation of the source container, to name a few. It is a problem that cannot be solved using traditional control policies for two reasons:

1. Lack of precise dynamics models: modeling fluid or granular motion precisely is either impossible or unfeasible because there are many unobservable parameters and those parameters vary with many factors such as the material and the shape of the pouring device.
2. Un-reversible feature of the task: the poured material cannot come back to the pouring device once it is poured out.

Therefore, it is not possible to have an overshoot in the system's response.

The two difficulties go hand-in-hand. The un-reversible feature of pouring calls for an approach that can predict. Fig. 1 shows an example of velocity and volume sequences collected from a person pouring water. It can be seen that after the backward rotation starts, the water still comes out of the source container and the volume in the receiving container keeps increasing for a while. Therefore, the approach needs to predict when to start the backward rotation to reach the goal. However, prediction requires a precise model. In this paper, we propose a self-supervised learning approach that learns from demonstrations that are either unsupervised or performed by unskilled demonstrators. The approach self-supervises the learning process by taking in all demonstrations without checking their performance or labeling them as successful or unsuccessful. Instead, the self-supervised approach uses the real outcomes of the demonstrations as the desired goals. It is drastically different from traditional learning from demonstration approaches (LfD) [2] that learn optimal motion trajectories from skilled demonstrators.

We designed a data collection system that collected 284 human pouring demonstrations. This new data collection approach extends the Daily Interactive Manipulation (DIM) dataset [3]. To learn water pouring dynamics, pouring motion, and outcomes, we have developed a peephole long short-term memory (LSTM) learning structure that used the previous step's outcome as the

* Corresponding author.
E-mail addresses: yongqiang@usf.edu (Y. Huang), jwilches@usf.edu (J. Wilches), yusun@usf.edu (Y. Sun).
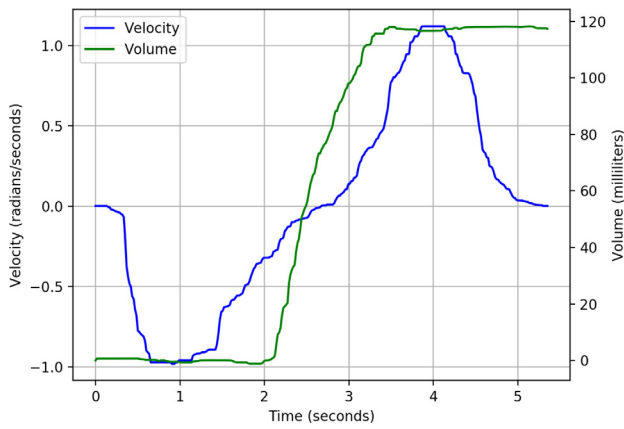
Fig. 1. An example of human pouring water. It can be seen that when the rotation is going backward, the volume is still increasing. Velocity: velocity of the source container measured counter clock-wise, negative radians/seconds means that the container is going forward and positive means backward. Volume: volume of water in the receiving container.

current input. The cell unit in the peephole LSTM learns, memorizes, and updates the liquid or granular material's movement dynamics over time. It allows the peephole LSTM model to learn the relationship between the manipulation motion and the disparity between the current outcome and the desired outcome based on the liquid or granular material's movement dynamics.

We have evaluated the proposed approach with five containers: one from the training set and four new containers that are not significantly different from the training set in terms of size. We have observed the mean volume error of those five containers to be as small as 4.12 milliliters (mL) and not higher than 12.35 mL, a result that is better than the pouring accuracy of a regular person. The pouring speeds of the proposed approach are on par with a regular human pouring speed that is about 5 to 10 times faster than state-of-the-art approaches [4,5]. The learned model exhibits either much higher pouring error or much greater pouring standard deviation when evaluated on unaccustomed containers that are far different from the ones in the training set. To generalize the learned model to unaccustomed containers, we propose a self-supervised practicing approach. It uses the learned model to practice with the unaccustomed containers, collects the motions and outcomes from the practices, and uses the real outcomes as the desired outcomes to fine-tune the model. We refer to this approach as *generalization by self-supervised practicing (GSSP)*, with which we achieved a reduction in mean volume error from values of more than 50 mL to values lower than state-of-the-art works.

Our contributions in this work include

1. Data collection system and pouring motion dataset. We designed a data collection system that captures the motion signals of human pouring employing a motion tracker and a force sensor (for volume measurement). We have collected 284 pouring motions for nine different sizes of source containers.
2. Self-supervised learning from demonstrations and outcomes. We present a motion model for accurate pouring learned from human demonstrations. The pouring target in the input of the model during training is set to be the **actual poured volume** instead of the desired target. It allows the learning to be self-supervised since the approach does not have to specify or know the desired outcomes.
3. The learned pouring skill could achieve human-like pouring accuracy and speed. The proposed approach models the

complicated spatial–temporal patterns of human pouring, and as a result, pours smoothly and as fast as humans. In our dataset collected from human demonstrations, the pouring time ranges from 3.2 to 8.7 s, and the duration executed by our model ranges from 2.8 to 7.6 s. It pours faster than related methods [4,5] which report 25 s and 20–45 s per pour, respectively. It achieves lower pouring error than existing methods that also use a single modality to monitor the poured volume [4,6–8].
4. Generalizing pouring skills by self-supervised practicing. We present the Generalization by self-supervised Practicing (GSSP) approach, which fine-tunes the model using the actual pouring outcomes of a robot. It allows the robot to pour accurately, using unaccustomed containers and materials.

1.1. Related work on LfD

One popular LfD approach is Gaussian mixture regression (GMR) [9]. The approach first learns the spatial–temporal relation of a motion. Then, it produces a novel sequence of the motion by producing the position or state of the motion corresponding to each time step. Functional Principal Component Analysis (fPCA) is another approach for spatial–temporal motion learning. The traditional PCA can be applied on both the temporal and spatial axes of a motion to find how the motion varies at different time steps [10,11]. Functional PCA (fPCA) extends PCA by representing a motion in a continuous-time format instead of using a collection of points [12–15]. The spatial–temporal GMR and fPCA in nature consider the motion as a whole rather than a dynamical system, which makes it inconvenient to receive timely feedback while executing the motion. In comparison, GMR can be configured to learn the relationship between states and actions and thus behave as a dynamical system [16].

Alternatively, one can use movement primitives (MP). The first MPs, the dynamic movement primitives (DMP), consist of three components: (1) a strictly damped string model that guarantees the convergence of the motion state to a goal state, (2) a forcing function that contains the shape that the motion is expected to go through, and (3) a canonical system that modulates the temporal profile of the motion [17]. Its variants include but are not limited to interactive primitives (IP), which enable the interaction between two agents [18] and probabilistic movement primitives (ProMP), which enable more flexibility on the force function [19]. The general GMR, MP, and fPCA involve the usage of a temporal alignment algorithm of the motion data, such as dynamic time warping (DTW) [20], which may damage certain spatial–temporal patterns in the data in unclear ways. In comparison, a recurrent neural network (RNN) does not require aligning the motion data in time. It is designed to process time sequences and is capable of representing dynamical systems [21,22]. RNN has been successfully applied for text generation as well as motion generation [23–25].

1.2. Related work on pouring

In [26], the authors propose trajectory planning algorithms for liquid body transfer that uses fluid simulation, [27] learns to predict the state of the fluid using neural networks. However, the fluid model of the liquid is in general difficult to obtain, for which reason [28] proposes adding goal learning to shape learning with MP for liquid transfer. [29] considers the amount of the liquid in the source container while pouring and proposes a liquid transfer algorithm based on a parametric hidden Markov model. The difficulty of generating pouring motion increases when pouring accuracy is essential. The demand for accurate pouring is

observed in casting factories where molten metal is poured into molds. [30] proposes predicting the residual pouring amount of the liquid to increase the accuracy of pouring. [31] introduces predictive sequence control, which suppresses the increase of error when the pouring amount increases.

Humans also control the amount when they pour and for which they combine pouring with shaking and tapping [32]. Efforts have been made to estimate the volume or height of the poured amount in the receiving container and to use the estimate as real-time feedback to a simple PID controller. [4] uses a deep neural network to estimate the volume of liquid in a cup using visual data and uses PID controllers to control the rotation of a robot arm. In 30 pours, the algorithm achieves an average error of 38 mL with 25 s per pour. [6] uses an RGB-D point cloud of the receiving cup to determine the liquid height and a PID controller to control the rotating angle of the source cup. They achieve a mean error of 23.9 mL, 13.2 mL, and 30.5 mL for three different receiving cups. In both algorithms, the PID controller stops and rotates the source container back to the original angle when the estimated volume/height reaches the target. However, the before-mentioned technique might lead to over-pouring since there is still liquid coming out from the source container when it starts its backward rotation as shown in Fig. 1. Moreover, the generalization of those vision-based approaches is limited when there is variation in the color of the receiving container, the lighting conditions, the background, or the type of pouring material.

Instead of using PID, [7] learns a policy using reinforcement learning simulation and transfers the policy to actual robots. The policy performs pouring to the same target heights for which it was trained. It reaches an average error of 19.96 mL over 40 pouring trials. However, the authors also use a vision-based system to detect the height of the liquid in the receiving container, leading to the same limitations already discussed. In [33] the authors rely on an audio spectrogram to determine the volume poured by the robot. The mean volume errors reported for different receiving containers ranged from 6.42 ml to 13.79 ml, such small errors are achieved by the usage of a spout at the opening of the pouring containers, which reduces the speed of pouring. [5,34] derives analytical pouring models for the source containers with known geometry and extends the model to source containers with similar geometry. The most up-to-date work [5] uses both vision and weight during pouring and achieves a pouring error of less than 5 mL. However, the proposed system pouring time ranges from 20 to 45 s. Humans can also achieve small pouring errors if requested to pour slowly. In our dataset, humans took 3.2 to 8.7 s to pour water. The authors in [8] apply model predictive control (MPC) based on a recurrent neural network for estimating the poured volume. The approach achieves average errors of 14.25 mL, 18.25 mL, and 26.13 mL for three unseen source containers. Another work [25] presented a Long-Short-Term Memory (LSTM) model that was trained using demonstration data. However, the learned model was only evaluated in simulation.

In summary, the accurate pouring approach presented in this paper has a significant improvement over our previous works. The proposed peephole LSTM approach drastically outperforms the MPC algorithm presented in [8]. The proposed approach was applied to a real robot, whereas in [25] the pouring motion velocities were generated in simulation. In [35], we presented limited preliminary results on pouring as an abstract report. A generalization in practice (GiP) approach was proposed in [36] with limited experiments on pouring water. This paper presents a comprehensive description of the peephole LSTM pouring motion generation approach and the generalization by self-supervised practicing (GSSP), both thoroughly evaluated with numerous experiments in real-world scenarios with a real robotic system.
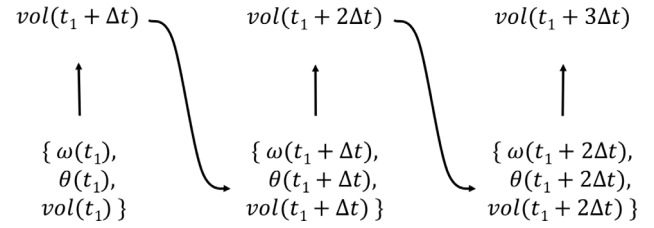
$$vol(t_1 + \Delta t) \qquad vol(t_1 + 2\Delta t) \qquad vol(t_1 + 3\Delta t)$$

$$\{ \omega(t_1), \qquad \{ \omega(t_1 + \Delta t), \qquad \{ \omega(t_1 + 2\Delta t),$$
$$\theta(t_1), \qquad \theta(t_1 + \Delta t), \qquad \theta(t_1 + 2\Delta t),$$
$$vol(t_1) \} \qquad vol(t_1 + \Delta t) \} \qquad vol(t_1 + 2\Delta t) \}$$

Fig. 2. The sequential pouring process with the input being the current angular velocity and the current poured volume and the output being the poured volume for the next time step.

