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

Learning to Pour

Yongqiang Huang and Yu Sun

Abstract— **Pouring is a simple task people perform daily. It is the second most frequently executed motion in cooking scenarios, after pick-and-place. We present a pouring trajectory generation approach, which uses force feedback from the cup to determine the future velocity of pouring. The approach uses recurrent neural networks as its building blocks. We collected the pouring demonstrations which we used for training. To test our approach in simulation, we also created and trained a force estimation system. The simulated experiments show that the system is able to generalize to single unseen element of the pouring characteristics.**

I. Introduction

Each time a daily manipulation is performed by humans, its execution is adjusted according to the environment and is different from last time. To make robots more widely useful, researchers have been trying to help robots learn a task and generalize to different situations, to which the approach of teaching robots by providing examples has received considerable attention, known as programming by demonstration (PbD) [1].

Among many manipulation tasks, pouring is the second most frequently executed motion in cooking scenarios after pick-and-place [2], [3], [4]. It relies on rotating a cup (or a container in general) that holds certain material. Sufficient rotation of the cup makes the material come out and sufficient recovery makes the pouring stop. Humans typically pour using vision feedback as well as force feedback. However, since the two senses are correlated in pouring, in this paper, we only use force feedback which is easier to define.

There are many existing motion trajectory generation frameworks. A popular one is called dynamical movement primitives (DMP) [5]. DMP is a stable non-linear dynamical system, and is capable of modeling discrete movement such as swinging a tennis racket [6], playing table tennis [7] as well as rhythmic movement such as drumming [8] and walking [9]. DMP consists of a non-linear forcing function, a canonical system and a transformation system. The forcing function defines the desired task trajectory.

Another approach for motion generation is based on Gaussian mixture model (GMM) and Gaussian mixture regression (GMR) [10]. GMM is used to model the trajectories of a task and GMR is used for task reproduction. GMM is learned using all the variables of a movement including time stamps, and GMR is conducted by inferring the movement variables using the learned GMM conditioned on the time stamp.

Principal Component Analysis (PCA) also proves useful for motion generation. Known as a dimension reduction technique used on the dimensionality axis of the data, PCA can be used on the time axis of motion trajectories instead to retrieve geometric variations [11]. Besides, PCA can be applied to find variations in how the motion progresses in time, which, combined with the variations in geometry enables generating motions with more flexibility [12]. Functional PCA (fPCA) extends PCA by introducing continuous-time basis functions and treating trajectories as functions instead of collections of points [13], [14]. [15] applies fPCA for producing trajectories of gross motion such as answering phone and punching, and for making the trajectories avoid obstacles with the guidance of quality via points. [2] uses fPCA for generating trajectories of fine motion such as pouring.

Recurrent neural networks (RNN) recently received increasing attention. At any time step, RNN takes a given input and the output emitted from the last time step, and emits an output which is passed to the next time step. The mechanism of RNN makes it inherently suitable for handling sequential data. Similar to DMP [5] and GMR based approach [16], RNN is also capable of modeling general dynamical systems [17], [18]. RNN can be readily used to generate trajectories by relating the emitted output to future inputs. For example, [19] generates English hand writing trajectories by predicting the location offset of the tip of the pen and the end of a stroke. As more manipulation datasets become available [20], it become feasible to learn a deep RNN.

The paper goes as follows. In Section II, we review the fundamentals of RNN and particularly LSTM, and present our pouring system. In Section III, we describe the data collection and preparation process, training the system, and creating and training a separate force estimation system. In Section IV, we conduct experiments to evaluate whether our system generalizes to unseen situations. We discuss the performance of our pouring system in Section V.

II. Methodology for Pouring Trajectory Generation

In this section, we describe in detail our system of generating a pouring trajectory which builds on long short-term memory. To explain why we choose RNN as the building block, prior to the system description, we review the basics of traditional RNN, and of one particular structure, the long short-term memory.

