Accurate Pouring using Model Predictive Control Enabled by Recurrent Neural Network

Tianze Chen, Yongqiang Huang, and Yu Sun

Abstract— **Humans perform the task of pouring often and in which exhibit consistent accuracy regardless of the complicated dynamics of the liquid. Model predictive control (MPC) appears to be a natural candidate solution for the task of accurate pouring considering its wide use in industrial applications. However, MPC requires the model of the system in question. Since an accurate model of the liquid dynamics is difficult to obtain, the usefulness of MPC for the pouring task is uncertain. In this work, we model the dynamics of water using a recurrent neural network (RNN), which enables the use of MPC for pouring control. We evaluated our RNN-enabled MPC controller using a physical system we made ourselves and averaged a pouring error of 16.4mL over 5 different source containers. We also compared our controller with a baseline switch controller and showed that our controller achieved a much higher accuracy than the baseline controller.**

I. INTRODUCTION

In recent times, researchers are continuously developing robots that can perform human activities such as cooking and cleaning. Among the many activities in cooking, pouring is one of the frequently performed motions in food preparation based on our Functional Object-Oriented Network (FOON) video set (FOON is a knowledge representation for robots) [1], [2]. If the pick-and-place motion in the pouring is counted as a separated motion, pouring is the second most frequently performed motion in the FOON video set after picking-and-placing [1], [3]. Otherwise, pouring is the most frequently performed motion in the FOON video set [4]. Several studies have paid close attention to the pouring motion, and dynamics estimation of liquid plays a key role in successfully performing the pouring motion. In particular, [5] used vision to perceive the level of liquid, [6] used force readings to estimate the trajectory of the water being poured, [7] used fluid dynamics to simplify the pouring process and [8] used motion harmonics to generate manipulation trajectory of pouring by taking dynamics into consideration.

MPC belongs to process control and according to [9] it is the most widely accepted modern control strategy. MPC has been applied to many industrial applications such as the control of precisely pouring molten metal into a container [10]. Due to the sensitivity between optimality and computational speed, traditional MPC approaches, which usually requires offline calculations in order to achieve better performance, are typically time and resource consuming. As a result, an online calculation method for MPC to improve the computational complexity, modifications of the online method [11] has become a trend. Due to the increasing popularity of deep neural networks, researchers have attempted to apply deep learning models into the MPC algorithm. For instance, in [12], the authors combine MPC with reinforcement learning in policy search for autonomous aerial vehicles.

Recurrent neural networks (RNN) have particularly seen an increase of use recently for sequence understanding and generation. RNN is designed to learn temporal patterns within a sequence. It has been used to perform tasks such as successfully synthesize hand writings [13], translate one language to another [14], generate controllers for object pushing and placing [15], and to simulate trajectories of water pouring [6]. It does not suffer from the sequence length alignment problem in other motion generation approaches [16].

In this work, we propose using RNN to model the dynamics of water while it is being poured. With the dynamics of water modeled by the RNN, the outcome of applying different rotation velocities to the source container can be predicted. Using the predictions, MPC can determine the optimal velocity to execute. Our work enables the use of MPC for the task of accurate pouring by using RNN to estimate, rather than analytically deriving the mechanism of flowing water. The accuracy of the RNN-enabled MPC pouring controller depends primarily on the accuracy of the RNN. With our controller, we are able to achieve a novel accuracy compared with the prior works.

In the following sections, we first present our proposed pouring controller from individual components to fullness. We then show the results of our evaluation of the controller using a physical system we made ourselves. We evaluate the proposed controller by using it to pour water from 4 different source containers and by comparing its pouring accuracy with that of a baseline switch controller.

II. METHODOLOGY

In this section, we present the technical detail of our pouring algorithm. The algorithm bases itself on MPC, within which it uses RNN to predict the outcomes of different control inputs. We will first describe the process of pouring, then we review the basics of both RNN and MPC. After we have covered the individual components of our algorithm, we finally present the algorithm in its entirety.