2. Problem description & approach

In this work, we do not consider the transfer of the source container, which is essentially pick-and-place, and only focus on controlling the flow of the liquid by manipulating the source container. We consider the receiving container to be large enough to prevent spilling. The main movement of the source container is its rotation, which resides mostly on a 2-dimensional plane. It enables the motion to be simplified to its rotation. The anchor of the rotation is fixed to be approximately at the middle point of the height of each container. This simplification is also applied in [4–6,34], which makes our assumptions reasonable. Volume can be perceived visually and is intuitive for measuring liquid. Therefore, in this work, we use volume to represent the amount of liquid.

We describe the pouring process as a result of rotating the source container. Initially, a certain volume of liquid exists in the source container. If the source container is full, then the liquid flows out as the source container starts rotating. If the source container is not full, then there will be a delay of the liquid flowing out after the source container has started rotating. The liquid flows into and stays in the receiving container, and therefore the poured volume only increases and never decreases. When the source container stops rotating, the liquid may either instantly stop flowing out or keep flowing for a short time until the surface of the liquid inside the source container is level. The pouring process is sequential, and the poured volume is determined by the trajectory of the rotation velocities of the source container.

We model the pouring process as a discrete-time series:

1: **for** i in $(1, 2, \dots)$ **do**
2: $\quad t = t_1 + (i - 1)\Delta t$
3: $\quad \theta(t + \Delta t) = \theta(t) + \omega(t)\Delta t$
4: $\quad vol(t + \Delta t) = F(\omega(t), \theta(t), vol(t))$
5: **end for**

where t_1 is the initial time instant, Δt is the time interval, $\theta(t)$ and $\omega(t)$ are the rotation angle and angular velocity of the source container, respectively, $vol(t)$ is the poured volume, $F(\cdot)$ denotes the pouring system. We also illustrate the process in Fig. 2. We do not impose a strict restriction on the initial angle $\theta(t_1)$ but assume that it is close to zero. The effect of the velocity $\omega(t)$ executed at time t is observed at the next time step, $t + \Delta t$, and the effects are the next rotation angle $\theta(t + \Delta t)$ and the next poured volume $vol(t + \Delta t)$. Other factors that affect the pouring behavior considered in this paper are the shape of the source container, the initial volume in the source container, and the target volume in the receiving container.

The angular velocity, $\omega(t)$, is the action that pushes the pouring process forward. To perform pouring, we can use a motion model that takes the target volume as input and generates the velocity as output. At any time step during pouring, the model takes the current poured volume as input, compares it with the

target volume, and adjusts the velocity accordingly. The model is represented as

$$\omega(t) = G(\omega(t - \Delta t), \theta(t), vol(t), vol_{2pour}), \tag{1}$$

where $G(\cdot)$ denotes the function that relates the previous velocity $\omega(t-\Delta t)$, current angle $\theta(t)$ and volume $vol(t)$, and target volume vol_{2pour} with the current velocity. The pouring process is written again as:

1: **for** i in $(1, 2, \dots)$ **do**
2:     $t = t_1 + (i - 1)\Delta t$
3:     $\omega(t) = G(\omega(t - \Delta t), \theta(t), vol(t), vol_{2pour})$
4:     $\theta(t + \Delta t) = \theta(t) + \omega(t)\Delta t$
5:     $vol(t + \Delta t) = F(\omega(t), \theta(t), vol(t))$
6: **end for**

2.1. Self-supervised learning from demonstration and outcomes

The learned pouring model should have the targeted volume as one of the inputs since the pouring behavior should change based on the target. However, as one of the critical innovations in our approach, we replace the targets with the actual results in training. For example, we ask subjects to pour 150 mL, but the subject pours 165 mL. In this case, the input to the model is 165 mL, not 150 mL. This change enables the proposed self-supervised learning since the subjects in the demonstration do not have to be supervised, and the desired target does not have to be labeled. The robot can observe the actual outcomes and use them as training inputs. This approach allows the model to pour with a human-like pace but achieves better accuracy.

In the model, although taking vol_{2pour} as input, $G(\cdot)$ is not guaranteed to generate the exact $\omega(t)$'s which will lead to vol_{2pour}. In reality, given vol_{2pour}, $G(\cdot)$ will generate $\omega(t)$'s whose execution will lead to a certain final volume vol_{final} in the receiving container. Assume the receiving container initially has a volume vol_{init}, then $vol_d = |vol_{final} - vol_{init} - vol_{2pour}|$ reflects the ability of $G(\cdot)$ to fulfill vol_{2pour}. If $vol_d = 0$, then $G(\cdot)$ is perfect in fulfilling the goal. Reversely, vol_{final} can be considered as the result of executing a *perfect* model $G^*(\cdot)$ assuming its goal is set to $vol_{2pour} = vol_{final} - vol_{init}$. To learn $G(\cdot)$, if we use the actual vol_{2pour}, i.e., the volume we *intend* to reach, then the learned model will approximate the one that is given by vol_{2pour} but ends with vol_{final}. If we set $vol_{2pour} = vol_{final} - vol_{init}$, the learned model will approximate the perfect model $G^*(\cdot)$. In the hope of learning a more accurate motion model, we set the motion goal vol_{2pour} using the actual outcome $vol_{final} - vol_{init}$.

2.2. RNN-based pouring skill model

A straightforward design choice for accurate pouring is a simple PID controller. [4,6] have applied PID for accurate pouring, in which they both used PIDs with fixed gains. While pouring, as the liquid flows out of the source container, the plant changes, which accordingly requires adjustment to the gains of the PID. Performing pouring thus requires an adaptive PID whose gains change their values throughout the pouring process. We speculate that this partly limits the achievable accuracy in [4,6]. An adaptive PID, however, is no longer a simple controller. That justifies our quest for a more complicated motion model.

We aim to learn a motion model from pouring demonstrations that can perform properly in new settings after being trained in a finite number of settings. In this work, we explore the generalization of the pouring skill model to different shapes of source containers, type of liquid, and granular materials. Generalization of neural networks has been observed in practice, and active research has been conducted, which tries to identify possible causes such as the norm of network parameters [37], the specialty of the

network structure and the landscape of the cost function [38], and sharpness/flatness of the minima [39].

Apart from generalization, we seek two other properties from the candidate model:

1. Since all demonstrations are sequences, the model should be inherently capable of dealing with sequences and capturing the spatial–temporal patterns in the sequences.
2. Since demonstrations vary in length, the model should be able to learn effectively from sequences with different lengths.

Due to the successful records of the generalizability of neural networks and our need for a sequential model, we use RNN to represent the motion model. RNN is a class of neural networks that is designed to process its inputs in order. It feeds its output from one time step into its input at the next time step, shown specifically in Eq. (2), where $x(t)$ is the given input, $h(t - 1)$ and $h(t)$ are outputs from the previous and the current step, respectively. The weight W and bias b are learned using Backpropagation Through Time [40].

$$h(t) = \tanh\left(W[h(t - 1), x(t)]^\top + b\right) \tag{2}$$

We need to decide the input features to the RNN at any time step. Each feature corresponds to a type of data. We write Eq. (1) again below for convenience:

$$\omega(t) = G(\omega(t - \Delta t), \theta(t), vol(t), vol_{2pour}) \tag{3}$$

The first feature is the previous angular velocity that can be encoded as the hidden state of the RNN represented by Eq. (2). We also use $\theta(t)$ as a feature. The next two features are $vol(t)$ and vol_{2pour}, respectively. Corresponding to vol_{2pour}, the initial volume of liquid in the source container vol_{total} can be set as a feature. We can also have features that describe the shape of the source container. We model the source container as a cylinder and set both the height H and the body diameter D as features. The four static features vol_{2pour}, vol_{total}, H, and D describe a pouring task and distinguish one task from another. The two sequential features $\theta(t)$ and $vol(t)$ represent the feedback from the rotation on the source container and the volume change on the receiving container. Fig. 4 illustrates the six input features. Therefore, the input and output of the RNN from Eq. (2) become:

$$x(t) = [\theta(t), vol(t), vol_{total}, vol_{2pour}, H, D] \tag{4}$$

$$\omega(t) = K(h(t)) \tag{5}$$

where the function $K(.)$ relates the hidden state of the RNN to the scalar angular velocity. The plain RNN as shown in Eq. (2) suffers from the problem of vanishing and exploding gradients [41, 42], which prevents it from learning long-term dependencies effectively. The problem was solved by long short-term memory (LSTM) which introduces gates and memory cells [42]. Later, peepholes were introduced to LSTM to enable the access of all gates to the memory cell [43]. The mechanism of peephole LSTM is illustrated in Fig. 3 and is written as:

$$i = \text{sigm}\left(W_i[h(t - 1), x(t)]^\top + b_i + p_i \odot c(t - 1)\right) \tag{6}$$

$$f = \text{sigm}\left(W_f[h(t - 1), x(t)]^\top + b_f + p_f \odot c(t - 1)\right) \tag{7}$$

$$g = \tanh\left(W_g[h(t - 1), x(t)]^\top + b_g\right) \tag{8}$$

$$c(t) = f \odot c(t - 1) + i \odot g \tag{9}$$

$$o = \text{sigm}\left(W_o[h(t - 1), x(t)]^\top + b_o + p_o \odot c(t)\right) \tag{10}$$

$$h(t) = o \odot \tanh(c(t)) \tag{11}$$

where i, o, and f are the input, output, and forget gates, respectively. W_i, W_f, W_g, W_o, b_i, b_f, and b_g are the LSTM cell
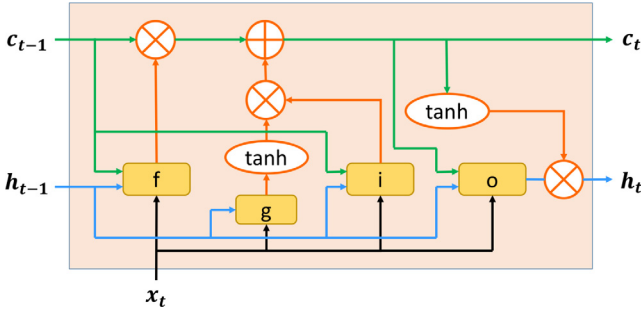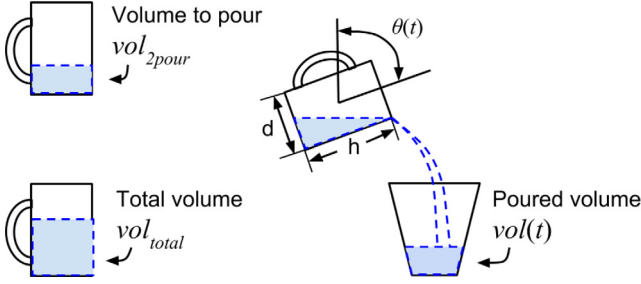
Fig. 3. Mechanism of a peephole LSTM block.