A. Recurrent Neural Network

Recurrent neural network (RNN) conducts its computation one step at a time, and at any step its input consists of two parts: a given input, and its own output from the previous

The authors are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA. Email: yongqiang@mail.usf.edu, yusun@cse.usf.edu

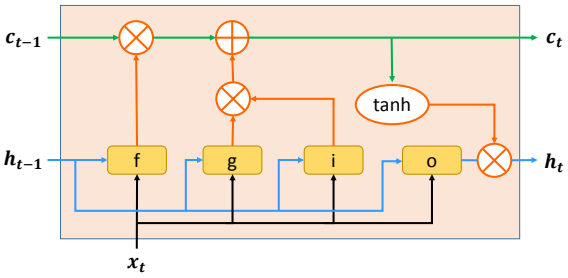

Fig. 1. Mechanism inside an LSTM unit, Zaremba's version [24]

time step. The idea is shown in Eq. (1) where x_t is the given input, h_{t-1} and h_t are output from the previous and at the current step. The weight W and bias b can be learned using Backpropagation Through Time [21].

$$h_t = \tanh\left(W[h_{t-1}, x_t]^\top + b\right) \tag{1}$$

In theory, by including its past output in its input, RNN takes the entire history of given inputs into account when it conducts computation, and therefore is inherently suitable for handling sequential data. However, the traditional RNN as shown in Eq. (1) is difficult to train and has vanishing gradients problem, and therefore is inadequate for problems involving long-term dependency [22], [23]. Long short-term memory (LSTM) is a specific RNN design that overcomes the vanishing gradient problem [23]. We use a version of LSTM whose working mechanism is described by [24]:

$$i = \text{sigm}\left(W_i[h_{t-1}, x_t]^\top + b_i\right) \tag{2}$$
$$o = \text{sigm}\left(W_o[h_{t-1}, x_t]^\top + b_o\right) \tag{3}$$
$$f = \text{sigm}\left(W_f[h_{t-1}, x_t]^\top + b_f\right) \tag{4}$$
$$g = \tanh\left(W_g[h_{t-1}, x_t]^\top + b_g\right) \tag{5}$$
$$c_t = f \odot c_{t-1} + i \odot g \tag{6}$$
$$h_t = o \odot \tanh(c_t) \tag{7}$$

where i, o, f are the input, output, and forget gates respectively, c is the cell, sigm is short for sigmoid, and $\odot$ represent element-wise multiplication. Fig. 1 gives an illustration.

We identify RNN, and specifically LSTM, as the architecture with which we build our pouring system. The reasons include:

1) The structure of RNN makes it inherently fit for handling sequences.
2) RNN is capable of modeling dynamical systems. Since a dynamical system is powered by velocity (or acceleration), it has the ability to react to changes of the environment.
3) RNN has proven ability to generate both categorical and continuous-valued sequences.
4) RNN eliminates the needs for temporally aligning sequences before modeling, and therefore preserves the dynamics in a sequence.
5) LSTM supercedes the traditional RNN, and has proven ability to handle long-term dependency.

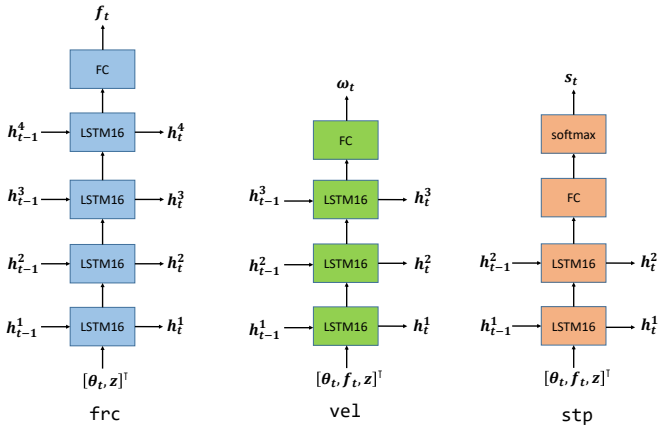

Fig. 2. The architectures of (left) `frc`, (middle) `vel`, (right) `stp`. LSTM16 refers to 16 LSTM units. FC refers to fully connected.

B. Generating Pouring Trajectory

The pouring system predicts the velocity of rotation using the force feedback produced by the cup, which is shown as (middle) in Fig. 1.