A. *Problem Description*

We consider the problem of pouring a target amount of water accurately from a source container to a receiving container as illustrated in Fig. 1. In this work, we represent

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

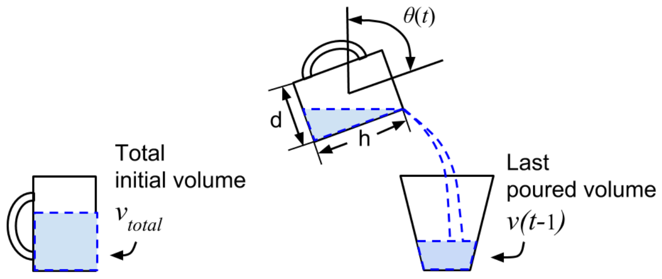

Fig. 1: Illustration of pouring problem.

the amount of the water using volume. The volume of water that ultimately ends up in the receiving container is determined by the continual flow of the water from the source container. To achieve an accurate pour, the source container must stop rotating or reverse at the proper moment. Neither stopping the rotation of the source container or reversing it guarantees instant stopping of the flow of the water, as the water may still keep flowing out for a brief period of time. The water has a delayed response to the rotation of the source container, and to stop its flow at a particular time, the delay must be taken into account. In other words, a pouring algorithm must stop or reverse the rotation of the source container in advance. To do that, the algorithm must be able to predict the behavior of the water in response to the rotation of the source container to certain extent.

B. Model Predictive Control

MPC is a process control method that generates control input sequentially, one time step after another. At any time step, MPC determines the control input by solving an optimization problem. It first looks several time steps into the future and predicts the outcomes of different sequences of control inputs. It then identifies the sequence of control inputs that gives the optimal outcomes and executes the first control input in that sequence. The above process keeps repeating itself until the outcome converges to a desired steady state. The range in which MPC looks into the future and make predictions is referred to as the *prediction horizon*.

The standard MPC algorithm can be represented as below:

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

$$y_k = Cx_k, \tag{2}$$

where x_k, u_k, y_k are the system state, the control input and the outcome respectively. The variables A, B and C refer to constants related to the control system parameters. Each iteration of MPC solves an optimization problem which can be written as:

$$\min \sum_{k=1}^{n} (y_k - \bar{y})^2 \tag{3}$$

with the following restriction:

$$x_k \in state\ range,\ k = 1,2,3,...,n \tag{4}$$

$$u_k \in input\ range,\ k = 1,2,3,...,n \tag{5}$$

where $\bar{y}$ is the steady state, n is the length of prediction horizon. k is the index for each time step within prediction horizon. Equation (6) is called objective function. By solving the optimization problem above, u_1 will be chosen as the input for next time step. The process iterates until the desired steady state is reached.

C. RNN for Volume Prediction

The recurrent neural network (RNN) is a class of neural networks that is designed to process sequential data. RNN differs from ordinary neural networks in that its computation involves the notion of order or time. At any time step t, the input to RNN includes two entities: an actual input from the outside $x(t)$, and the output of the RNN itself from the previous time step $h(t-1)$, i.e.

$$h(t) = f(x(t), h(t-1)), \tag{6}$$

where $f(\cdot)$ represents a non-linear function. The network is trained using back propagation through time [17]. Unfortunately, a RNN implemented exactly as described above cannot effectively process long sequences because the gradients that are used for updating the weights either explode or vanish [18], [19]. One popular solution to the problem is *long short-term memory* (LSTM) [18], which introduces memory cells and gates that therefore allow the network to propagate gradients properly back in time. In this work, we use a particular design of LSTM whose mechanism is described in [20]. Hence, in this paper, all instances of "RNN" refer to the LSTM design proposed in [20].

We use RNN to model the dynamics of the water and to predict the volume of water in the receiving container. We use z to represent static task specifications, which include any factor that can affect the dynamics of the water in a particular task and that also does not change with time. In this work, we determine that $z = \{h, d, v_{total}\}$, where d and h are the diameter and height of the source container, and v_{total} is the total volume of water existing in the source container before the pouring starts.

We determine that the input features to the RNN include the static task specification z, the rotation angle of the source container $\theta(t)$, and the volume from the last time step $v(t-1)$. The computation of the RNN can be written as

$$v(t) = RNN(\theta(t), v(t-1), z) \tag{7}$$

where $v(t)$, the output of the RNN is the prediction of the volume. Let the actual volume be $v'(t)$. The loss function is defined as

$$L_{RNN} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{t=1}^{T_i}(v(t) - v'(t))^2 \tag{8}$$

where N is the number of training sequences, T_i is the length of sequence i.