Fig. 4. An illustrative pouring scene that shows the six physical quantities to obtain. vol_{2pour} and vol_{total} is the target and initial volume. D and H are the diameter and height of the source container. $\theta(t)$ and $vol(t)$ are the sequences of rotation angle and of the poured volume.
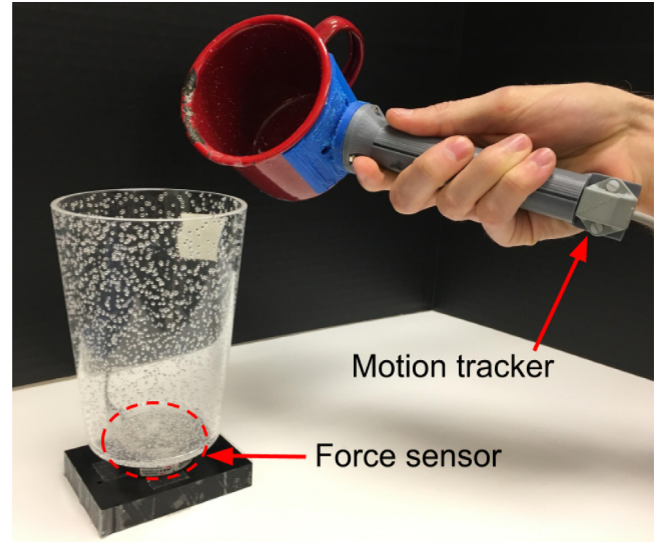

Fig. 5. Illustration of the data collection setup. The source container is connected to the motion tracker through a 3-D printed adapter. The force sensor is placed underneath the receiving container.

weights to learn. p_i, p_o, and p_f are also the peephole connection weights to be learned for gates i, o, and f, respectively. $c(t)$ is the long-term memory, $h(t)$ the output, and $x(t)$ the input of the LSTM block. "sigm" represents the sigmoid function applied element-wise, and is used to implement gates. "tanh" represents the hyperbolic tangent function, applied element-wise, and is used to avoid vanishing or exploding gradients. $\odot$ represent element-wise multiplication. In this work, we use peephole LSTMs.

Taking into account Eqs. (4) to (11) we can see that the dynamic model of the pouring motion is allocated in the long-term memory $c(t)$ of the LSTM. The combination of Eqs. (8) and (9) gives:

$$c(t) = f \odot c(t-1) + i \odot \tanh \left(W_g [h(t-1), x(t)]^\top + b_g \right) \qquad (12)$$

where the dynamic model $c(t)$ depends on both the previous dynamics and the previous input and output. The gate i decides which part of the current input and past output contributes to the current dynamic model, and the gate f decides which part of the past dynamic model contributes to the current dynamic model. Eq. (11) shows that gate o decides which part of the dynamic model will be used as the current output. The previous mechanism allows the LSTM network to predict when to start the backward rotation of the source container based on the information provided by its current input (feedback signals and static features), its past output (past angular velocity) and its long-term memory (past dynamic model).

3. Training

3.1. Data collection

We want to collect all the input features that we have identified for RNN and we need to decide how to measure volume. Intuitively, the volumes vol_{total} and vol_{2pour} can be measured using

a measuring cup. However, obtaining $vol(t)$ using a measuring cup requires a real-time video stream of the measuring cup and a computer vision algorithm that extracts the volume from the video stream. To simplify the problem that we have to solve, we decide that we will not include the above vision problem in our solution, and instead, we compute the volume from other quantities. The volume can be computed as the mass m divided by the density ρ, i.e., $v = m/\rho$. We consider the weight as the gravitational force acting on an object that keeps the object in place. The weight f is the product of mass m and gravitational acceleration g, i.e., $f = mg$. Therefore, the volume can be calculated from the weight:

$$v = \frac{f}{\rho g}, \qquad (13)$$

We represent vol_{total} by its corresponding weight f_{total}, vol_{2pour} by weight f_{2pour}, and similarly the current poured volume $vol(t)$ by weight $f(t)$. Fig. 5 illustrates the setup for our data collection. We collect data of pouring water from 9 different source containers into the same receiving container. The 9 source containers are shown as the left half of Fig. 6. We measure H and D of each source container in millimeters (mm) using a ruler. We 3D-print a handle where the source container is mounted on one end, and a Polhemus Patriot motion tracker is mounted on the other end. The motion tracker records the rotating angles $\theta(t)$'s of the source container in degrees. We place an ATI Mini40 force/torque sensor under the receiving container to record the raw force reading $f_{raw}(t)$ in pound-force (lbf).

We obtain f_{total} and f_{2pour} from $f_{raw}(t)$. f_{2pour} is calculated by $f_{2pour} = f_{final} - f_{init}$, where f_{init} and f_{final} are the weights read from the receiving container before and after a trial, respectively. Thus, we set f_{2pour} using the actual poured outcome. In each trial, $f_{total} > f_{2pour}$, that is, there is water left in the source container after pouring. Various f_{total} and f_{2pour} are recorded to aid the generalizability of the prospective motion model. $\theta(t)$'s are recorded at 60 Hz and $f_{raw}(t)$'t are recorded at 1 kHz. The collected pouring data is part of RPAL Daily Interactive Manipulation (DIM) dataset [3], which is publicly available. More manipulation datasets can be found at [44].

Fig. 6. Source containers used for (left) training and for (right) evaluation. The red cup was used both for training and for evaluation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
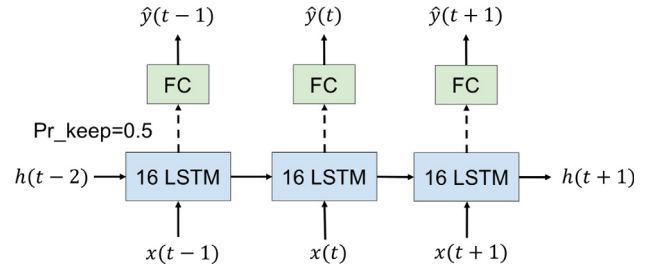

Fig. 7. Our network has 1 layer and 16 peephole LSTM units. Dropout with a keep probability of 0.5 is applied to non-sequential connections.

3.2. Implementation

The neural network can have multiple layers and each layer can contain multiple peephole LSTM units. By units we mean the size of the vectors c_t and h_t of Fig. 3 formalized by Eqs. (9) and (11), respectively. Dropout [45] is applied between layers to avoid memorizing the data and aid generalizability. The final layer is a fully connected layer with linear activation which generates the angular velocity. The mechanism of the network with L layers at time t is represented as

1: $h_0(t) = x(t)$
2: **for** $i = (1, 2, \ldots, L)$ **do**
3: $\quad h_i(t) = \text{LSTM}(h_i(t-1), h_{i-1}(t); n_{unit})$
4: $\quad h_i(t) = \text{Dropout}(h_i(t); p_{keep})$
5: **end for**
6: $\hat{y}(t) = W_y h_L(t) + b_y$

where $\text{LSTM}(\cdot; n_{unit})$ means using an LSTM block from Fig. 3 with n_{unit} units. $\text{Dropout}(\cdot; p_{keep})$ means dropout with a keep probability of p_{keep}. $\hat{y}(t)$ corresponds to a linear layer that converts the hidden state $h_L(t)$ to the final output.

To feed the input features into the network, we group them into a vector $x(t) = [\theta(t), f(t), f_{total}, f_{2pour}, H, \kappa]^\top$ for $t = 1, \ldots, T - 1$, where T is the length of the trial and

1. $\theta(t)$ is the rotating angle of the source container.
2. $f(t)$ is the weight of the poured liquid.
3. f_{total} is the weight of the initial amount of liquid present in the source container before pouring.
4. f_{2pour} is the weight of the target poured amount.
5. H is the height of the source container.
6. κ is the body curvature of the source container.

The body curvature κ of the source container is calculated from the body diameter, D:

$$\kappa = 2/D \tag{14}$$

The angular velocities $\omega(1 : T - 1)$ are computed from $\theta(1 : T)$:

$$\omega(t) = (\theta(t+1) - \theta(t))f_s, \qquad t = 1, 2, \ldots, T - 1 \tag{15}$$

where f_s is the sampling frequency of $\theta(t)$. For each trial, at time $t \in [1, 2, \ldots, T - 1]$, the input $x(t)$ and target $y(t)$ of the network are

$$x(t) = [\theta(t), f(t), f_{total}, f_{2pour}, H, \kappa]^\top \tag{16}$$

$$y(t) = \omega(t) \tag{17}$$

The output of the network is denoted by $\hat{y}(t)$. Assume we have N trials in total, and each trial has length T_i, $i \in [1, 2, \ldots, N]$. The loss function is defined as

$$c = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{T_i - 1} \sum_{t=1}^{T_i - 1} (\hat{y}_i(t) - y_i(t))^2. \tag{18}$$

3.3. Data preparation

We set the sampling frequency $f_s = 60$ Hz since it is the lower one between the frequencies of $\theta(t)$ and $f_{raw}(t)$. We kept the recorded $\theta(t)$'s intact and downsampled $f_{raw}(t)$ to 60 Hz. We obtain $f(t)$ by filtering the raw reading from the force sensor $f_{raw}(t)$, specifically

$$f_m(1 : t) \leftarrow \text{median\_filter}(f_{raw}(1 : t)), \quad \text{window\_size} = 5, \tag{19}$$

$$f(t) \leftarrow \text{Gaussian\_filter}(f_m(1 : t)), \quad \sigma = 2. \tag{20}$$

We normalize each input dimension independently using the mean and standard deviation of that dimension. The model had 1 layer and 16 LSTM units. We trained models with different numbers of layers and LSTM units, and we found the model with 1 layer and 16 units had a simple structure and performed well. We set the keep probability of dropout to be 0.5. Specifically, the computation for time step t is represented as:

$$h(t) = \text{LSTM}(h(t-1), x(t)) \tag{21}$$

$$h_d(t) = \text{Dropout}(h(t)) \tag{22}$$

$$\hat{y}(t) = W_y h_d(t) + b_y \tag{23}$$

The network is shown in Fig. 7.

The learning model involves 284 trials in total, among which 221 are for training and 63 for validation. Each iteration is an epoch, in which the entire training and validation data are traversed. We ran 2000 epochs and picked the model that has the lowest validation loss. We used the Adam optimizer and set the initial learning rate to be 0.001. The code is written using TensorFlow.

4. Generalization by self-supervised practicing

The robot with the motion model has limited generalizability related to certain pouring containers for which it achieved high pouring error. We recorded the outcomes that the robot provided when pouring with the poor-performing containers. We used those outcomes to fine-tune the model. We refer to this fine-tuning process using the outcomes generated by the robot as generalization by self-supervised practicing (GSSP). GSSP updates the motion model by training it with data generated by practices. Although the outcomes in the first few practices may present a high volume error, they contain the response of the model to new conditions and are therefore valuable. GSSP mimics the behavior of humans when they face a new instance of the pouring task. When we use a cup that we are not familiar with, we pour based on our experience. Shortly after a few practices, the model improves and adapts to the new cup. The key component in GSSP is that the robot uses the actual outcomes instead of the desired outcomes to fine-tune the model.

We refer to the process of using the model on a robot to finish a pouring trial as a *practice*. We use the robot's outcome sequences that result from several practices to form a new dataset, which we use to fine-tune the model. The fine-tuning also uses the actual outcomes as the motion goal. The inputs for the fine-tuning process are:

- H: height of the source container.
- $\kappa = 2/D$: body curvature of the source container with d being the diameter of the container.
- f_{total}: the initial weight of water present in the source container before pouring.
- f_{2pour}: the actual weight of water that was poured during the practice.
- $f(t)$: the sequence of the weight of water in the receiving container in the practice.
- $\theta(t)$: source container's current angle during the practice.

The output of the fine-tuning is:

- $\omega(t)$: angular velocity of the source container **in the practice**.