We assume n trials of pouring motion are available. The data of trial i are represented by $(\theta_{1...T_i}, f_{1...T_i}, z)^{(i)}$, where $\theta_{1...T_i}$ is the sequence of cup rotation, T_i is the sequence length, $f_{1...T_i}$ is the sequence of sensed force, and z represents static data that characterize the trial. For simplicity, we assume θ, f, z are all one-dimensional.

We refer to the system that predicts the velocity of rotation as `vel`. The actual velocity is computed by

$$\omega_t = \theta_{t+1} - \theta_t, \quad t = 1 \dots T_i - 1. \tag{8}$$

At step t, `vel` takes $[\theta_t, f_t, z]^\top$ as input, and generates predicted velocity $\hat{\omega}_t$:

$$\mathbf{h}_t = \text{LSTM}([\theta_t, f_t, z]^\top) \tag{9}$$
$$\hat{\omega}_t = \text{fc}(\mathbf{h}_t) \tag{10}$$

where 'fc' is short for 'fully connected'. The loss is defined using Euclidean distance:

$$L_{\text{vel}} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{T_i - 1}\sum_{t=1}^{T_i-1}(\omega_t^{(i)} - \hat{\omega}_t^{(i)})^2. \tag{11}$$

In order to automatically stop the generation process after the pouring task has completed, we create a stopping system. We refer to the system that stops the pouring motion as `stp` shown as (right) in Fig 2, which is a binary classifier. At step t, `stp` takes $[\theta_t, f_t, z]$ as input, and outputs a 2-vector $\mathbf{r}_t$. We define class 0 as 'continue', and class 1 as 'stop'.

$$\mathbf{h}_t = \text{LSTM}([\theta_t, f_t, z]^\top) \tag{12}$$
$$\mathbf{r}_t = \text{fc}(\mathbf{h}_t) \tag{13}$$
$$\mathbf{s}_t = \text{softmax}(\mathbf{r}_t) \tag{14}$$

Let the target be represented by a trivial one-hot vector $\mathbf{s}'_t = [s'_{t,1}, s'_{t,2}]^\top$, where $s'_{t,1}, s'_{t,2} \in \{0, 1\}$ and $s'_{t,1} + s'_{t,2} = 1$.

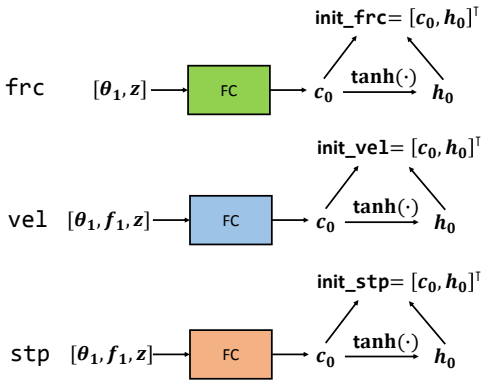

Fig. 3. Initializing `frc`, `vel`, `stp`

The loss is defined using cross entropy:

$$L_{\text{stp}} = -\sum_{i=1}^{n}\sum_{t=1}^{T_i}\left(s'^{(i)}_{t,1}\ln s^{(i)}_{t,1} + s'^{(i)}_{t,2}\ln s^{(i)}_{t,2}\right) \tag{15}$$

The initial state of LSTM includes c_0 and h_0, which are obtained by

$$c_0 = \text{fc}([\theta_1, f_1, z]^\top), \tag{16}$$

$$h_0 = \tanh(c_0), \tag{17}$$

as shown in Fig. 3.

The trajectory is generated by first initializing `vel` and `stp`, and then keep generating and executing rotational velocities. Specifically, the trajectory generation process is described in Alg. 1.

Algorithm 1 Trajectory Generation

Initialize `vel` and `stp` using $[\theta_1, f_1, z]^\top$
$t \leftarrow 1$
while True **do**
  $\omega_t \leftarrow \texttt{vel}([\theta_t, f_t, z]^\top)$
  $\theta_{t+1} \leftarrow \theta_t + \omega_t$
  $s \leftarrow$ argmax $\texttt{stp}([\theta_t, f_t, z]^\top)$
  $t \leftarrow t + 1$
  if $s == 1$ **then**
    Break
  end if
end while

III. Data Preparation and Training

The equipment for data collection includes six different cups, ten different containers, one ATI mini40 force and torque (FT) sensor, and one Polhemus Patriot motion tracker. We refer to the pour-from container as *cup* and the pour-to container as *container*. All cups are mutually different and so are all the containers. The FT sensor records $(f_x, f_y, f_z, \tau_x, \tau_y, \tau_z)$ at 1KHz. The motion tracker records $(x, y, z, \text{yaw}, \text{pitch}, \text{roll})$ at 60Hz. The cup, the force sensor, and the motion tracker are connected by 3D printed adapters, shown in Fig. 4. The materials that are poured include water, beans, and ice.