D. RNN-enabled MPC controller

In order to pour a target volume of water into the receiving container, we use MPC as the control protocol. The system model represented by Eq. (4) and (5) is linear and is incapable of accurately capturing the dynamics of water. Therefore during the pouring process, we model the dynamics of water using RNN. Let n be the length of the prediction horizon, we define the pouring problem as:

$$\min \sum_{i=1}^{n} (v_i - v_{target})^2 \tag{9}$$

with the following restriction:

$$\omega_{min} \leq \mid \frac{\theta_i - \theta_{i-1}}{\Delta t} \mid \leq \omega_{max} \tag{10}$$

where θ_i is the rotation angle of the source container, ω_{min}, ω_{max} are the lower and upper velocity limit for pouring. Δt is a fixed value that represents the time interval between two consecutive time steps. v_i is the volume predicted by the RNN for time step i within the prediction horizon, and v_{target} is the target volume we want to pour (the steady state of volume in the receiving container).

We define two velocities ω_p and ω_r in which ω_p is the pouring velocity and ω_r is the reversing velocity. We define the possible velocity at each time step $j = 1, \ldots, n$ inside the prediction horizon to be either ω_p or ω_r. This produces a total of $m = 2^n$ different sequences of velocities of length n. We denote the velocity of sequence i at time step j by w_{ij}. All of the different sequences of velocities can be represented using a matrix of the form:

$$\Omega = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \cdots & \omega_{1n} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_{m1} & \omega_{m2} & \omega_{m3} & \cdots & \omega_{mn} \end{bmatrix}, \tag{11}$$

where $\omega_{ij} \in \{\omega_p, \omega_r\}$. Figure 2 is an example of generating

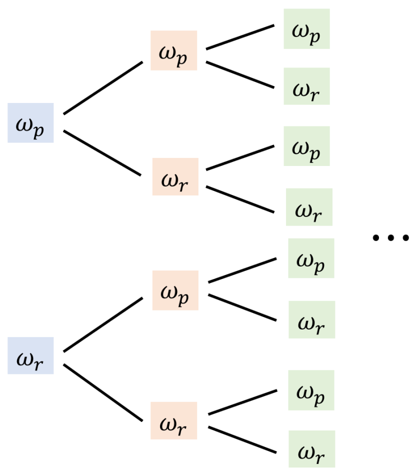

Fig. 2: Example of generating all possible velocity sequences

all the possible velocity sequences. At each time step, only ω_p and ω_r are valid value for the velocity, thus the generated sequences are represented as two binary trees. Here we choose ω_p as the average velocity of the human demonstrated pouring data we collected and we choose ω_r as negative 1.5 times of the value of ω_p. Due to the computational complexity, ω_p and ω_r are scalars. If with higher computational power we can sample more values to make ω_p and ω_r to vectors. Each complete path from the root to a leaf is a velocity sequence, such as $\{\omega_p, \omega_p, \omega_r, \ldots\}$. In our algorithm, we only allow the velocity to change sign once. We trim the sequences that contain an ω_r followed by ω_p. This trimming can furthermore reduce the computational complexity. For instance, if we look into 5 steps into the future, our trimming method can reduce 80% the computation time.

Let θ_{cur} be the current rotation angle of the source container, the sequence of rotation angles corresponding to the sequence of velocities is computed by:

$$\theta_{i0} = \theta_{cur} \tag{12}$$

$$\theta_{ij} = \theta_{i(j-1)} + \omega_{ij}\Delta t \tag{13}$$

where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The resulting rotation angles θ_{ij}'s can be represented by matrix Θ as:

$$\Theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & \ldots & \theta_{1n} \\ \vdots & \vdots & \vdots & & \vdots \\ \theta_{m1} & \theta_{m2} & \theta_{m3} & \ldots & \theta_{mn} \end{bmatrix} \tag{14}$$

The volume v_{ij} for sequence i, time step j can be predicted using RNN by the following:

$$v_{ij} = RNN(\theta_{ij}, v_{i(j-1)}, z) \tag{15}$$

where z is the static task specification that does not change with time. Let the current volume be v_{cur}, then sequence i of volume can be predicted by going through sequence i of rotation angles in time, specifically:

```
1: $v_{i0} = v_{cur}$
2: **for** $j$ in $(1, 2, \ldots, n)$ **do**
3:     $v_{ij} = RNN(\theta_{ij}, v_{i(j-1)}, z)$
4: **end for**
```

where $i = 1, \ldots, m$. The predicted volumes v_{ij}'s can be represented by matrix $\mathbf{V}$.