Before fine-tuning, we obtain an initial model through the presented self-supervised learning from demonstration. The fine-tuning process in GSSP can be carried out in two ways:

- **Gradual Fine-tuning**: First, the robot performs n practices, where n is relatively small. The resulting pouring sequences form a new dataset, which is used to fine-tune the model. If the error of the updated model is larger than a predefined threshold, n practices are performed again, the newly generated n data points are added to the new dataset. The practices keep being performed and the new dataset keeps growing until the error of the updated model is below the predefined threshold. We define the error threshold to be two times that of an averaged human's pouring error. Algorithm 1 describes gradual fine-tuning with $n \in [5, 15]$.
- **Batch Fine-tuning**: First, the robot performs n practices, where n is relatively large. The resulting pouring sequences are used to fine-tune the model. Batch fine-tuning is equivalent to conducting one iteration of gradual fine-tuning with a large n, i.e., $n > 35$, where 35 is the average number of samples per source container for the initial training set discussed in Section 3.

The GSSP approach can be used for any manipulation where the actual outcomes of several practices can be used as the desired outcomes, i.e., the actual outcome can substitute the desired outcome for training. Taking throwing objects into bins as an example [46], we can practice throwing with the robot using unseen objects, record the outcome, and apply GSSP to generalize the learning model to new objects. Another example can be recording the state of food ingredients [47], or a state change [48] after the execution of a manipulation, record the action sequences, use them as training samples, and fine-tune the manipulation model to expand its generalization. However, the GSSP approach may not generalize well among different manipulations. For example, a throwing motion model may not be generalized to mixing manipulations since the two motion models may have different model structures. Our latest work on motion code and motion embedding [49,50] may help in this kind of cross-motion-type generalization.

5. Experiments & evaluation

To evaluate the motion model, we built a robotic system that consists of the trained RNN, a Dynamixel MX-64 motor, and the
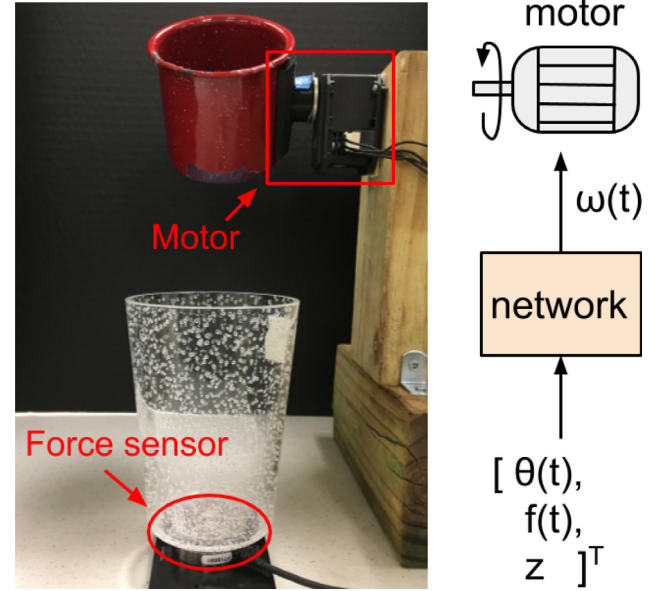

Fig. 8. (Left) The robotic system consists of a motor that executes the generated velocity command and a force sensor that monitors the poured amount. The source containers are attached to the motor through a 3-D printed adapter. (Right) Before pouring, we obtain the static features $z = [f_{total}, f_{2pour}, H, \kappa]$. At time step t, the robotic system obtains $\theta(t)$ and $f(t)$, combine them with z, and send to the network. The network generates velocity command $\omega(t)$ which is executed by the motor.

same force sensor with which we collected the data. The motor was placed at a certain height above the surface. The force sensor was placed on the surface close by. The source container was attached to the motor. The receiving container was placed on top of the force sensor. We properly placed the receiving container (along with the force sensor) according to the particular source container used so that there is little spilling. Fig. 8 (Left) shows the setup of the robotic system. It runs at 60 Hz, the same as the data collection. The time between consecutive time steps is $\Delta t = 0.016$ s. Before performing each separate pouring trial, we obtain the four static features which we denote by $z = [f_{total}, f_{2pour}, H, \kappa]$. During the trial, at time step t, we obtain $\theta(t)$ from the motor and $f(t)$ from the force sensor, and we feed the input features $x(t) = [\theta(t), f(t), z]^\top$ to the model, which then generates the velocity $\omega(t)$. The motor executes the velocity. The above process repeats at time step $t + \Delta t$. Fig. 8 (Right) shows the working process of the robotic system at time t.

The robotic system:

1. Normalized every input dimension.
2. Obtained $f(t)$ by filtering the raw force readings.

in the same way as in training. We evaluated the motion model by testing it on pouring certain kinds of liquid from certain source containers. The difficulty of the task changes when the liquid and the source container change. For each pair of liquid and source containers, the model pours 15 times, each time with arbitrary vol_{total} and vol_{2pour}, where $vol_{total} > vol_{2pour}$. We show the pouring error of each pair of liquid and source container in the form of figures. In the figure, we plot the actual poured volume against the target volume for all 15 trials. We also show the liquid type, the mean, and standard deviation of the pouring error: μ_e and σ_e in milliliters. At the bottom right of the figure, we show the source container that was used. We also show a black dashed line that illustrates zero pouring error. Translating the force reading to volume requires the density of the liquid ρ and the gravitational acceleration g. We used 0.997 g/mL for the density of water and 9.80665 m/s^2 for gravitational acceleration.

Algorithm 1 Generalization by self-supervised practicing (gradual fine-tuning)

1: M_{init}: Initial model
2: n: Number of practices
3: $\mathcal{R} \leftarrow \{(r_s^1, r_g^1), \ldots, (r_s^n, r_g^n)\}$ ▷ r_s^i: start state; r_g^i desired outcome or goal
4: $err\_th \leftarrow$ error threshold
5: $\mathcal{D} \leftarrow \{\}$
6: **procedure** PRACTICE($M, n, \mathcal{R}$)
7:     **repeat**
8:        Robot practices once using one item in $\mathcal{R}$ using model M
9:        $\mathcal{D} \leftarrow \mathcal{D} \cup \{d\}$ ▷ d : outcome sequence from the practice
10:    **until** n practices have been performed
11:    error $\leftarrow$ mean error between actual outcome and desired outcome among n practices
12:    **return** error
13: **end procedure**
14: **procedure** GSSP
15:    $M_{new} \leftarrow M_{init}$
16:    **while** True **do**
17:       err $\leftarrow$ PRACTICE($M_{new}, n, \mathcal{R}$)
18:       **if** $err < err\_th$ **then**
19:         *break*
20:       **else**
21:         $M_{new} \leftarrow$ Fine-tune($M_{new}, \mathcal{D}$)   ▷ Fine-tune the model using $\mathcal{D}$
22:         $\mathcal{R} \leftarrow$ Generate-Random-Practices (n)   ▷ Randomly generate n set of requirements for future practices
23:       **end if**
24:    **end while**
25:    **return** M_{new}
26: **end procedure**

Table 1
Errors of pouring water from different source containers.

Cup	Cup in training	μ_e (mL)	σ_e (mL)
Red	Yes	3.71	3.88
Water bottle	No	4.12	4.29
Bubble	No	6.77	5.76
Glass	No	7.32	8.24
Fat bottle	No	12.35	8.88
Red (by human)	n/a	12.37	9.80
Measuring cup	No	11.29	12.82
Wine bottle	No	51.22	39.61
Blue bottle	No	55.84	47.26

5.1. Model evaluation

5.1.1. Model evaluation of pouring water

We started with the task that has the lowest difficulty and tested the model by pouring water from the red cup that has been used for training. Fig. 12(a) shows a small error of $\mu_e = 3.71$mL, indicating that the learning is successful. Then we increase the difficulty of the tasks and test the model by pouring water from different source containers that have not been used for training. Table 1 summarizes the mean and standard deviation of the errors, μ_e and σ_e, in milliliters of the model pouring water from different source containers, and of the human pouring water from the red cup. Figs. 12 (b) through (e) show the error of five source containers whose σ_e is smaller than that of human. Compared with the error of using the red cup $\mu_e = 3.71$mL, the error of using the five source containers is larger, ranging from $\mu_e = 4.12$mL to $\mu_e = 12.35$ mL, which is expected. Based on the results, we call them "accustomed containers". We have also evaluated the model on a UR5e robotic arm using the water bottle (Fig. 10) in which $\mu_e = 7.83$ mL and $\sigma_e = 6.62$ mL.

We wanted to compare the pouring model with humans and therefore we asked four human subjects to do accurate water pouring with the red cup. We made an animation on a computer screen that shows the target volume and the real-time volume
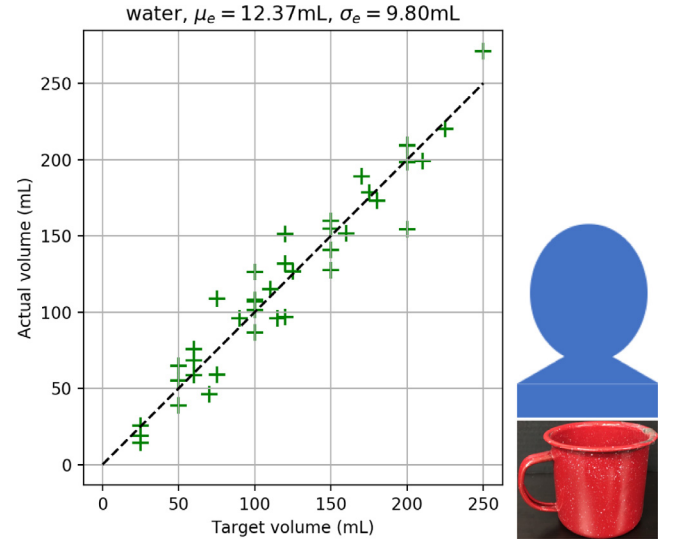

Fig. 9. Actual-vs-target comparison of 4 human subjects pouring water from the red cup. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of water that has already been poured. The animation faithfully shows the fluctuation of the volume reading while pouring. The subjects were asked to look only at the animation and pour the target volume. They were asked to pour naturally and with a single pour. Pouring too fast or too slow was not allowed. We collected 10 trials with each subject, resulting in 40 total trials. Fig. 9 shows the results of human accurate pouring: $\mu_e = 12.37$mL and $\sigma_e = 9.80$mL. Compared with humans, pouring water from the red cup (Fig. 12(a)) achieves a lower $\mu_e = 3.71$mL and $\sigma_e = 3.88$mL. The model achieves lower error than humans because the model is trained using the actual outcomes.
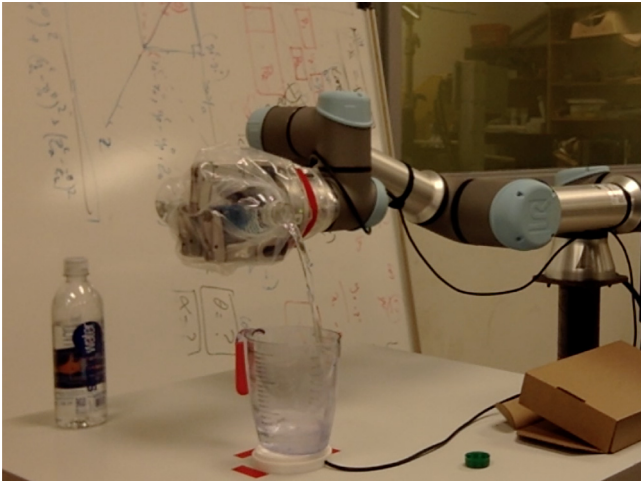
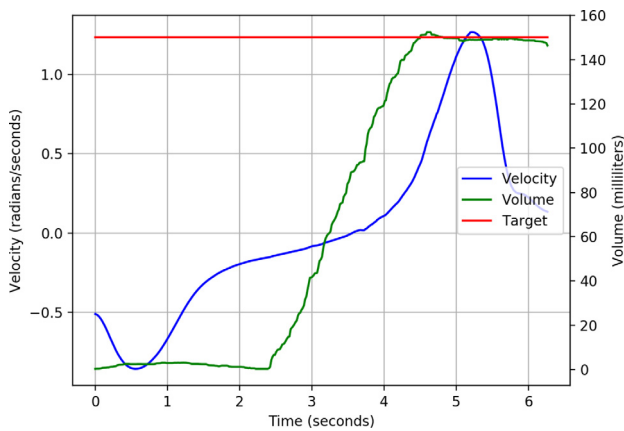**Fig. 10.** Evaluating the model on UR5e collaborative robotic arm.