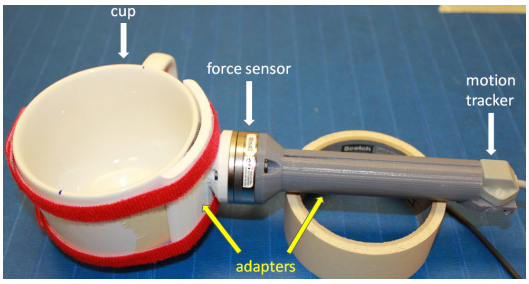

Fig. 4. 3D printed adapters that connect the cup, the force sensor, and the motion tracker.

We obtain the *empty* reading by keeping an empty cup in a level position, taking 500 FT samples (which takes 0.5 second), and then taking the average. Similarly, for each trial, we obtain the *initial* reading right before the trial with material in the cup, and the *final* reading right after the trial with or without material in the cup depending on the trial.

We define the sensed force as

$$f = \sqrt{f_x^2 + f_y^2 + f_z^2}. \tag{18}$$

In total we collected 1,138 trials which involves 3 subjects. Each trial is represented by a sequence $\{\mathbf{a}_t\}_{t=1}^{T_i}$ where $\mathbf{a}_t \in \mathbb{R}^{10}$ and

$\mathbf{a}_t =$
[

θ_t	rotation angle at time t (degree)
f_t	sensed force at time t (lbf)
f_{init}	sensed force before pouring (lbf)
f_{empty}	sensed force while cup is empty (lbf)
f_{final}	sensed force after pouring (lbf)
d_{cup}	diameter of the cup (mm)
h_{cup}	height of the cup (mm)
d_{ctn}	diameter of the container (mm)
h_{ctn}	height of the container (mm)
ρ	material density / water density (unitless)

]

We pad all the sequences to the maximum length in the data: $T_{max} = \max(\{T_i\})$. For `vel`, we pad using zero because zero padding makes it easy to compute the original length of a sequence during training. For `stp`, we pad using the end value of the sequence because `stp` is intended to be used on generated motions which will not have zero padding.

In this work we aim to learn and test the system's ability to generalize to unseen pouring situations. Therefore, we extract certain pouring situations from the data and use them as the test set (Sec. IV provides the list of those situations). We shuffle the rest of the data, which exclude those pouring situations, using a fixed seed for the random number generator. Then we use the first 80% of the shuffled data for training and the rest 20% for validation. For training and validation, the system applies Alg. 1 which uses the force available in the data. For testing, the system applies Alg. 2 which generates the force by itself.

We train using the Adam optimizer [25] and set the learning rate to 0.01. We trained each system for a fixed number of epochs: 4,000 for `vel`, and 2,000 for `stp`. The training error for `vel` ranges from 0.002 to 0.005 (mm), and the accuracy of `stp` ranges from 0.9 to 0.98.

A. Training Force Estimation

In order to run our approach in simulation, we need to have force feedback after we have arrived at a new rotation. Real force feedback is not applicable in simulation. The movement of the liquid during pouring forms a complex dynamical system and is difficult to calculate analytically. Thus, to get force feedback, we decide to generate the force by ourselves. To that end, we learn from data the mapping relationship from rotation angles to force, and then use the learned model to estimate the force corresponding to current rotation.