$$\mathbf{V} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & \ldots & v_{1n} \\ \vdots & \vdots & \vdots & & \vdots \\ v_{m1} & v_{m2} & v_{m3} & \ldots & v_{mn} \end{bmatrix} \tag{16}$$

We define the loss function at time step j using the squared difference between the predicted volume v_{ij} and the target volume v_{target}:

$$l_{ij} = \begin{cases} (v_{ij} - v_{target})^2, & v_{ij} \leq v_{target} \\ \lambda (v_{ij} - v_{target})^2, & v_{ij} > v_{target} \end{cases} \tag{17}$$

where $i = 1, \ldots, m$. Particularly, when over-pour occurs, i.e. when $v_{ij} > v_{target}$, we increase the loss value significantly by multiplying it with a coefficient $\lambda >> 1$. The reason we greatly penalize velocities that cause over-pour is because over-pour is a state from which a pouring system is never able to recover. We define the loss of each sequence to be the sum of the losses for all the time steps in that sequence:

$$L_i = \sum_{j=1}^{n} l_{ij}, \quad i = 1, \ldots, m \tag{18}$$

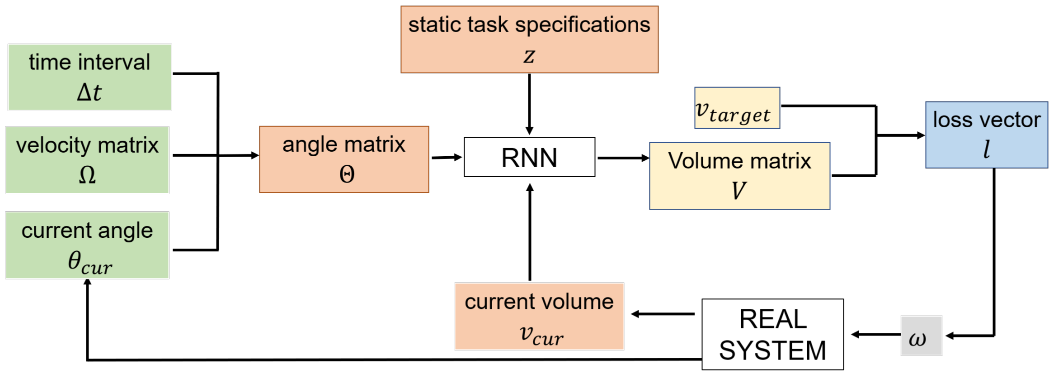

Fig. 3: An illustration of the pipeline of our pouring control algorithm. First, we generate several sequences of velocities and put them into Ω. We then compute the corresponding sequences of angles, which we put in Θ. Using each sequence of angles, we predict the sequence of volume, which we put in V. Using the target volume v_{target}, we compute the loss for each sequence of volume. The index of the minimum-loss sequence is identified and the first velocity in that corresponding sequence of velocities is executed. The real system proceeds to the next time step and the above process is repeated.

The losses for difference sequences L_i's can be grouped into a vector $\mathbf{l}$

$$\mathbf{l} = \begin{bmatrix} L_1 \\ \vdots \\ L_m \end{bmatrix} \tag{19}$$

We find the minimum loss in $\mathbf{l}$

$$i_{min} = \arg\min_i \mathbf{l} \tag{20}$$

The system executes the velocity at the first time step ($j = 1$) in velocity sequence i_{min} and proceeds to next step:

$$\omega = \omega_{i_{min}1}. \tag{21}$$

The above control algorithm is described more concisely in Algorithm 1. We use *flag* to check whether the source container has started to reverse. Once it starts reversing, the control algorithm keeps running for *breakvalue* iterations and then stops generating velocities. Then, the source container returns to $\theta = 0$ at a fixed speed. We also illustrate the algorithm in Fig. 3.