Fig. 11. An example of the robot pouring water using the water bottle. Velocity: velocity of the source container measured counter clock-wise, negative radians/seconds means that the container is going forward and positive means backward. Volume: volume of water in the receiving container. Target: target volume for the trial.

Fig. 11 shows an example of the resulting $\omega(t)$, $vol(t)$, and vol_{2pour} for a trial of pouring water using the trained model with the water bottle. The target vol_{2pour} was established as 150 mL from an initial vol_{total} of approximately 430 mL. The final amount poured was 149 mL, an error of 1 mL. We can see that for this example the pouring was finished in less than 6 s and the water stopped to be poured out in less than 5 s. The small spike seen in the volume is due to the force sensor's noise generated by the water movement. We measured the final volume poured once the water had stabilized in the receiving container.

We have also tested the model on two containers that are significantly different from the training set: a wine bottle and a blue bottle that were available in our laboratory. As shown in Table 1, their mean volume errors were 51.22 mL and 57.06 mL, respectively. The mean volume error for the wine bottle and the blue bottle is 13 times higher than that of the red cup (3.71 mL). They are unaccustomed containers to our pouring model because of their unusual shapes.

Having evaluated the error of the model pouring different but relatively large amounts of water, we evaluated the error of the model pouring a small amount of water. We use the model to pour 20 mL and 15 mL using the red cup, respectively, each for 15 times. For 20 mL $\mu_e = 9.68$ mL and $\sigma_e = 7.96$ mL. For 15 mL

Table 2
Results of pouring with MPC or Switch Controller.

Cup	Model	Cup in training	μ_e (mL)	σ_e (mL)
Red	MPC	Yes	7.25	4.92
	Switch ω_1		33.50	7.76
	Switch ω_2		4.50	1.87
Glass	MPC	No	14.25	9.11
Bottle	MPC	No	15.88	5.13
Fat	MPC	No	18.25	8.30
Bubble	MPC	No	26.13	6.29
	Switch (ω_1)		56.25	5.85
	Switch (ω_2)		22.25	4.29

$\mu_e = 2.83$ mL and $\sigma_e = 3.33$ mL. Both μ_e and σ_e for pouring 20 mL are smaller than those of pouring a larger volume with the red cup (Fig. 12(a)). The error of pouring 15 mL is larger than both the error of pouring 20 mL and the error of pouring a larger volume. In Fig. 13, we plot the reading of the force sensor for a 1.0-lbf weight for 300 s. Fig. 13 also shows the water volume converted from the corresponding force. For a 1.0-lbf weight, the force sensor has a nonlinearity error of around 0.01 lbf, which is 1% of 1.0 lbf. The corresponding error in volume is around 5 mL.

5.1.2. Comparison with related works

The difficulty of accurate pouring increases as the duration it takes to pour decreases. Our model pours as fast as humans. The duration of each pour in the human demonstrations dataset ranges from 3.2 to 8.7. The duration range of our model was 2.8 to 7.6 s. In comparison, [4] achieves 38 mL error using 25 s for each pour using similar containers as our accustomed containers, it takes much longer than ours and causes a larger error. [5] achieves under 5 mL error and uses 20–45 s for each pour, it achieves lower error than ours but takes longer than [4]. We are not aware of any prior art that has evaluated containers as diverse as ours.

Our approach uses weight to monitor the poured volume. [4, 6,7] also use one single modality (weight or vision) to monitor the poured volume, whose reported pouring error is 38 mL [4], 23.9 mL, 13.2 mL, and 30.5 mL [6], and 19.96 mL [7] respectively. [5] achieves an error that is under 5 mL, but two modalities (weight and vision) are used to monitor the poured volume. It also pours slowly, a behavior that can help with the higher accuracy achieved. The error reached by the model lies between 3.71 mL and 12.35 mL, lower than the above approaches.

In our previous work [8], we used model predictive control (MPC) to address the problem of accurate pouring. The proposed controller uses RNN to predict the weight of the liquid in the receiving container and controls the angular velocity of the source container. We evaluated the performance of the controller by comparing it with a switch controller. The switch controller applies a constant forward velocity to the source container when the volume in the destination container is less than the target. It applies a constant backward velocity when the volume reaches the target. Table 2 shows the results. The Switch ω_1 controller used 20 deg/sec as the forward velocity and -30 deg/sec as the backward velocity. The Switch ω_2 controller used 5 deg/sec as the forward velocity and -7.5 deg/sec as the backward velocity. We can see that when the forward angular velocity becomes smaller, the mean volume error decreases. This result is expected since the difficulty of controlling the volume in the destination container decreased. Based on Table 2, the errors for the MPC controller go from 7.25 mL and 26.13 mL. We can see that model M_0 also performs better than the MPC controller. Without the guarantee that the RNN-generated physics model is highly precise, the performance of the powerful MPC algorithm is compromised. Pouring is not a trivial task that can be solved by a traditionally powerful algorithm.
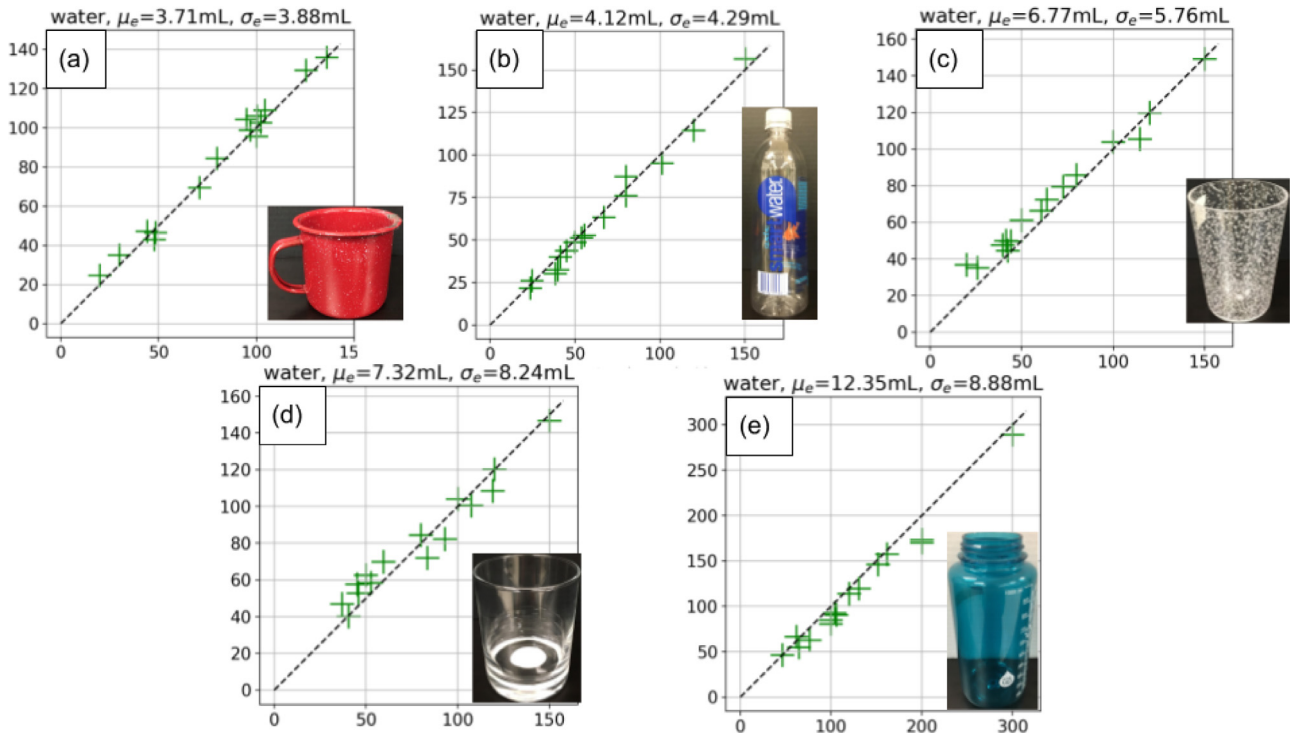
Fig. 12. Actual-vs-target comparison of pouring water using (a) red cup which is used for training (b) water bottle (c) cup with bubble pattern which we referred to as the bubble cup (d) glass cup (e) fat bottle.
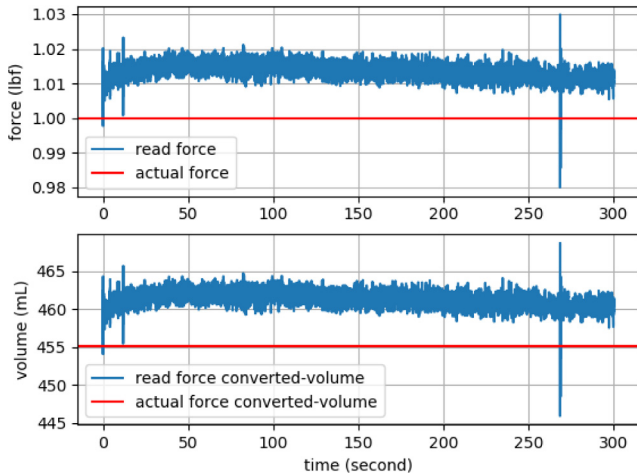

Fig. 13. Readings of the force sensor for a weight of 1.0 lbf taken during 300 s. The bottom subfigure shows the volume converted from force.

Fig. 14. Pouring accuracy of liquids with different viscosity. *x*-axis plotted in logarithmic scale.