Thus, we need to train a new system. We refer to the system that estimates the sensed force from rotation as `frc`, shown as (left) in Fig. 2. At step t, `frc` takes $[\theta_t, z]^\top$ as input, and produces estimated force $\hat{f}_t$:

$$\mathbf{h}_t = \text{LSTM}([\theta_t, z]^\top) \tag{19}$$

$$\hat{f}_t = \text{fc}(\mathbf{h}_t) \tag{20}$$

The loss is defined using Euclidean distance:

$$L_{\text{frc}} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{T_i}\sum_{t=1}^{T_i}(f_t^{(i)} - \hat{f}_t^{(i)})^2. \tag{21}$$

The initialization of `frc` includes

$$c_0 = \text{fc}([\theta_1, z]^\top), \tag{22}$$

$$h_0 = \tanh(c_0), \tag{23}$$

as shown in Fig. 3.

The data preparation for `frc` uses zero padding. We train the `frc` with a fixed 2000 epochs, and the error ranges between 0.002 to 0.003 (lbf).

With `frc`, the trajectory generation process needs modification. Force can no longer be assumed to be available, but must be produced explicitly by `frc`. The modified trajectory generation process is shown in Alg. 2.

Algorithm 2 Trajectory generation for simulation

Initialize `frc` using $[\theta_1, z]^\top$
$f_1 \leftarrow$ `frc`$([\theta_1, z]^\top)$
Initialize `vel` and `stp` using $[\theta_1, f_1, z]^\top$
$t \leftarrow 1$
while $t < T_{max}$ **do**
  $\omega_t \leftarrow$ `vel`$([\theta_t, f_t, z]^\top)$
  $\theta_{t+1} \leftarrow \theta_t + \omega_t$
  $s \leftarrow$ argmax `stp`$([\theta_t, f_t, z]^\top)$
  $f_{t+1} \leftarrow$ `frc`$([\theta_{t+1}, z]^\top)$
  $t \leftarrow t + 1$
  if $s == 1$ **then**
    Break
  end if
end while

IV. Experiment on Generalization

We evaluate the generalization ability of the system and see if it can generate pouring motion in unseen situations. Given a test sequence, we extract θ_1 and z, and generate a sequence using Alg. 2. The evaluation is conducted in simulation.

We test the system using unseen

1) cup,
2) container,
3) material,
4) cup and container,
5) container and material,
6) cup and material,
7) cup and container and material.

A. Identifying success

We evaluate the generalization ability of the pouring system using dynamic time warping (DTW) [26], which gives the minimum normalized distance between two trajectories.

We provide a set of test sequence which include an element that is unseen during training and see if the system is able to adapt to the changes. Let the set of test sequences be $\{x_i\}_{i=1}^m$. We first compute the distance between each pair of test sequences and draw a histogram:

$$h_1 = \text{hist}(\{\text{dtw}(x_i, x_j)\}_{i \neq j}) \quad i, j = 1, 2, \ldots, m. \tag{24}$$

Each x_i can be used to generate a new trajectory x_i'. We compute the distance between x_i' and every test sequence x_j and draw another histogram.

$$h_2 = \text{hist}(\{\text{dtw}(x_i', x_j)\}) \quad i, j = 1, 2, \ldots, m. \tag{25}$$

Both histograms are normalized. We visually compare the similarity between h_1 and h_2. If they are similar, then it means the generated trajectories are similar to the trajectories executed by humans, which identifies that the generalization succeeds. The system fails to generalize if otherwise.

B. Results

The results for the seven cases of unseen elements of the pouring characteristics are shown in Fig. 5 to 11. Generalization on cup, or container, or material alone is successful because the pairing histograms are similar (Fig. 5, 6 and 7). Generalizing on cup and container (Fig. 8) and container and material (Fig. 9) can be considered successful because of the similarity in the concentration of the small-distances, despite the difference on mid to high-valued distance parts, which occupy only a small portion of all the distances. Generalizing on cup and material fails as well as on cup and container and materials, as shown in Fig. 10 and Fig. 11. For cup and container and material, only 8 test sequences are available, which may partly contribute to the difference between the two histograms.

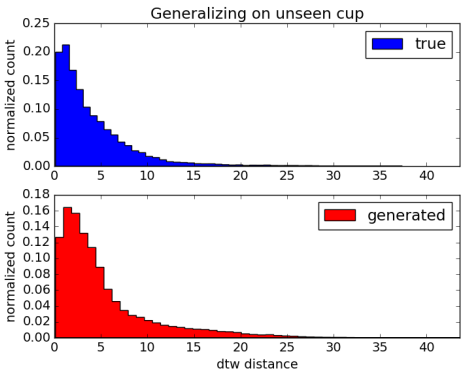

Fig. 5. Generalizing on an unseen cup.