Algorithm 1 Pouring algorithm

```
1:  Initialize $\omega$ and $v_{cur}$
2:  $count \leftarrow 0$
3:  $flag \leftarrow 0$
4:  while $count < breakvalue$ do
5:      Sample the velocity matrix $\Omega$
6:      Calculate the angle matrix $\Theta$ using $\Omega$ and $\Delta t$
7:      Predict the volume matrix $\mathbf{V}$ from $\Theta$ using RNN
8:      Calculate the loss vector $\mathbf{l}$
9:      Find minimum-loss sequence $i_{min} = \arg\min \mathbf{l}$
10:     From $\Omega$, retrieve velocity sequence $i_{min}$, i.e. $[\omega_{i_{min}1}, \omega_{i_{min}2}, \ldots, \omega_{i_{min}n}]$
11:     $\omega \leftarrow \omega_{i_{min}1}$
12:     if source container rotates back then
13:         $flag \leftarrow 1$
14:     end if
15:     if $flag == 1$ then
16:         $count \leftarrow count + 1$
17:     end if
18: end while
```

III. Experiments and Evaluation

In this section, we present our experiment results and evaluate our algorithm's performance. We also compare our algorithm with a simple pour-and-reverse pouring algorithm. We use the mean and standard deviation of the pouring error to represent the accuracy of a trial. We use fixed pouring velocity and reversing velocity. The source container only reverses once, which means it will only reverse when the algorithm generates the reverse velocity ω_r and then afterwards, it keeps executing ω_r. Obtaining the reading of volume is not straightforward without the use of a measuring cup. However, volume can be converted from weight f if the density of the liquid and the gravitational acceleration is known. Thus, in the experiment, we use a force sensor to measure the weight f of the water in the receiving container, and convert the weight f to volume v.

A. Data Collection

We collected 316 trials of human pouring water from different source containers to the same receiving container, which is part of the USF RPAL Daily Interactive Manipulation (DIM) dataset[21], [22]. We used the Polhemus Patriot motion sensor to record the rotation of the source containers and ATI mini40 force sensor to record the weight of the water in the receiving container. We also measured the total weight of water in the source container before the start of each trial, as well as the height and diameter of each source container. The DIM dataset contains recordings of 32 types of manipulations. At the moment, it is the only open manipulation dataset that contains both motion and force [23].

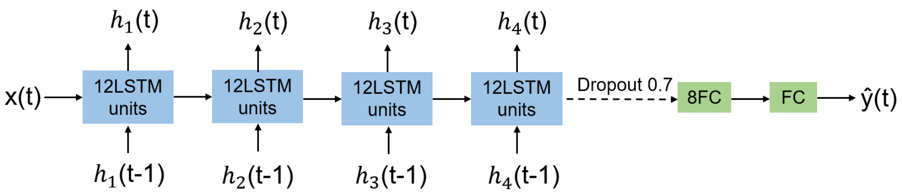

Fig. 4: The structure of the implemented RNN. 8FC refers to fully connected layer with 8 units. Input $x(t) = \{\theta(t), f(t-1), d, h, f_{total}\}$ and output $y(t) = f(t)$. The volume v is represented using weight f.

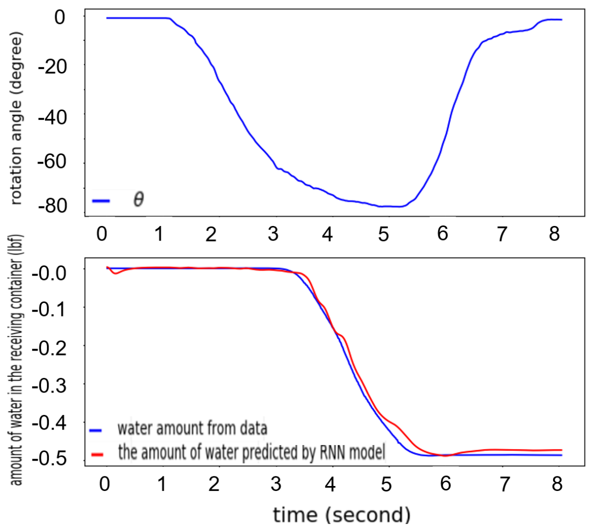

Fig. 5: The model's prediction on one of the test sequence, the force are negative since they are pointing to the gravity direction

B. Training the RNN

The architecture of our RNN is shown in Fig. 4. It has 4 LSTM layers, followed by 2 fully connected (FC) layers. Each LSTM layer has 12 LSTM units. The first FC layer has 8 units and the second FC layer has 1 unit and is the output layer. Dropout is used to the output of the fourth layer, and we choose 0.7 as the dropout rate, which means we will randomly drop 30% of the units in the network. We divide the data into training, validation and test with the percentages of 60%, 20%, 20%. We train the network for 2500 epochs with a batch size of 64. The average test loss on our model is $2.6051523e-05lbf$ which is equalivent to $2.32mL$. One of the testing results generated by our trained RNN model is shown in figure 5.