5.1.3. Model evaluation on different materials

We also tested the model on liquids with different viscosity from water such as cooking oil and syrup. We used the red cup to pour since it is part of the training dataset. However, the dataset comes from pouring motions of water. We speculate that viscosity played an important role in the accuracy of pouring different kinds of liquids. Therefore, Fig. 14 shows the error bars of pouring water, oil, and syrup with the red cup versus their viscosity for 15 pours. The three types of liquids have very different viscosities. We use 1 centipoise (cps) as the viscosity for water, 65 centipoises for oil, and 2000 for syrup. We plot the viscosities on a logarithmic scale. We can see that the mean error increases as the viscosity increases. The relationship is neither linear nor exponential.
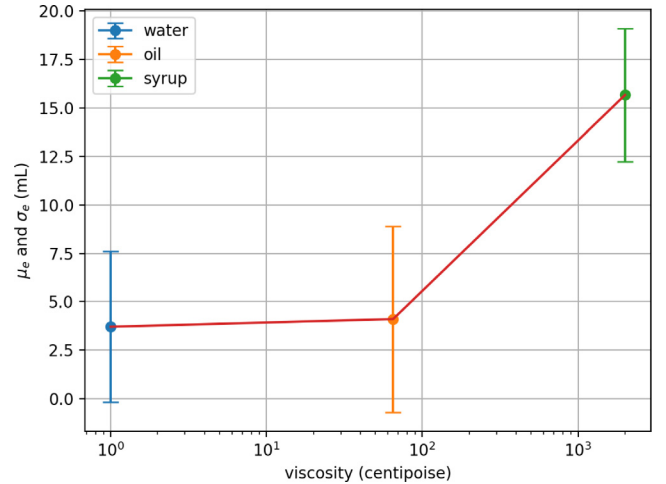
We also evaluated the model on granular materials poured in cooking scenarios: beans and rice. We used the same model trained from pouring water. Fig. 15 shows the result of 15 pours of beans and rice using the red cup. We changed the unit of measure to grams (g) as it is more suitable for solid materials than volume. We can see that the mean error is small for rice, whereas it is higher for beans. However, the mean errors presented by the model are similar to those presented in [51], where the authors state to have a mean error estimation of 14.3g for red beans and 4.36g for pouring rice. Their approach is not meant for accurate pouring but for pouring mass estimation based on fingertip sensors of a robotic hand.
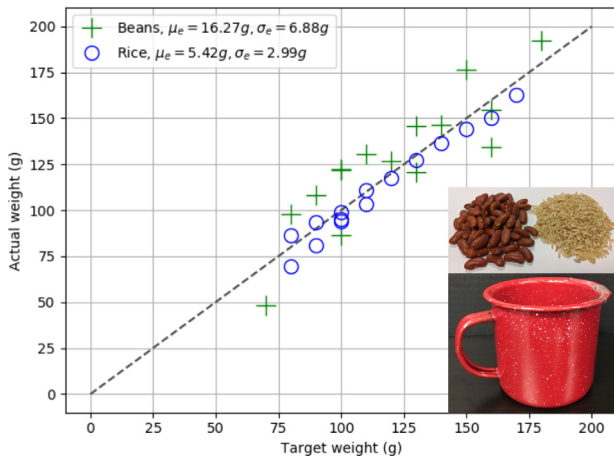
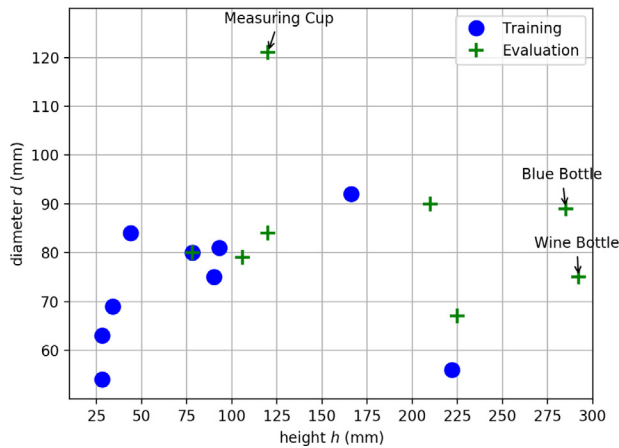**Fig. 15.** Pouring accuracy of beans and rice using the red cup.

Fig. 16. Scatter plot of height vs. diameter for source containers used for experiments.

5.2. GSSP evaluation — generalization to unaccustomed containers

We evaluated GSSP on the two unaccustomed containers — a wine bottle and a blue bottle. For comparison, we also added the measuring cup into the unaccustomed container set. We chose those three containers because, as Table 1 shows, the wine bottle and blue bottle had the highest mean errors among all the test containers evaluated. Although the measuring cup did not present a significantly high mean error, it did present a much higher standard deviation error than humans. Fig. 16 shows the scatter plot of the height versus diameter of the source containers used for experiments. We can see that the wine bottle and the blue bottle are both much taller than the rest of the containers, and the measuring cup has a much larger diameter.

We evaluated GSSP using batch fine-tuning for the wine bottle, blue bottle, and measuring cup. We evaluated the accuracy of the resulting models by pouring water 15 times per experiment for batch fine-tuning and carried out the experiments maintaining the same set of volumes for a fair comparison. We also evaluated GSSP using gradual fine-tuning for the wine bottle. We refer to the initially learned model that was evaluated in Section 5.1 as M_0.

5.2.1. Wine bottle

We executed a total of 36 practices using the initial model M_0 with the wine bottle for different f_{total} and f_{2pour}. Then, we
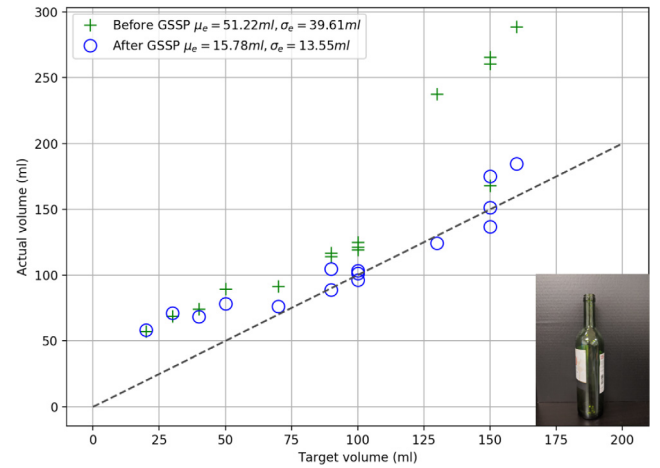

Fig. 17. Results for Wine Bottle before and after GSSP.

Table 3
Accuracy for wine bottle after gradual fine-tuning.

Base model	Fine-tuned model	μ_e (mL)	σ_e (mL)
M_0		80.23	49.13
M_0	M_2	38.67	11.98
M_2	M_3	30.04	17.26
M_3	M_4	18.21	8.76

fine-tuned the model using the obtained dataset. We call the fine-tuned model M_1. Fig. 17 shows the mean and standard deviation of the error for 15 pours before and after applying GSSP, i.e., using models M_0 and M_1, respectively. We can see that the wine bottle's mean volume error became 15.78 mL, a reduction of around 69% from 51.22 mL mean error. After applying GSSP, some trials over pour but others under pour. When using model M_0, all trials over pour water.

We also tested the gradual fine-tuning approach using the wine bottle for which we set the number of practices $n = 10$. The set of volume variations $\mathcal{R} = \{(f^i_{total}, f^i_{2pour})\}$ for $i = 1, \ldots, 10$, was chosen to be the same for each iteration of the algorithm. Table 3 shows the accuracy evolution of the fine-tuning algorithms we carried out. The mean and standard deviation errors from the table's first row are different from the ones shown in Table 1 as the set $\mathcal{R}$ and the number of trials was different for both experiments. We hypothesize that running more iterations of the algorithm will further improve the result. However, we believe that there should exist enough variation in the selection of f_{total} and f_{2pour} to outperform the result of batch fine-tuning for this particular container.

Comparing the results of Table 3's fourth row with Fig. 17 after applying GSSP, we can see that both methodologies yield similar results. Gradual fine-tuning has the advantage over batch fine-tuning w.r.t. the cost it takes to collect the practices. However, there exists a trade-off in training time: batch fine-tuning only trains once, while gradual fine-tuning trains several times. Nevertheless, gradual fine-tuning allows us to realize whether there is an improvement or not after the first iteration of the algorithm using only a few practices. In batch fine-tuning, after taking a considerable time carrying out practices, we expect that there exists an improvement, but the practices collected may lead to unsatisfactory results.

5.2.2. Blue bottle

We decided to apply only batch fine-tuning for the blue bottle. We executed 54 practices using the blue bottle and fine-tuned M_0. We call the resulting model M_5. Fig. 18 shows the mean
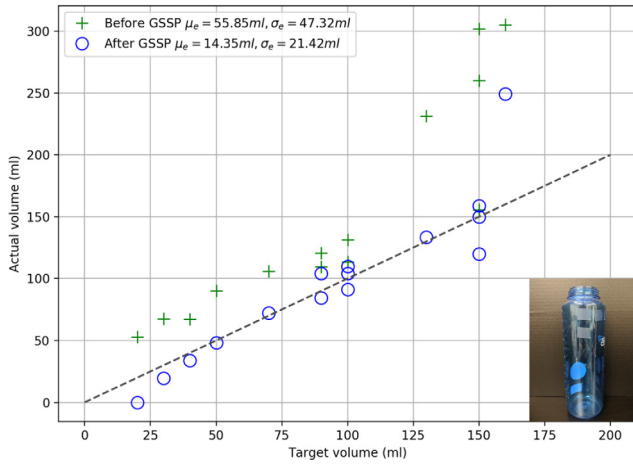
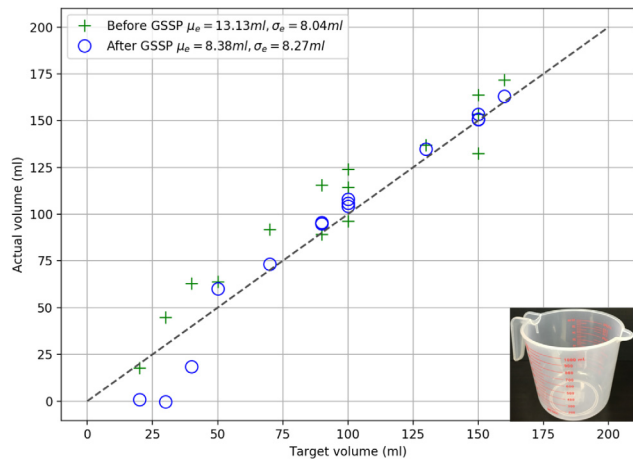**Fig. 18.** Results for Blue Bottle before and after GSSP.

Fig. 19. Results for Measuring Cup before and after GSSP.

and standard deviation of the error for 15 pours before and after applying GSSP, i.e., using M_0 and M_5, respectively. We can see a reduction of 74% in average error from 55.85 mL to 14.35 mL. We can also see that there exists an outlier for the target of 160 mL that may be affecting the standard deviation. However, there was a 54% reduction in this statistic from 47.32 mL to 21.42 mL.

5.2.3. Measuring cup

We executed 36 practices using the measuring cup. We also decided to use batch fine-tuning for this source container and applied GSSP. We call the resulting model M_6. Fig. 19 shows the scatter plot of the target versus the actual poured volume for the measuring cup before and after GSSP, i.e., when using M_0 and M_6, respectively. We can see a reduction again in mean error when comparing the target with the actual volume poured by the robot.

5.3. GSSP evaluation — generalization to new material

We have also carried out GSSP experiments on syrup and red beans. We used the red cup as the container for pouring with the model M_0. This model comes from training using the human demonstrations dataset of pouring water. We selected the red cup as it is the most accurate container for pouring water with model M_0.
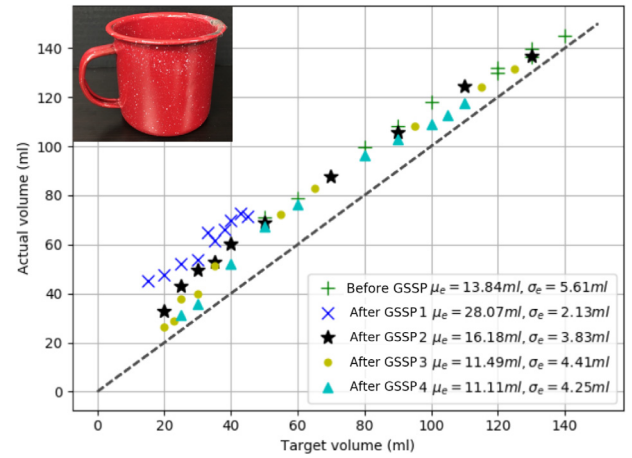

Fig. 20. Accuracy for syrup before and after 4 gradual fine-tuning iterations.