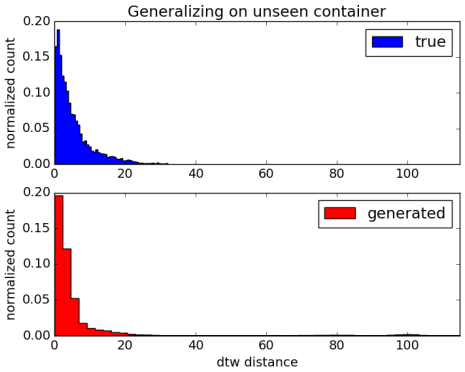

Fig. 6. Generalizing on an unseen container

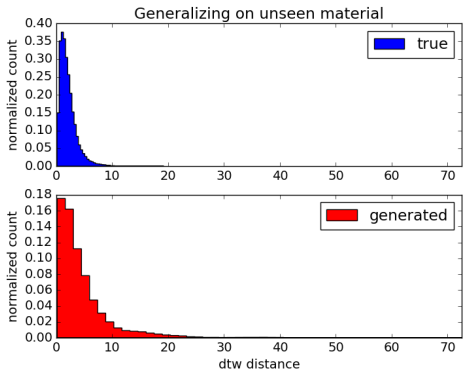

Fig. 7. Generalizing on an unseen material

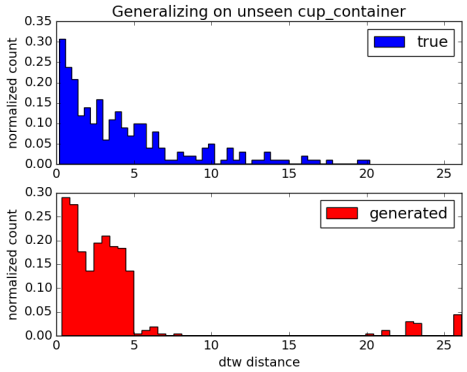

Fig. 8. Generalizing on an unseen cup and container

V. Discussion

We have presented an approach of generating pouring trajectories by learning from pouring motions demonstrated by human subjects. The approach uses force feedback from the cup to determine the future velocity of pouring. We aim to make the system generalize its learned knowledge to unseen situations. The system successfully generalize when either a cup, a container, or the material changes, and starts to stumble when changes of more than one element are present. Since the total size of data does not change, the more that is left out for testing (more unseen elements), the less there is available for training. Thus, the system accepts weaker training and after which faces more demanding challenges. The observed results of degrading performance with increasing generalization difficulty is expected.

We have started evaluating the system on an industrial robot that is equipped with a force sensor. The evaluation is still under way. Future work includes finishing the evaluation on the industrial robot, designing a quantitative measure that measures the degree of success of a generated trajectory, which could be similar to the distance measures in [27], [28]. We also plan to modify the architecture to emphasize the role of initial and final force, and getting help from reinforcement learning.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1421418 and No. 1560761.

References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, *Robot Programming by Demonstration*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1371–1394.

[2] D. Paulius, Y. Huang, R. Milton, W. D. Buchanan, J. Sam, and Y. Sun, "Functional object-oriented network for manipulation learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 2655–2662.

[3] Y. Sun, S. Ren, and Y. Lin, "Object–object interaction affordance learning," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 487–496, 2014.

[4] S. Ren and Y. Sun, "Human-object-object-interaction affordance," in *Robot Vision (WORV), 2013 IEEE Workshop on*. IEEE, 2013, pp. 1–6.

[5] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2 2013.

[6] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, 2002, pp. 1398–1403.

[7] J. Kober, K. Mlling, O. Krmer, C. H. Lampert, B. Schlkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 853–858.