C. Physical System

To evaluate our pouring algorithm, we made a physical system which consists of a Dynamixel MX-64 servo motor and the same force sensor with which we collected the pouring data. The source container was attached to the motor and the receiving container was placed on top of the force sensor. The algorithm reads force from the force sensor and rotation angle from the motor. The motor executes the velocity issued by the algorithm. The physical system is shown in Fig. 6.

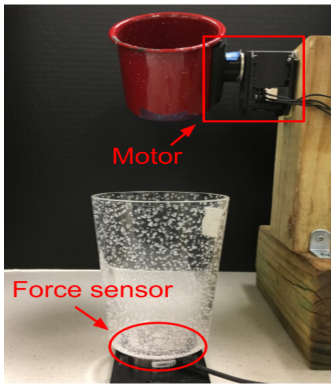

Fig. 6: The physical system mainly consists of a servo motor which executes the velocity and a force sensor which obtain the weight of the water

D. Pouring with the proposed controller

We used the proposed controller to pour water from 5 different source containers. For each source container, we performed 8 trials. The target volume ranges from $40mL$ to $180mL$, with an increment of $20mL$. The lowest observed pouring velocity in our data is 20 deg/sec. Therefore, we specify the pouring velocity $\omega_p = 20$ deg/sec, and the reversing velocity $\omega_r = -30$ deg/sec for the evaluation. We specify the length of the prediction horizon to be 5 due to limited computation capability. The system runs at 30Hz. Among the 5 cups with which we conducted the experiments, only the red cup is used for training the RNN. Table I shows the mean and standard deviation of the pouring error for all cups. As examples, Fig. 7 shows the actually poured volume of each trial for the glass cup and the fat cup.

TABLE I: Results of pouring with the proposed controller

cup	**cup diameter (mm)**	**cup height (mm)**	**in training**	**error mean (mL)**	**error std (mL)**
red	88.0	81.0	Yes	7.25	4.92
glass	80.0	103.0	No	14.25	9.11
bottle	23.0	218.0	No	15.88	5.13
fat	85.0	109.0	No	18.25	8.30
bubble	88.0	121.0	No	26.13	6.29

E. Pouring with the baseline controller

We compare our controller with a baseline controller. The baseline controller is a switch controller that pours with a

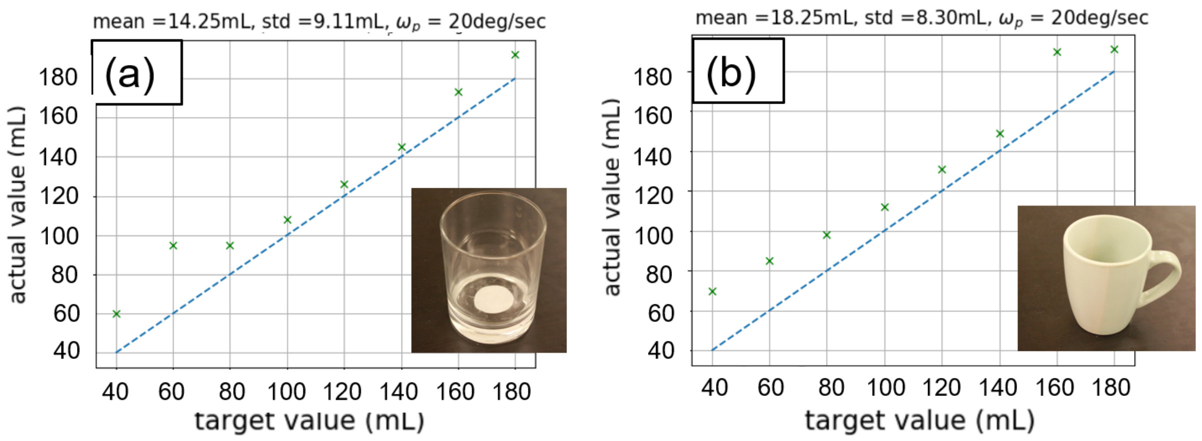

Fig. 7: Results of MPC-based pouring, where: (a) is the result for the glass cup, (b) is the result for the fat cup.

certain pouring velocity until the volume of water in the receiving container reaches the target volume, then the source container is reversed right away using a certain reversing velocity.