Table 4
Comparison of accuracy for red cup before and after fine-tuning M_0 using data collected for red beans.

Source container	Model	μ_e (g)	σ_e (g)
Red cup	M_0	16.27	6.88
	M_{11}	11.49	6.98

5.3.1. Syrup

We applied gradual fine-tuning for syrup using model M_0 for which we set the number of practices $n = 10$. Fig. 20 shows the evolution of accuracy for 5 iterations of Algorithm 1. We can see that the initial mean error of 13.84 mL was mostly affected for target volumes lower than 100 mL. For the second iteration, we decided to pour small volumes for which we could see a mean volume error of 28.07 mL. For the third iteration, we could see a reduction of the mean error from 28.07 mL to 16.18 mL (42% reduction). We could see that the improvement was caused by targets lower than 50 mL and higher than 80 mL. For the fourth iteration, we could see an improvement from 16.18 mL to 11.49 mL (28% reduction). At this stage, we were able to reduce the initial mean error from 13.8 4 mL to 11.49 mL (17% reduction). Finally, we carried out a fifth iteration for which the mean error decreases from 11.49 mL to 11.11 mL (3% reduction).

5.3.2. Red beans

We collected a total of 36 practices for pouring beans with different initial target weights, where the 15 pouring trials shown in Fig. 15 are included. Table 4 summarizes the mean and standard deviation weight errors. Model M_{11} results from fine-tuning M_0 using the 36 pouring trials collected. We can see that GSSP not only works for reducing the error when pouring liquids from new source containers. It also can be used to improve the accuracy of pouring a different material. Interestingly, the model M_0 trained for pouring water has a considerable small error when used for pouring beans.

5.4. GSSP discussion

At this point, we have applied GSSP as formulated in Algorithm 1 where the dataset used to fine-tune comes entirely from the robot practices. We performed further experiments to analyze the effects of combining the human demonstrations dataset described in Section 3.1 with the robot practices. The hierarchical diagram of Fig. 21 illustrates the details of the fine-tuned models that we have presented so far and the new ones we will present next. The models presented in Sections 5.2 and 5.3 correspond to the
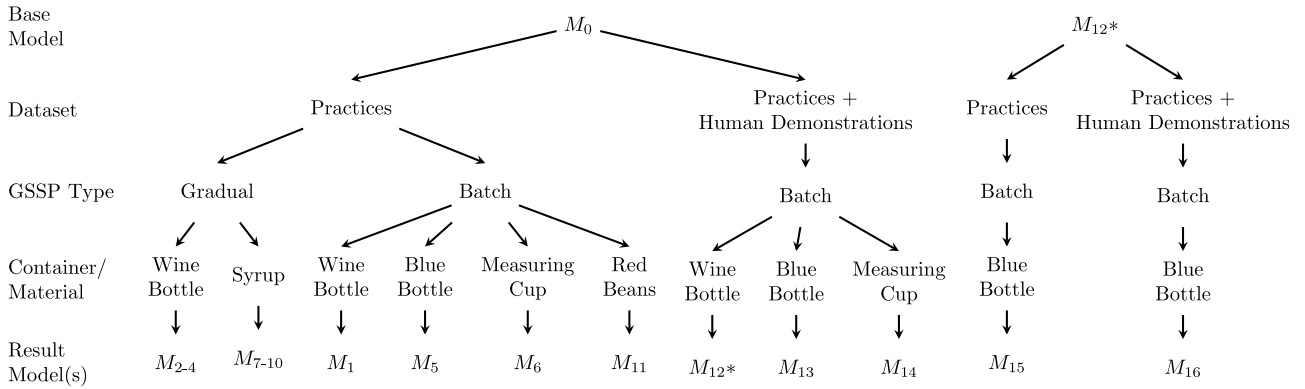
Fig. 21. Model relationships for the application of batch or gradual GSSP to particular datasets, containers, or materials.

Table 5
Accuracy comparison of pouring water using the fine-tuned models.

Source container	Base model	Fine-tuned model	μ_e (mL)	σ_e (mL)
Wine bottle	M_0	M_1[a]	15.78	13.55
		M_{12}[b]	17.43	13.65
Blue bottle	M_0	M_5[a]	14.35	21.42
		M_{13}[b]	12.65	5.99
Measuring cup	M_0	M_6[a]	8.38	8.27
		M_{14}[b]	8.03	8.34

[a]Model fine-tuned with robot practices only.
[b]Model fine-tuned with robot practices plus human demonstrations.

Table 6
Accuracy of pouring water with the red cup using the fine-tuned models.

Source container	Base model	Fine-tuned model	μ_e (mL)	σ_e (mL)
Red cup	M_0	M_1[a]	31.88	19.60
		M_{12}[b]	8.67	5.13
		M_5[a]	25.29	14.98
		M_{13}[b]	8.52	5.51
		M_6[a]	4.84	2.79
		M_{14}[b]	6.95	5.57

[a]Model fine-tuned with robot practices only.
[b]Model fine-tuned with robot practices plus human demonstrations.

children of the "Practices" branch of Fig. 21. Such models were derived from fine-tuning M_0 with robot practices using the same model. In the following sections, we will compare the results of new models fine-tuned with a combination of robot practices and human demonstrations, versus the models fine-tuned with robot practices only. We also carried out experiments using the red cup with the fine-tuned models to verify the impact of accuracy to the accustomed containers. Finally, we will also show the results of applying GSSP to the robot practices using a fine-tuned model instead of M_0.

5.4.1. Combination of practices and human demonstrations

We fine-tuned model M_0 using the combination of the human demonstrations dataset with the robot practices. We believe this is the same as training from scratch using the combined datasets, the only difference being that the training converges faster as M_0 already learned from the human demonstrations. Table 5 shows the summary of mean and standard deviation errors for the fine-tuned models. M_{12} is the fine-tuning of M_0 with the training dataset of human demonstrations from Section 3.1 plus the batch of wine bottle robot practices from Section 5.2.1. Similarly, M_{13} and M_{14} are the results of fine-tuning M_0 with the training dataset of human demonstrations of Section 3.1 plus the blue bottle and measuring cup batches of robot practices, respectively.

Model M_{12} has slightly higher mean and standard deviation errors than model M_1 w.r.t. wine bottle, meaning that model M_1 is better suitable for pouring accurately with the wine bottle. Interestingly, model M_{13} slightly outperformed model M_5 w.r.t. the blue bottle's accuracy. Similarly, model M_{14} marginally outperformed model M_6. Given this marginal improvement, fine-tuning using only the robot practices is sufficient to achieve a satisfactory precision. Moreover, the original human demonstrations dataset is necessary to generate models M_{13} and M_{14} with the increased cost of training with a larger dataset.

5.4.2. GSSP effect on accustomed containers

We investigated how the fine-tuned models affected the generalization of pouring containers that already work well with M_0. To this end, we selected the red cup as it was the best performing container for M_0 and is also part of the human demonstrations dataset. From Table 1, it has $\mu_e = 3.71$ mL and $\sigma_e = 3.88$ mL. Table 6 shows the accuracy of 15 pouring trials using the red cup using models fine-tuned with robot practices only and the combination of practices with human demonstrations. Based on the results, we can see that the accuracy of the red cup was severely affected by the fine-tuned models that come from the practices of the wine bottle and the blue bottle, i.e., M_1 and M_5, respectively. The accuracy was not drastically impacted by using M_6.

We believe this is related to the fact that M_0 was already performing well with the measuring cup. Therefore, the fine-tuning needed to learn more, e.g., modifies M_0 more drastically, from the practices of the wine bottle and the blue bottle than from the measuring cup's. Based on this result, we can state that models M_1 and M_5 are specialized models that accurately pour with the wine bottle and the blue bottle, respectively. At this stage, the robot can use a selector such that when it needs to use the wine bottle, M_1 is chosen to pour. Similarly, when it needs to use the blue bottle, M_5 is chosen to pour.

5.4.3. GSSP on a different base model

We also investigated the effect of applying GSSP to different starting models. We selected model M_{12} that was the result of fine-tuning model M_0 using the combination of the dataset of human demonstrations plus the wine bottle robot practices. We used the blue bottle to collect 54 practices using model M_{12}. Table 7 summarizes the results. We can see that the blue bottle's accuracy using model M_{12} resulted in $\mu_e = 26.38$ mL and $\sigma_e = 38.09$ mL. This result outperforms its counterpart when using M_0, where $\mu_e = 55.84$ mL and $\sigma_e = 47.26$ mL. Model M_{12} is more accurate to pour with the blue bottle than M_0. We fine-tuned model M_{12} using only the blue bottle's practices, which

Table 7
Accuracy of pouring water with the blue bottle fine-tuning model M_{12}.

Source container	Base model	Fine-tuned model	μ_e (mL)	σ_e (mL)
	M_{12}		26.38	38.09
Blue bottle	M_{12}	M_{15}[a]	25.12	27.43
	M_{12}	M_{16}[b]	28.27	25.07

[a]Model fine-tuned with robot practices only.
[b]Model fine-tuned with robot practices plus human demonstrations.

resulted in model M_{15}. Its improvement was marginal around 4.7% (25.12 mL from 26.38 mL). We also used the combination of the human demonstrations plus the wine bottle practices plus the blue bottle practices to fine-tune M_{12}. This resulted in model M_{16}. We could see that there was no improvement in the average error. Therefore, we believe that the most reliable model to fine-tune is the one that comes from human demonstrations, i.e., M_0. We hypothesize that such model has learned the variations inherent to humans. Therefore, it has more information than a fine-tuned model with the robot practices.

5.5. Evaluation summary

Overall, the experiments and evaluation results show that

1. The model trained on data of humans pouring water pours accurately using accustomed containers not seen during training. The mean volume error range of pouring water was from 4.12 mL to 12.35 mL for such containers.
2. The model also generalizes the accurate pouring behavior to liquids such as oil and syrup and solid materials such as rice and beans.
3. By using robot practices as new training data, GSSP lowers the initial pouring error to values smaller than the state-of-the-art. For instance, experiments with a wine bottle showed a reduction of mean volume error from 51.22 mL to 15.78 mL (69% reduction).
4. Batch or gradual fine-tuning yield similar results when there is enough variation on the practices collected.
5. Applying GSSP with the combination of human demonstrations and robot practices does not improve the accurate pouring generalization results.

6. Conclusion

In this work, we presented a self-supervised learning from demonstrations approach that allows robots to pour as accurately and fast as humans. The presented work is based on a peephole LSTM that learns the motion dynamics by using the actual outcome of the demonstrations, regardless of an expert human execution. We evaluated the model using a robotic system that we devised and a UR5e robotic arm.[1] Based on the extensive experiments carried out, the presented model pours more accurately and faster than related works that have approached the accurate pouring problem. The capability of the model was further expanded with generalization by self-supervised practicing (GSSP) for containers and materials that presented high pouring error.

Declaration of competing interest

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

[1] Videos of the robotic pouring demos can be found at:
https://youtu.be/MYfZBiHTDBc,
https://youtu.be/u4OyQeMbwsQ, and
https://www.youtube.com/watch?v=xp9nEDTntU4.

References

[1] D. Paulius, Y. Huang, J. Meloncon, Y. Sun, Manipulation motion taxonomy and coding for robots, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 5596–5601.