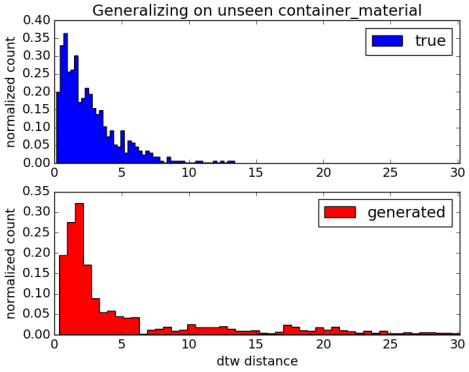

Fig. 9. Generalizing on an unseen container and material

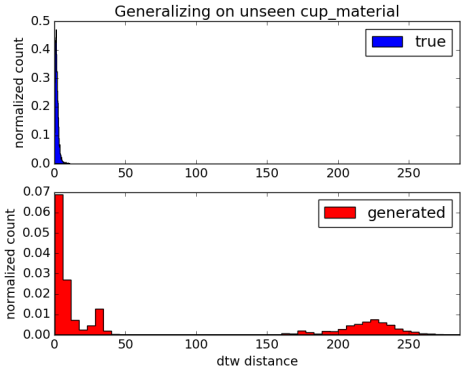

Fig. 10. Generalizing on an unseen cup and material

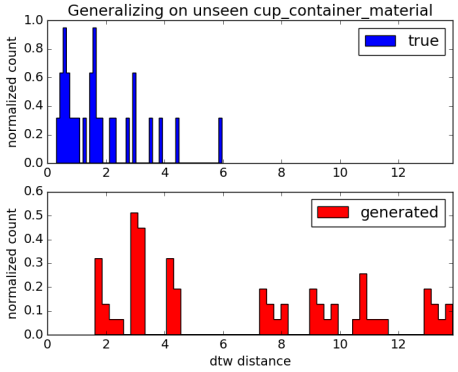

Fig. 11. Generalizing on an unseen cup, container, and material

[8] S. Schaal, "Movement planning and imitation by shaping nonlinear attractors," in *Proceedings of the 12th Yale Workshop on Adaptive and Learning Systems*, Yale University, New Haven, CT, 2003. [Online]. Available: http://www-clmc.usc.edu/publications/S/schaal-YWALS2003.pdf

[9] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, "Learning from demonstration and adaptation of biped locomotion," vol. 47, no. 2-3, pp. 79–91, 2004. [Online]. Available: http://www-clmc.usc.edu/publications/N/nakanishi-RAS2004.pdf

[10] S. Calinon, *Robot Programming by Demonstration: A Probabilistic Approach*. EPFL/CRC Press, 2009.

[11] B. Lim, S. Ra, and 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*, April 2005, pp. 4630–4635.

[12] J. Min, Y.-L. Chen, and J. Chai, "Interactive generation of human animation with deformable motion models," *ACM Trans. Graph.*, vol. 29, no. 1, pp. 9:1–9:12, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1640443.1640452

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

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

[15] Y. Huang and Y. Sun, "Generating manipulation trajectory using motion harmonics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 4949–4954.

[16] S. Calinon, F. D'halluin, D. G. Caldwell, and A. G. Billard, "Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework," in *2009 9th IEEE-RAS International Conference on Humanoid Robots*, Dec 2009, pp. 582–588.

[17] M. Han, Z. Shi, and W. Wang, *Modeling Dynamic System by Recurrent Neural Network with State Variables*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 200–205.

[18] A. P. Trischler and G. M. DEleuterio, "Synthesis of recurrent neural networks for dynamical system simulation," *Neural Networks*, vol. 80, pp. 67 – 78, 2016. [Online]. Available: //www.sciencedirect.com/science/article/pii/S0893608016300314

[19] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[20] Y. Huang, M. Bianchi, M. Liarokapis, and Y. Sun, "Recent data sets on object manipulation: A survey," *Big data*, vol. 4, no. 4, pp. 197–216, 2016.

[21] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990.

[22] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1109/72.279181

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[26] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb 1978.

[27] Y. Lin and Y. Sun, "Robot grasp planning based on demonstrated grasp strategies," *The International Journal of Robotics Research*, vol. 34, no. 1, pp. 26–42, 2015.

[28] ——, "Grasp mapping using locality preserving projections and knn regression," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1076–1081.