We evaluated the accuracy of the baseline controller using two sets of pouring and reversing velocity: set A ($\omega = 5$ deg/sec, $\omega_r = -7.5$ deg/sec) and set B ($\omega = 20$ deg/sec, $\omega_r = -30$ deg/sec). In set A, both the pouring and reversing velocity are very small which makes accurate pouring an easy problem because the speed at which the water flows will be slow. Set B includes the same pouring and reversing velocity as used for our RNN-enabled MPC controller as mentioned in Sec. III-D. With ω_p and ω_r the same, the difference between our proposed controller and the baseline controller set b lies in our controller's ability of looking into the future to prevent overpour. Using each baseline controller, we pour with 2 different source containers: the red cup and the bubble cup since they represent the best and the good results for the proposed controller (RNN+MPC). For each source container, we performed 8 trials. The target volume ranges from $40mL$ to $180mL$, with an increment of $20mL$.

Table II shows the comparison of accuracy between the proposed controller and the 2 baseline controllers. Fig. 8 shows the actual poured volume for each trial conducted by the proposed controller and 2 baseline controllers using the red cup, and Fig. 9 using the bubble cup.

In Table II, Fig. 8 and 9 we can see that the baseline controller set A, i.e. the slow baseline controller always achieves the highest accuracy among the three controllers. This is understandable because as the pouring velocity decreases, so does the difficulty of controlling the pouring volume. The extreme case of a slow baseline controller is one that drips water into the receiving container, for which no difficulty of controlling the pouring volume will be encountered. For a daily pouring task, if the velocity of pouring is too slow then the efficiency of the task will be very low, which disobeys the initial purpose of designing our controller - to learn pouring from human's demonstration. Hence it is important for our controller to perform the whole pouring motion at the average pouring velocity of human. When the baseline

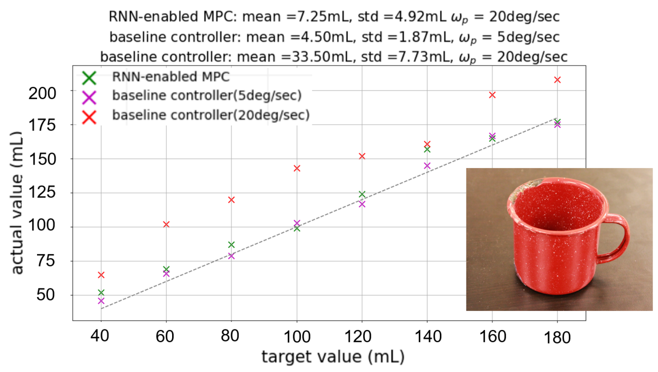

Fig. 8: Accuracy of proposed controller against 2 baseline controllers with the red cup as the source container.

controller adopts the same pouring velocity as the proposed controller, it achieves much lower accuracy than that of the proposed controller.

Our proposed controller managed to achieve a much higher accuracy than baseline controller set b supports our design philosophy that the accuracy of pouring will be improved if the velocity selection process takes into account the future outcomes a velocity may cause.

TABLE II: Accuracy of proposed controller against 2 baseline controller

controller	**cup**	ω_p **(deg/sec)**	**err mean (mL)**	**error std (mL)**
RNN+MPC	red	20	7.25	4.92
baseline	red	20	33.50	7.76
baseline	red	5	4.50	1.87
RNN+MPC	bubble	20	26.13	6.29
baseline	bubble	20	56.25	5.85
baseline	bubble	5	22.25	4.29

F. Limitations

There are two factors that we think influence the accuracy of the pouring. The first factor is the lack of fine velocity control. Our algorithm does not provide a proper decrease of