[2] A. Billard, S. Calinon, R. Dillmann, S. Schaal, Robot programming by demonstration, in: B. Siciliano, O. Khatib (Eds.), Handbook of Robotics, Springer, Secaucus, NJ, USA, 2008, pp. 1371–1394.

[3] Y. Huang, Y. Sun, A dataset of daily interactive manipulation, Int. J. Robot. Res. 38 (8) (2019) 879–886.

[4] C. Schenck, D. Fox, Visual closed-loop control for pouring liquids, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 2629–2636, http://dx.doi.org/10.1109/ICRA.2017.7989307.

[5] M. Kennedy, K. Schmeckpeper, D. Thakur, C. Jiang, V. Kumar, K. Daniilidis, Autonomous precision pouring from unknown containers, IEEE Robot. Autom. Lett. 4 (3) (2019) 2317–2324, http://dx.doi.org/10.1109/LRA.2019.2902075.

[6] C. Do, W. Burgard, Accurate pouring with an autonomous robot using an RGB-D camera, in: International Conference on Intelligent Autonomous Systems, Springer, 2018, pp. 210–221.

[7] C. Do, C. Gordillo, W. Burgard, Learning to pour using deep deterministic policy gradients, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 3074–3079, http://dx.doi.org/10.1109/IROS.2018.8593654.

[8] T. Chen, Y. Huang, Y. Sun, Accurate pouring using model predictive control enabled by recurrent neural network, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 7688–7694.

[9] S. Calinon, A. Billard, A probabilistic programming by demonstration framework handling constraints in joint space and task space, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 367–372, http://dx.doi.org/10.1109/IROS.2008.4650593.

[10] B. Lim, S. Ra, F.C. Park, Movement primitives, principal component analysis, and the efficient generation of natural motions, in: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005, pp. 4630–4635, http://dx.doi.org/10.1109/ROBOT.2005.1570834.

[11] J. Min, Y.-L. Chen, J. Chai, Interactive generation of human animation with deformable motion models, ACM Trans. Graph. 29 (1) (2009) 9:1–9:12, http://dx.doi.org/10.1145/1640443.1640452, URL http://doi.acm.org/10.1145/1640443.1640452.

[12] J.O. Ramsay, G. Hooker, S. Graves, Functional Data Analysis with R and Matlab, Springer, 2009.

[13] W. Dai, Y. Sun, X. Qian, Functional analysis of grasping motion, in: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE, 2013, pp. 3507–3513.

[14] Y. Huang, Y. Sun, Generating manipulation trajectory using motion harmonics, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 4949–4954, http://dx.doi.org/10.1109/IROS.2015.7354073.

[15] D. Paulius, Y. Huang, R. Milton, W.D. Buchanan, J. Sam, Y. Sun, Functional object-oriented network for manipulation learning, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 2655–2662, http://dx.doi.org/10.1109/IROS.2016.7759413.

[16] M. Hersch, F. Guenter, S. Calinon, A. Billard, Dynamical system modulation for robot learning via kinesthetic demonstrations, IEEE Trans. Robot. 24 (6) (2008) 1463–1467.

[17] A.J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal, Dynamical movement primitives: Learning attractor models for motor behaviors, Neural Comput. 25 (2) (2013) 328–373, http://dx.doi.org/10.1162/NECO_a_00393.

[18] H.B. Amor, G. Neumann, S. Kamthe, O. Kroemer, J. Peters, Interaction primitives for human-robot cooperation tasks, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 2831–2837, http://dx.doi.org/10.1109/ICRA.2014.6907265.

[19] A. Paraschos, C. Daniel, J.R. Peters, G. Neumann, Probabilistic movement primitives, in: C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26, Curran Associates, Inc., 2013, pp. 2616–2624, URL http://papers.nips.cc/paper/5177-probabilistic-movement-primitives.pdf.

[20] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, 26 (1) (1978) 43–49.

[21] M. Han, Z. Shi, W. Wang, Modeling dynamic system by recurrent neural network with state variables, in: Advances in Neural Networks - ISNN 2004, 2004.

[22] A.P. Trischler, G.M. D'Eleuterio, Synthesis of recurrent neural networks for dynamical system simulation, Neural Netw. 80 (2016) 67–78.

[23] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.

[24] A. Graves, Generating sequences with recurrent neural networks, 2013, CoRR abs/1308.0850. URL http://arxiv.org/abs/1308.0850.

[25] Y. Huang, Y. Sun, Learning to pour, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 7005–7010, http://dx.doi.org/10.1109/IROS.2017.8206626.

[26] Z. Pan, C. Park, D. Manocha, Robot motion planning for pouring liquids, in: Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling, in: ICAPS'16, AAAI Press, 2016, pp. 518–526, URL http://dl.acm.org/citation.cfm?id=3038594.3038659.

[27] Z. Pan, D. Manocha, Feedback motion planning for liquid pouring using supervised learning, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1252–1259.

[28] M. Tamosiunaite, B. Nemec, A. Ude, F. Wörgötter, Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives, Robot. Auton. Syst. 59 (11) (2011) 910–922, http://dx.doi.org/10.1016/j.robot.2011.07.004, URL http://www.sciencedirect.com/science/article/pii/S0921889011001254.

[29] L. Rozo, P. Jiménez, C. Torras, Force-based robot learning of pouring skills using parametric hidden Markov models, in: 9th International Workshop on Robot Motion and Control, 2013, pp. 227–232, http://dx.doi.org/10.1109/RoMoCo.2013.6614613.

[30] K. Yano, M. Kaneko, Y. Noda, K. Terashima, Supervisory control of automatic pouring process considering residual pouring quantity, in: 2007 European Control Conference (ECC), 2007, pp. 2045–2050, http://dx.doi.org/10.23919/ECC.2007.7068564.

[31] Y. Noda, K. Terashima, Predictive filling weight sequence control in automatic pouring system, in: 2008 34th Annual Conference of IEEE Industrial Electronics, 2008, pp. 1527–1532, http://dx.doi.org/10.1109/IECON.2008.4758180.

[32] A. Yamaguchi, C.G. Atkeson, T. Ogasawara, Pouring skills with planning and learning modeled from human demonstrations, Int. J. Humanoid Robot. 12 (03) (2015) 1550030, http://dx.doi.org/10.1142/S0219843615500309, arXiv:https://doi.org/10.1142/S0219843615500309.

[33] H. Liang, S. Li, X. Ma, N. Hendrich, T. Gerkmann, F. Sun, J. Zhang, Making sense of audio vibration for liquid height estimation in robotic pouring, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 5333–5339, http://dx.doi.org/10.1109/IROS40897.2019.8968303.

[34] M. Kennedy, K. Queen, D. Thakur, K. Daniilidis, V. Kumar, Precise dispensing of liquids using visual feedback, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1260–1266, http://dx.doi.org/10.1109/IROS.2017.8202301.

[35] Y. Huang, Y. Sun, Accurate robotic pouring for serving drinks, 2019, arXiv preprint arXiv:1906.12264.

[36] J. Wilches, Y. Huang, Y. Sun, Generalizing learned manipulation skills in practice, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020, pp. 9322–9328.

[37] B. Neyshabur, S. Bhojanapalli, D. Mcallester, N. Srebro, Exploring generalization in deep learning, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 5947–5956, URL http://papers.nips.cc/paper/7176-exploring-generalization-in-deep-learning.pdf.

[38] L. Wu, Z. Zhu, W. E, Towards understanding generalization of deep learning: Perspective of loss landscapes, 2017, CoRR abs/1706.10239. arXiv:1706.10239. URL http://arxiv.org/abs/1706.10239.

[39] L. Dinh, R. Pascanu, S. Bengio, Y. Bengio, Sharp minima can generalize for deep nets, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, in: ICML'17, JMLR.org, 2017, pp. 1019–1028, URL http://dl.acm.org/citation.cfm?id=3305381.3305487.

[40] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560, http://dx.doi.org/10.1109/5.58337.

[41] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, Trans. Neural Netw. 5 (2) (1994) 157–166, http://dx.doi.org/10.1109/72.279181.

[42] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[43] F.A. Gers, N.N. Schraudolph, J. Schmidhuber, Learning precise timing with lstm recurrent networks, J. Mach. Learn. Res. 3 (2003) 115–143, http://dx.doi.org/10.1162/153244303768966139.

[44] Y. Huang, M. Bianchi, M. Liarokapis, Y. Sun, Recent data sets on object manipulation: A survey, Big Data 4 (4) (2016) 197–216.

[45] W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization, 2014, CoRR abs/1409.2329. URL http://arxiv.org/abs/1409.2329.

[46] A. Zeng, S. Song, J. Lee, A. Rodriguez, T. Funkhouser, Tossingbot: Learning to throw arbitrary objects with residual physics, IEEE Trans. Robot. (2020).

[47] A.B. Jelodar, M.S. Salekin, Y. Sun, Identifying object states in cooking-related images, 2018, arXiv preprint arXiv:1805.06956.

[48] A.B. Jelodar, Y. Sun, Joint object and state recognition using language knowledge, in: 2019 IEEE International Conference on Image Processing (ICIP), IEEE, 2019, pp. 3352–3356.

[49] D. Paulius, N. Eales, Y. Sun, A motion taxonomy for manipulation embedding, in: Robotics: Science and Systems 2020 (RSS), 2020.

[50] M. Alibayev, D. Paulius, Y. Sun, Estimating motion codes from demonstration videos, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[51] T. Kiyokawa, M. Ding, G.A.G. Ricardez, J. Takamatsu, T. Ogasawara, Generation of a tactile-based pouring motion using fingertip force sensors, in: 2019 IEEE/SICE International Symposium on System Integration (SII), 2019, pp. 669–674, http://dx.doi.org/10.1109/SII.2019.8700365.
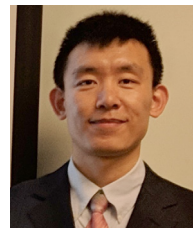
Dr. Yongqiang Huang received his B.Eng. degree from the Beijing University of Posts and Telecommunications in 2010. He also received his M.S. degree from the North Carolina State University. Since August 2013, he was a Ph.D. student in the Department of Computer Science and Engineering at the University of South Florida and a graduate researcher in the Robot Perception and Action Lab (RPAL). He received his Ph.D. degree from the University of South Florida in Spring 2019. His research interests include robotics, machine learning, and deep learning.

Juan Wilches received his B.S. degree from the Universidad del Norte, Barranquilla, Colombia in 2009. He also received his M.S. degree from the same university in 2015. Since January 2019, he is a Ph.D. student in the Department of Computer Science and Engineering at the University of South Florida and a graduate researcher in the Robot Perception and Action Lab (RPAL). His research interests include robotics, machine learning, and deep learning.
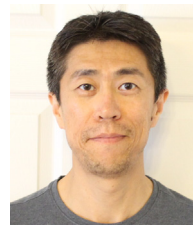
Dr. Yu Sun is an Associate Professor and Associate Chair of Graduate Affairs in the Department of Computer Science and Engineering at the University of South Florida. He was a Visiting Associate Professor at Stanford University. He was the recipient of the 2018 USF Excellence in Innovation Award. He has served as reviewer/panelist for the U.S. National Science Foundation, the French National Research Agency, the Kingdom of Saudi Arabia RDO, and IEEE Admission& Advancement (A&A). He has served as Senior Editor or Associate Editor for several journals and conferences such as IEEE Transactions on Robotics, ICRA, and IROS. He has published numerous papers and holds nine US patents in intelligent systems, robotics, deep learning, computer vision, virtual reality, and medical applications.