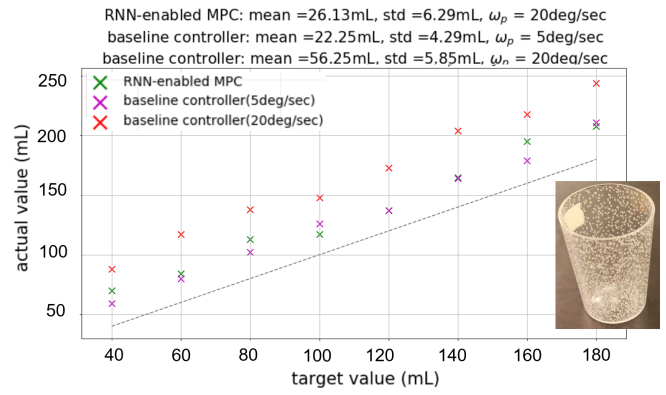

Fig. 9: Accuracy of proposed controller against 2 baseline controllers with the bubble cup as the source container.

the velocity when the source container is about to reverse, which, if exists, is expected to help increase the pouring accuracy. The second factor is the insufficient length of the prediction horizon. Due to the computational complexity, we can only look into 5 steps into the future.

IV. Conclusions

In this work, we present a controller for accurate pouring. The controller bases itself on the MPC algorithm. It uses RNN to model the dynamics of water, with which it is able to predict the outcomes of different velocities applied to the source container, and then determine the optimal rotation velocity to execute. The dynamics modeling RNN plays an essential role in the design of the controller, and its training accuracy affects the accuracy of the controller. The accuracy of our controller relies on the prediction of the RNN model. In another word, our controller will not generalize well if the RNN model doesn't learn well. The next step should be methods on how to train a RNN model that can have good generalization ability.

We evaluated the controller using a physical system that we made ourselves. We used the controller to pour water from 5 different source containers which averaged a pouring error for 16.4 mL. The 5 different source containers represent three commonly used pouring containers in our daily pouring activity: rectangular shape containers(red cup and glass cup), trapezoid shape containers(fat cup and bubble cup) and containers with narrow opening but relatively long body(water bottle), which indicates the good generalization ability of our controller. We then compared our controller with a baseline switch controller and our controller achieved a much higher accuracy than the baseline controller.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1421418, No. 1560761 and No. 1812933.

References

[1] 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.
[2] D. Paulius and Y. Sun, "A survey of knowledge representation in service robotics," *Robotics and Autonomous Systems*, vol. 118, pp. 13–30, 2019.
[3] D. Paulius, A. B. Jelodar, and Y. Sun, "Functional object-oriented network: Construction & expansion," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–7, 2018.
[4] D. Paulius, Y. Huang, J. Meloncon, and Y. Sun, "Manipulation motion taxonomy and coding for robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1–6.
[5] C. Schenck and D. Fox, "Visual closed-loop control for pouring liquids," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 2629–2636.
[6] Y. Huang and Y. Sun, "Learning to pour," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 7005–7010.
[7] Z. Pan and D. Manocha, "Motion planning for fluid manipulation using simplified dynamics," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4224–4231.
[8] Y. Huang and Y. Sun, "Accurate robotic pouring for serving drinks," *CoRR*, vol. abs/1906.12264, 2019. [Online]. Available: http://arxiv.org/abs/1906.12264
[9] B. Kouvaritakis and M. Cannon, *Model Predictive Control: Classical, Robust and Stochastic*, ser. Advanced Textbooks in Control and Signal Processing. Springer International Publishing, 2015. [Online]. Available: https://books.google.com/books?id=DmoiCwAAQBAJ
[10] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733 – 764, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066102001867
[11] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, March 2010.
[12] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 528–535.
[13] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850
[14] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215
[15] R. Rahmatizadeh, P. Abolghasemi, and L. Bölöni, "Learning manipulation trajectories using recurrent neural networks," *CoRR*, vol. abs/1603.03833, 2016. [Online]. Available: http://arxiv.org/abs/1603.03833
[16] Y. Huang and Y. Sun, "Generating manipulation trajectory using motion harmonics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4949–4954.
[17] 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.
[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[19] 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.
[20] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014.
[21] Y. Huang and Y. Sun, "A dataset of daily interactive manipulation," *The International Journal of Robotics Research*, vol. 38, no. 8, pp. 879–886, 2019. [Online]. Available: https://doi.org/10.1177/0278364919849091
[22] "USF RPAL daily interactive manipulation (DIM) dataset," http://rpal.cse.usf.edu/datasets_manipulation.html, accessed: 2019-07-31.
[23] 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.